Politechnika Wrocławska

FIELD OF SCIENCE: Engineering and Technology

DISCIPLINE OF SCIENCE: Information and Communication Technology

DOCTORAL DISSERTATION

“Methods for selected problems in unsupervised graph representation learning”

mgr inż. Piotr Bielak

Supervisor:
dr hab. inż. Tomasz Kajdanowicz, prof. uczelni

Keywords: representation learning, unsupervised learning, graph machine learning

WROCŁAW 2023

„...when you have eliminated the impossible,
whatever remains, however improbable,
must be the truth."
– Sir Arthur Conan Doyle

Acknowledgments

I want to express my sincere gratitude to my supervisor, Tomasz Kajdanowicz, for his help, guidance, shared knowledge, and continuous support during my Ph.D. studies. Thank you for the countless hours spent together looking for solutions and discussing ideas; your patience, motivation and understanding; and all your help with the administrative duties.

I want to thank all my friends and family for your undying support throughout this journey. Especially, I would like to thank Mateusz and Szymon for all the time spent together at this university, including lunches, laughs, and gaming evenings. Thanks also to Kacper for our scientific and casual discussions. I thank my mother and grandmother for always believing in me and for your support in tough times.

Thanks to all my colleagues from room 441 (and 233, later on). I will never forget the wonderful atmosphere, all the fascinating discussions, and the laughs we had in our office. It was a pleasure working with you.

To all of you, from the bottom of my heart, thank you!

Abstract

Machine learning (ML) methods have been studied in a variety of applications and data types. Since most downstream ML models expect a vector from a continuous space as input, representation learning methods have been developed to automatically create representation vectors (embeddings) for the input data. While many embedding methods exist for traditional data types, such as BERT for text or ResNet for images, this task is much more difficult for graph-structured data. Graphs can be used to describe objects (using nodes) and their relationships (using edges). For simple graphs, the main objective of representation learning methods is to capture the graph's structure. Nowadays, graphs contain multiple information sources, i.e., besides the structure (nodes and edges), one can assign attributes to nodes, edges, and even whole graphs (attributed graphs). Deriving appropriate graph representations is important for ML tasks to enable them to perform well, e.g., node classification or link prediction.

In recent years, various graph representation learning methods have been proposed. Despite the success of these methods, the following issues and research gaps are still not covered. First of all, most proposed approaches are inherently transductive (they optimize a fixed-size embedding matrix). Such a setting does not allow obtaining embeddings for previously unseen examples. Moreover, for real-world, large-scale graphs with millions or even billions of nodes, this solution requires an infeasible amount of memory and is impractical as graphs tend to evolve over time. The next major issue is that the (semi-)supervised setting requires labeled data. Obtaining such labels is an expensive and time-consuming task. Unsupervised (and self-supervised) learning has shown that graph representations can also be derived solely from the network structure and attributes. Yet, many proposed approaches were introduced in the semi-supervised setting, i.e., the model is jointly optimized in a particular downstream task. Hence, their expressive power is also limited by the choice of the downstream task and its connected loss function. In the unsupervised setting, the focus is on the only available structural and attribute information. Finally, the last issue concerns the problem of defining negative samples in contrastive loss functions utilized in unsupervised graph representation learning methods. The quality of the embeddings and, consequently, the

performance of downstream tasks highly depend on how the negative samples are defined. Choosing appropriate negative samples is particularly difficult in graphs compared to other data types. Hence, the research community was exploring negative-sample-free methods in computer vision or natural language processing but were not explored for graphs.

The objectives of the doctoral dissertation were: 1) verifying whether the usage of the cross-correlation measure applied to augmented views of an attributed graph allows training a graph neural network in a self-supervised setting, such that this GNN computes better node representation vectors than existing self-supervised approaches, measured by the performance in downstream tasks and training time complexity, 2) developing a deep neural network model for calculating edge representation vectors that is trained using a combination of a contrastive learning objective with a feature reconstruction loss, such that this method's embedding vectors are better in downstream tasks than those obtained as aggregations of source and target node representations, 3) developing an incremental learning method for node representation vectors in dynamic graphs, which utilizes any kind of static node embeddings from consecutive graph snapshots and exhibits a lower time and memory complexity than contemporary methods for representation learning on dynamic graphs while providing competitive quality measure gains, and 4) developing a method for fusing together node attribute information with a given precomputed structural node embedding, resulting in a single low-dimensional embedding that performs better in downstream tasks than: the structural embedding, the node attributes or other attributed node representation learning methods.

The dissertation is a collection of thematically related works in the form of five scientific publications centered around the topic of unsupervised representation learning methods for graphs.

The first publication (*Graph Barlow Twins: A self-supervised representation learning framework for graphs*) proposes a novel framework for self-supervised representation learning of nodes in attributed graphs, which utilizes the empirical cross-correlation matrix between embeddings of two augmented views of a graph. It provides a simple yet powerful symmetrical neural network architecture and does not require any negative samples in the training process. Experimental evaluation shows that the node representations computed by the proposed framework achieved an analogous performance compared to state-of-the-art methods while requiring substantially fewer hyperparameters and converging in order of magnitude training steps earlier, leading to an overall speedup of up to 42 times compared to state-of-the-art methods.

In the second article (*AttrE2vec: Unsupervised attributed edge representation learning*), a novel method for learning edge representation vectors was introduced. It explores edge neighborhoods via random walks and applies an aggregation function to obtain neighborhood summaries, which are passed along with edge attributes to a deep neural network encoder module. The model is optimized using a compound loss function consisting of a contrastive learning one (to capture the graph structure) and a feature reconstruction loss (to ensure the edge attributes are encoded in the representation vector). Experiments showed that the proposed method build more powerful edge vector representations that achieved better performance in edge classification and edge clustering tasks than other state-of-the-art approaches.

The third publication (*FILDNE: A Framework for Incremental Learning of Dynamic Networks Embeddings*) proposes a novel framework for learning node representation vectors in dynamic graphs using an incremental learning approach. The model utilizes any provided static node representation learning method and applies it to the dynamic graph (modeled as a sequence of graph snapshots). Next, the framework aggregates the embedding vectors from consecutive snapshots using a linear convex combination function, whose parameters are estimated using a Dirichlet-Multinomial model based on an unsupervised link prediction task. The experimental evaluation showed that the proposed framework reduces memory and computational costs while providing competitive quality measure gains with respect to contemporary methods.

The fourth (*Retrofitting Structural Graph Embeddings with Node Attribute Information*) and fifth article (*A deeper look at Graph Embedding RetroFitting*) tackled the problem of updating (retrofitting) precomputed structural node representation vectors with node attribute information. First, a method based on a threefold objective function, consisting of an invariance loss (to preserve the structural embedding information), a graph neighbor loss (to increase the similarity of vectors of nodes connected by edges), and an attribute neighbor loss (to increase the similarity of vectors of nodes with similar attributes), was introduced. The experimental evaluation showed that the proposed method achieved better results in a node classification task than other attributed node representation learning methods. However, the method's hyperparameters required a manual adjustment, and the objective function contained redundant terms. Hence, the fifth article proposed an extension of the introduced method – it simplified the objective function and proposed an algorithm for automatic hyperparameter estimation. An extended experimental evaluation showed that the new method computed better representation vectors than existing approaches for attributed graphs.

To sum up, the results of studies conducted within this doctoral dissertation showed that there exist unsupervised representation learning methods for selected graph entities (nodes, edges) that compute better representation vectors than state-of-the-art unsupervised methods measured by means of downstream task evaluation.

Streszczenie

Metody uczenia maszynowego (ML) były badane w różnych zastosowaniach dla różnych rodzajów danych. Ponieważ większość modeli uczenia maszynowego oczekuje na wejściu wektora z ciągłej przestrzeni, zaproponowano metody uczenia reprezentacji, które automatycznie tworzą wektory reprezentacji (osadzenia) dla danych wejściowych. Podczas gdy istnieje wiele metod osadzania dla tradycyjnych typów danych, takich jak BERT dla tekstu czy ResNet dla obrazów, zadanie to jest znacznie trudniejsze dla danych o strukturze grafu. Grafy mogą być używane do opisania obiektów (za pomocą węzłów) i ich relacji (za pomocą krawędzi). Dla prostych grafów głównym celem metod uczenia reprezentacji jest uchwycenie struktury grafu. Grafy jednakże mogą zawierać dodatkowe informacje, tzn. oprócz struktury (węzłów i krawędzi), można przypisać atrybuty do węzłów, krawędzi, a nawet całych grafów (grafy atrybutowane). Budowa odpowiednich wektorów reprezentacji dla grafów jest kluczowa dla osiągnięcia dobrej jakości w zadaniach uczenia maszynowego, np. klasyfikacji węzłów czy predykcja powiązań.

W ostatnich latach zaproponowano różne metody uczenia reprezentacji grafów. Pomimo sukcesu tych metod, nadal nie rozwiązano wielu problemów i nie zapełniono luk badawczych. Przede wszystkim większość proponowanych podejść jest transduktywna (optymalizują macierz osadzeń o stałym rozmiarze), co nie pozwala na uzyskanie osadzeń dla wcześniej niewidzianych przykładów. Co więcej, dla rzeczywistych, dużych grafów z milionami, a nawet miliardami węzłów, to rozwiązanie wymaga ogromnej ilości pamięci i jest niepraktyczne, ponieważ grafy mogą ewoluować w czasie. Kolejnym poważnym wyzwaniem jest to, że ustawienie semi-nadzorowane wymaga danych oznaczonych, których pozyskanie jest drogim i czasochłonnym zadaniem. Większość istniejących podejść uczenia reprezentacji grafów zostało wprowadzonych w tym właśnie ustawieniu. Model jest optymalizowany pod konkretne zadanie docelowe, co ogranicza ekspresywność reprezentacji. Wykorzystanie alternatywnego podejścia uczenia nienadzorowanego i samo-nadzorowanego pozwoliło uzyskać reprezentacje grafów wyłącznie ze struktury sieci i atrybutów. Kolejnym wyzwaniem jest definicja negatywnych próbek w kontrastowych funkcjach kosztu używanych w metodach uczenia reprezentacji grafów. Jakość osadzeń,

a co za tym idzie, wydajność w zadaniach docelowych, w dużej mierze zależy od tego, jak zdefiniowane są próbki negatywne. W porównaniu do innych typów danych wybór odpowiednich próbek negatywnych jest szczególnie trudny w grafach. Podjęto dlatego trud badawczy skupiony na metodach nie wykorzystujących próbek negatywnych (*negative-sample-free*), np. w przetwarzaniu obrazów i przetwarzaniu języka naturalnego. Jednakże grafy nie były przedmiotem takiego rozpoznania.

Rozprawa doktorska zakładała zrealizowanie następujących celów: (1) sprawdzenie, czy wykorzystanie miary korelacji krzyżowej zastosowanej do zmodyfikowanych widoków grafu atrybutowanego pozwala na trenowanie grafowej sieci neuronowej w sposób samo-nadzorowany, tak aby sieć dostarczała lepsze wektory reprezentacji węzłów niż inne istniejące podejścia samo-nadzorowane (pod względem jakości w zadaniach docelowych i złożoności czasowej procesu uczenia), 2) opracowanie modelu głębokiej sieci neuronowej do wyznaczania wektorów reprezentacji krawędzi, który jest trenowany przy użyciu połączenia kontrastowej funkcji kosztu z funkcją rekonstrukcji cech, tak aby wektory osadzenia tej metody były lepsze w zadaniach docelowych niż te uzyskane jako agregacje reprezentacji węzła źródłowego i docelowego, 3) opracowanie przyrostowej metody uczenia reprezentacji dla węzłów w grafach dynamicznych, która wykorzystuje dowolne statyczne metody osadzania węzłów z kolejnych migawek grafu i wykazuje niższą złożoność czasową i pamięciową niż istniejące metody uczenia reprezentacji na grafach dynamicznych, zapewniając jednocześnie przyrosty miary jakości, oraz 4) opracowanie metody łączenia strukturalnych osadzeń węzłów z informacjami o ich atrybutach w celu otrzymania pojedynczego niskowymiarowego osadzenia, które działa lepiej w zadaniach docelowych niż: strukturalne osadzenia, atrybuty węzłów czy inne metody uczenia reprezentacji węzłów.

Rozprawa doktorska jest zbiorem powiązanych tematycznie prac w postaci pięciu publikacji naukowych skupionych wokół tematu nienadzorowanych metod uczenia reprezentacji grafów.

Pierwsza publikacja (*Graph Barlow Twins: A self-supervised representation learning framework for graphs*) proponuje nowy framework do samo-nadzorowanego uczenia reprezentacji węzłów w atrybutowanych grafach, który wykorzystuje empiryczną macierz korelacji krzyżowej pomiędzy osadzeniami dwóch zmodyfikowanych widoków grafu. Zapewnia to prostą i ekspresywną symetryczną architekturę sieci neuronowej, która nie wymaga negatywnych próbek w procesie uczenia. Eksperymentalna ewaluacja uwidoczniła, że otrzymywane reprezentacje węzłów osiągały analogiczną jakość w porównaniu do najnowszych metod, wymagając znacznie mniejszej liczby hiperparametrów i zbiegając szybciej, co prowadzi do ogólnego przyspieszenia do 42 razy w porównaniu do najnowszych metod.

W drugim artykule (*AttrE2vec: Unsupervised attributed edge representation learning*) wprowadzono nową metodę uczenia wektorów reprezentacji krawędzi. Metoda ta bada sąsiedztwo krawędzi za pomocą spaceru losowego i stosuje funkcję agregacji do uzyskania podsumowań sąsiedztwa, które następnie są przekazywane razem z atrybutami krawędzi do modułu kodera głębokiej sieci neuronowej. Model jest optymalizowany za pomocą złożonej funkcji straty składającej się z kontrastowego uczenia (aby uchwycić strukturę grafu) i funkcji rekonstrukcji cech (aby zapewnić, że atrybuty krawędzi są zakodowane prawidłowo w wektorze reprezentacji). Eksperymenty wykazały, że proponowana metoda tworzy reprezentacje, które osiągnęły lepszą jakość w zadaniach klasyfikacji i grupowania krawędzi w porównaniu do innych podejść najnowszej generacji.

Trzecia publikacja (*FILDNE: A Framework for Incremental Learning of Dynamic Networks Embeddings*) proponuje nowy framework do uczenia wektorów reprezentacji węzłów w dynamicznych grafach za pomocą uczenia przyrostowego. Model wykorzystuje dowolną podaną statyczną metodę uczenia reprezentacji węzłów i stosuje ją do dynamicznego grafu (modelowanego jako sekwencja migawek grafu). Następnie framework agreguje wektory osadzeń z kolejnych migawek za pomocą funkcji liniowo-wypukłej, której parametry są wyznaczane za pomocą modelu Dirichlet-Multinomial przy użyciu nienadzorowanego zadania predykcji połączeń. Eksperymentalna ewaluacja wykazała, że proponowany framework jest bardziej wydajny pod względem złożoności czasowej i pamięciowej, a jednocześnie osiąga lepszą jakość w porównaniu do współczesnych metod uczenia reprezentacji na dynamicznych grafach.

Czwarty artykuł (*Retrofitting Structural Graph Embeddings with Node Attribute Information*) oraz piąty (*A deeper look at Graph Embedding RetroFitting*) zajęły się problemem aktualizacji (retrofitowania) wstępnie obliczonych strukturalnych wektorów reprezentacji węzłów za pomocą informacji o ich atrybutach. Najpierw wprowadzono metodę opartą na złożonej funkcji celu, składającą się ze straty niezmienności (aby zachować informacje o osadzeniu strukturalnym), straty sąsiedztwa grafu (aby zwiększyć podobieństwo wektorów węzłów połączonych krawędziami) i straty sąsiedztwa atrybutów (aby zwiększyć podobieństwo wektorów węzłów o podobnych atrybutach). Ewaluacja eksperymentalna wykazała, że proponowana metoda osiągnęła lepsze wyniki w zadaniu klasyfikacji węzłów niż inne metody uczenia reprezentacji węzłów z atrybutami. Jednakże zaproponowana metoda posiadała wiele hiperparametrów, a funkcja celu była nadmiarowa. W związku z tym w piątym artykule zaproponowano uproszenie metody w zakresie funkcji celu i zaproponowano algorytm do automatycznego wyznaczania hiperparametrów. Rozszerzona ewaluacja eksperymentalna wykazała, że nowa metoda obliczała lepsze wektory reprezentacji niż istniejące podejścia do grafów z atrybutami.

Podsumowując, wyniki badań przeprowadzonych w ramach rozprawy doktorskiej wykazały, że zaproponowane metody i podejścia, jako nienadzorowane metody uczenia reprezentacji, dostarczają bardziej generalizujące wektory reprezentacji podczas ewaluacji na zadaniach docelowych niż dotychczas istniejące metody nienadzorowane.

Contents

Chapter 1

Introduction

In recent years, the amount of data produced by humans has been exponentially growing. Handling such huge volumes, i.e., processing the data and extracting valuable knowledge, is far beyond human reach. To effectively solve this problem, scientists built a wide variety of so-called machine learning models that are able to **learn** from the provided data and detect useful patterns. The complexity of the phenomena that is reflected in the produced data makes the data unstructured or highly complex in its structure. However, mathematical and statistical models require data in a given, specific format, i.e., virtually all machine learning models operate on real-valued number vectors. Such vector **representations** should reflect certain characteristics of the corresponding data.

The majority of all developed classical machine learning models operate under the assumption that the data is independent and identically distributed (iid). However, data points are rarely independent of each other; in fact, many data types exhibit a **relational nature**, i.e., there exist certain relations (dependencies) between samples. For instance, social networks have multiple types of relations between people, such as family relations, business ones, and friendships. In computational chemistry, when analyzing molecules, there are relations between certain atoms due to chemical bonds. Another example are citation networks, where scientific articles are not only connected to their authors but also to venues and research areas. One could try to process each data point independently, but harvesting the information from the relations could solve the target problems more effectively.

Such relational types of data, as given in the examples above, are built of certain **entities** connected to each other by **relations**. A mathematical way to formally describe them is a **graph**. In particular, a graph $\mathcal{G}$ is defined as a 2-tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes (entities) and $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ is the set of edges connecting

pairs of nodes. The connections in a graph are often also presented as a binary adjacency matrix $\mathcal{A} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, where ones are present if and only if there is an edge: $\mathcal{A}_{ij} = 1 \iff (i,j) \in \mathcal{E}$, and zeros denote no relations between given nodes.

In order to provide better expressiveness of graphs, especially for real-world applications, they may be equipped with additional features (attributes). The graph becomes an **attributed graph** when some features are associated with any of its entities, e.g., for attributes associated with each node graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V}$ are nodes, $\mathcal{E}$ are edges, and nodes' attributes are described by an attribute matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|\times m}$. Depending on the actual data, features might also be attached to edges, sub-graphs, or whole graphs.

Originated from the combination of graph theory and deep learning intelligent processing of such complex graphs, also known as **graph machine learning** (GraphML), involves prediction over nodes, edges and graphs itself. For instance, in **node classification**, the goal is to predict the most likely node's label. In a social network, this might be an interest or preference of the user, or in a citation network, the research area the paper belongs to [70]. In **link prediction**, the existence of a link between a pair of nodes is modeled [61, 68, 45]. Predicted links in social networks may denote real-life friends and in citation networks related but unmentioned references. In computational chemistry [48, 100], prediction over whole graphs (**graph classification**) may be carried out to decide whether a certain molecule will cause a specific chemical reaction, or if we consider larger structures, such as proteins, will they cure a given disease.

As previously stated, machine learning models necessitate the use of numerical vectors (features). There are two primary methods to obtain these. The traditional technique involves creating unique, independent features manually, utilizing specialized knowledge specific to the domain in question. Such a historical approach is called **manual feature engineering**. In the case of graphs, the vector representations in node classification or link prediction were obtained from some graph transformation and just adopted some of the structural properties, e.g., node degrees, Laplacian matrices, or neighborhood overlapping. However, this manual approach does not generalize well across different prediction tasks and usually strongly depends on the domain. This thesis is focused on the alternative approach – **representation learning** [10], where the features (representations) are automatically extracted by specialized machine learning models that solve some optimization problem.

Let's now introduce some crucial concepts to understand better the thesis's research area, i.e., **unsupervised representation learning for graphs**.

Representation learning. The main objective in representation learning [10] is to learn the parameters θ of a so-called **embedding** function $f_\theta : \mathcal{X} \to \mathbb{R}^d$ that maps given set of objects $\mathcal{X}$ to low-dimensional real-valued vectors (i.e., $d \ll \dim(\mathcal{X})$), such that these vectors encode certain characteristics of the corresponding objects. If two objects are similar, their **representation vectors** should also be similar, e.g., if two images contain cats, the vectors should be more similar than vectors of cat and dog images.

Learning principles. In machine learning, there are two basic approaches for training a model: supervised and unsupervised. The difference lies in the access to labeled data used in model parameter estimation. For the supervised setting, the model is trained using annotated data points, i.e., each sample is labeled. Note that humans execute this annotation process, which is time-consuming, error-prone, and subject to personal bias. In the **unsupervised** approach, there are no labels available, so the model only has access to the data points. In the case of representation learning, vectors obtained through the supervised paradigm are often specialized to the task they were trained on, hence the generalization ability to other tasks is limited. As representations trained in an unsupervised manner are focused on the data structure itself, they might exhibit a better generalization ability to various **downstream (application) tasks**.

Unsupervised vs. self-supervised learning. In recent years, the **self-supervised learning (SSL)** [27] term has gained attraction in the research community. While unsupervised methods often utilize data reconstruction approaches, in the SSL setting, the methods automatically create so-called soft labels during training and use them to train a particular backbone model (in the case of representation learning, the backbone model is the embedding function/feature extractor). Despite this difference, note that as there is no external annotation process involved, so self-supervised learning still resides within the area of unsupervised learning (i.e., SSL methods are a subset of unsupervised methods).

Transductive and inductive approaches. An important aspect of representation learning is the ability of the embedding model to provide representation vectors for unseen data points. In the case where a model learns a fixed lookup table (or matrix), with each row essentially being the embedding vector of an individual data point, it is not feasible to generate a vector for a new, unseen instance without completely retraining the model using a dataset that includes this new point. Such a model is considered to be **transductive**. On the other hand, if the model learns a mapping function that transforms some initial attributes into the embedding vector, it is possible

to generalize to new data points – i.e., pass their attributes through the embedding function to obtain its vector representations. Such a scenario is an example of an **inductive** model. Inductive models are often more feasible in large-scale applications, where keeping track of a huge lookup matrix might not be possible – inductive models only store the function parameters in memory. Naturally, this entails a trade-off with respect to the inference time – inductive setting requires a forward pass to obtain the representation vector of a given data point.

1.1 Graph representation learning

The term **graph representation learning (graph embedding)** [8, 50] is quite general and defines a whole hierarchy of representation learning methods. The aim is to capture certain graph entities' characteristics and encode them as real-valued vectors. More formally, the embedding function $f_\theta : \mathcal{G} \rightarrow \mathbb{R}^d$ transforms the high dimensional graph data $\mathcal{G}$ into a low dimensional vector representation $\mathbb{R}^d$, where d is the dimensionality of the vector representation and θ parameterizes the embedding function. Note that we expect the embedding dimension *to be much smaller than* the original data dimensionality: $d \ll \dim(\mathcal{G})$.

There are different 4 entities in graphs that could be the target for representation learning methods (see: Figure 1.1):

- **nodes** – Most of the published algorithms are designed for nodes. The embedding function f_θ will return an embedding matrix with one vector for each node, i.e., $\mathbb{R}^{|\mathcal{V}| \times d}$, where $|\mathcal{V}|$ is the number of nodes. Various groups of node embedding methods will be briefly discussed in Section 1.3.

- **edges** – Although link (edge) prediction is one of the most popular applications for graph embedding, the number of edge-oriented embedding methods is relatively low. Special transformations are mostly used to obtain edge representations from node embeddings. The embedding function f_θ is defined similarly as for nodes, but the output matrix contains one vector for each edge, i.e., $\mathbb{R}^{|\mathcal{E}| \times d}$, where $|\mathcal{E}|$ denotes the number of edges.

- **subgraphs** – This group of embedding methods is often used when modeling social networks. A quite popular application is detecting and representing communities in networks, e.g., groups of students at a university or football fan clubs. The output matrix will contain one vector for each community/subgraph, i.e., $\mathbb{R}^{|C| \times d}$, where $|C|$ is the number of communities/subgraphs.

- **whole graphs** – Embedding whole graphs has many applications in chemistry, biology, and medicine. The objective is to find a representation of chemical structures, like molecules or other more advanced structures (proteins). The embedding function is defined as mapping the whole graph to one vector.

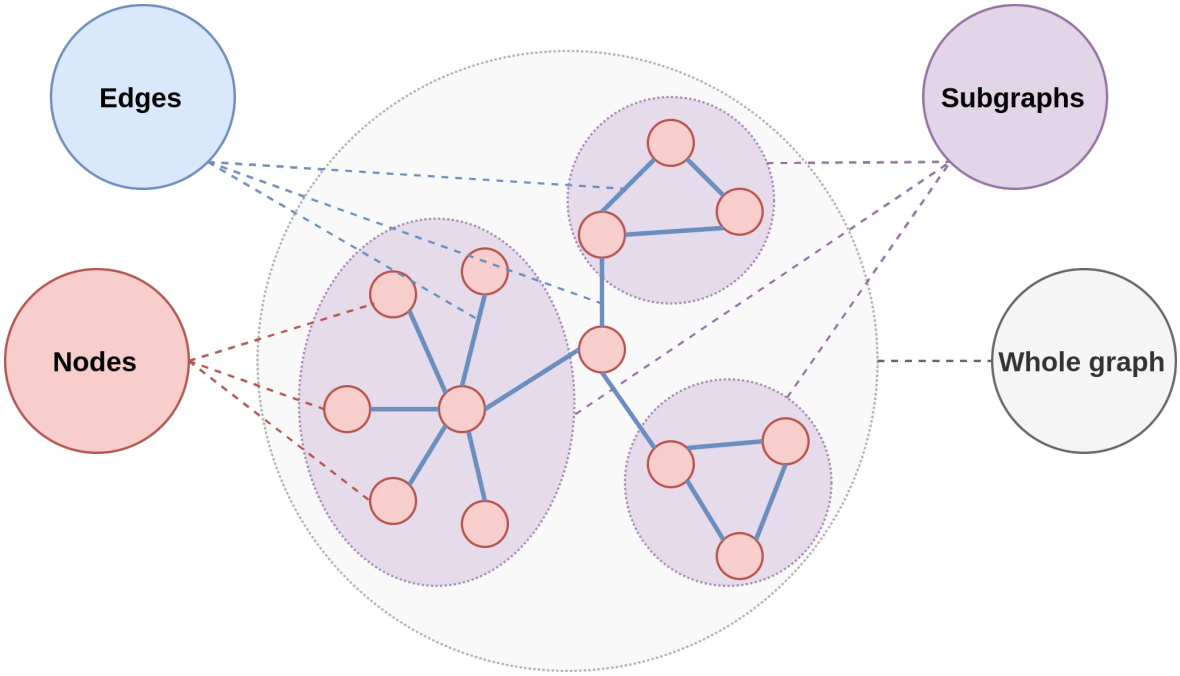


Figure 1.1: Representation learning entities in graphs

1.2 Graph representation learning applications and tasks

Although quite broad, the application areas can be classified into a finite set of groups. Let's briefly introduce the most essential downstream tasks and their respective application areas. Figure 1.2 shows the downstream tasks with respect to the graph entity they concern.

Link prediction Edges (links) are the main building block of graphs. They provide information about the existence and/or type of relation between two entities (nodes). The observed edge set might often be incomplete or inaccurate. The link prediction task focuses on finding missing edges (for static graphs) or predicting which edges will be observed in the future (dynamic, evolving graphs). It is one of the most popular graph representation learning downstream tasks. Real-world applications can be found in biology, where link prediction is used as a cost-effective alternative to traditional

experiment-based verification of link existence in social networks, where it can be used for friend and content recommendation, providing a better and personalized user experience. Link prediction surveys [61, 68, 45] propose the following methods taxonomy: similarity-based (with local and global network structure perspective) [51, 1], maximum likelihood [103, 22] and probabilistic models [34, 111].

Node classification Nodes might have assigned labels and/or feature vectors, e.g., in social networks, these reflect peoples' interests, social status, personal data, etc.; in biological networks, node labels indicate their role in the whole graph. Often only a fraction of all nodes is labeled (the rest is assumed to be unknown). Node classification is a task that aims to infer the missing labels and is one of the two most popular downstream tasks. Various methods can be found in Bhagat et al. [12] survey. Based on their taxonomy, there are (1) feature extraction and (2) random walk-based methods. In the first case [74, 66, 13], node features are aggregations of the neighbor nodes' features, refined with some local network statistics. The new feature vectors are fed into a classifier (popular ones include Logistic Regression and Naive Bayes), which predicts the node labels. In the case of random walk-based approaches [6, 7], the labels are propagated through the network using paths obtained by random walks.

Edge classification/regression This task is similar to node classification, but the goal is to predict edge labels or continuous values in the case of edge regression. In contrast to node classification, this application area is rather unexploited. Aggrawal et al. [2] presents a survey of existing methods and real-world application use cases. These include social networks with different kinds of relations/edges [5, 14] (e.g., friendship, family relations) between people. Edge classification will be used to infer the types of non-labeled edges based on already labeled ones. Further analysis of these relations could also be used for making better, personalized recommendations [43, 58]. When edge labels describe the strength of a relationship [24, 109] (e.g., number of likes, ratings), edge regression can be applied to predict missing values or how they will change after the network evolves.

Graph reconstruction When obtaining embedding vectors, the algorithms should preserve distances from the original graph, i.e., if two nodes were close to each other in the graph, the distance of their representation vectors should be small. To measure the quality of an embedding (algorithm), one could define two factors: a local one, which checks if neighbor nodes remain close to each other in the embedding space without considering the actual distances, and a global one, which keeps track of these distances. Two measures are often used: MAP (Mean Average Precision) to model the local factor

and Distortion as the global one. Better embeddings allow easier training of so-called graph decoders, which aim at reconstructing the graph's adjacency/distance matrix from the embeddings. These decoders can be used in network compression scenarios (see below) or as generative models for creating new graphs (or predicting the state of an evolving network). Graph reconstruction can also be considered only in terms of measuring the quality of embedding (as defined above). Although it was not quite popular in the past, recent development in hyperbolic graph embedding [56, 17, 82] has focused on graph (tree) reconstruction.

Clustering Finding clusters in graphs can be based on their structure, node/edge attributes, or hybrid methods. In the case of the first criterion (structure) [25, 84], one could ask to find dense substructures with many edges connecting the nodes within a given cluster and fewer edges connecting to other clusters (so-called *community-based*). Another group of structure-aware methods is *structurally equivalent clustering* [107], where the goal is to find nodes with similar roles in the graph, e.g., bridges, hubs, outliers. The second major group of clustering methods is based on node/edge attributes [114] and aims to find dense subgraphs based not only on observed edges but also on similar node features (labels).

Network compression Network compression, also known as graph simplification, was introduced by Feder et al. [28]. Based on a given input graph, the goal is to obtain a graph with a smaller number of edges, allowing it to run other algorithms faster and efficiently store the graph itself. Initially, aggregation methods [91, 92] (not embedding-based) were proposed, which used the edge structure to group nodes. Other methods were based on information theory, e.g., Navlakha et al. [73] using Minimum Description Length [80]. The first papers that used an embedding in this application area [97, 77] encoded the graph into embeddings, and then based on these, they reconstructed the graph, measuring the reconstruction error. On the whole, network compression is rather a less popular application area.

Visualization Visualization techniques are often used in scientific articles to emphasize the predictive power of the proposed methods by showing 2-dimensional projections of the representation vector spaces. Depending on the actual task, different kinds of plots are being used, e.g., in node classification, scatter plots with different point colors are a perfect fit. Authors claim that the better different classes are separated, the better the embedding algorithm. Of course, this might not always be true, as the projections may not reflect the separation in the original spaces. In most cases, either Principal

Component Analysis (PCA) or t-distributed stochastic neighbor embedding (t-SNE) [93] is applied. Other applications for graph visualizations can be found in electrical circuits analysis [26], biology [90], sociology [30], and software engineering [31].

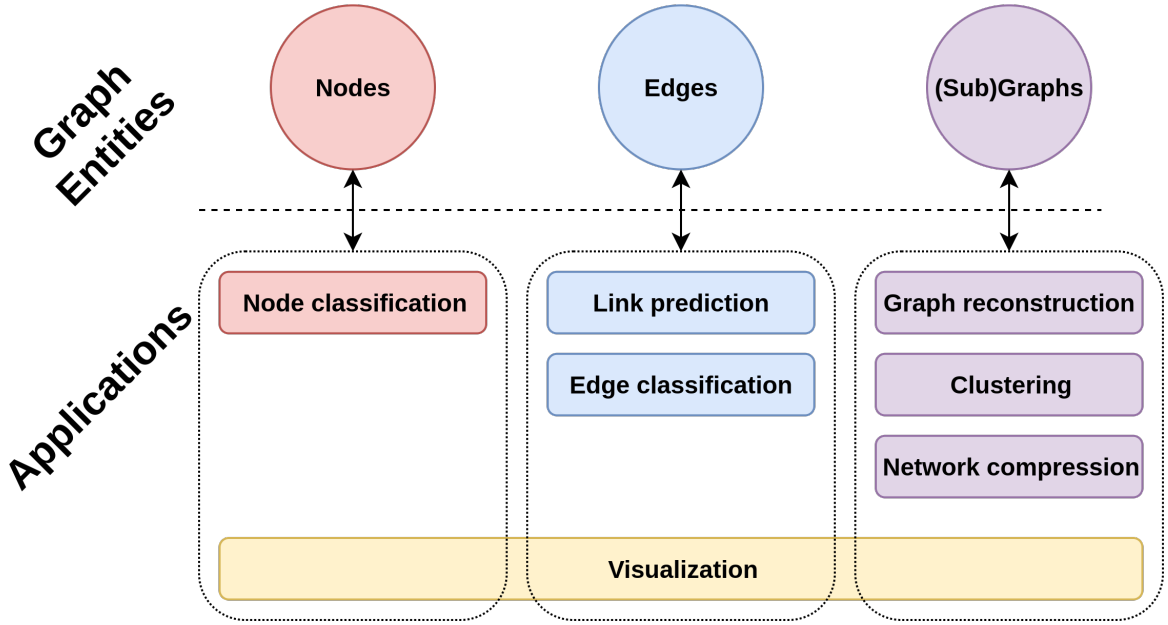


Figure 1.2: Graph representation learning applications and graph entities they concern.

1.3 Methods for learning node representations

Methods designed for node representation learning can be based on: **matrix factorization**, **random walks**, **deep learning** and **graph neural networks**. Let's now provide some details about each method group.

1.3.1 Matrix factorization based methods

Factorization algorithms can be used to decompose a given input matrix into a group of smaller matrices whose product should approximate the input matrix. For instance, if the input matrix is positive semidefinite (e.g., Laplacian matrices), algorithms like eigenvalue decomposition or SVD (Singular Value Decomposition) [65] are utilized. Otherwise, one can use gradient descent methods. The input matrix is essential as it determines what the final representation vector will encode (e.g., local or global neighborhoods; first, second or higher-order proximities, etc.). Popular choices for graphs are the adjacency matrix (**LLE** [81]), Laplacian matrices (**Laplacian Eigenmaps** [9], **Cauchy graph embedding** [67], **Structure Preserving Embedding** [83]) or node transition probability matrices (**GraRep** [15]). Methods, such as **M-NMF** [99], might

also preserve both microscopic graph structures (1st and 2nd order node proximities) and mesoscopic properties (i.e., communities). In general, factorization-based approaches are computationally expensive, however, some methods address this limitation, e.g., **Graph Factorization** [4] (considers only observed edges), **HOPE** [77] (uses sparse representations). Moreover, some methods extend existing approaches and reformulate them into a matrix factorization setting, e.g., **TADW** (Text-attributed DeepWalk) [108] enables the DeepWalk [78] to incorporate node text features; Yang et al. [47] converts the results of dimensionality reductions algorithms, like Principal Component Analysis [49], ISOMAP [88], Multi-Dimensional Scaling [57] into non-negative embeddings.

1.3.2 Random walk-based methods

Random walks are a good choice for dealing with extremely large graphs, where factorization-based methods are not suitable (due to time or memory complexity). They can approximate many graph measures, like node centralities [75] or similarities [29]. A random walk generates a sequence of nodes by iteratively choosing a single node neighbor and moving to that node afterwards, until some stopping condition applies, e.g., walk length or no other choice available. Both the neighbor selection algorithm and the stopping condition are method-specific. **DeepWalk** [78] builds representations that preserve higher-order node proximities by adopting the Skipgram [72] model (known from natural language processing) into the graph domain. The node sequences generated by the random walks are treated as sentences and are fed into the Skipgram algorithm. Neighbor selection in the random walk uses a uniform distribution (unbiased random walk). **Node2Vec** [41] extends the neighbor selection procedure so the random walks can model both mesoscopic structures, such as communities (breadth-first search), as well as preserve local structures (depth-first search). The random initialization of the embedding matrix and the non-convex objective function may cause these two algorithms to be stuck in local minima. **HARP** [18] uses graph coarsening [20] to hierarchically compute embeddings using DeepWalk or Node2vec and use the computed vectors as initialization for the next layer in the hierarchy. **Walklets** [79] preserve higher-order proximities by combining a factorization-based approach with random walks.

1.3.3 Deep learning-based methods

The successful applications of deep learning in various machine learning tasks, like image recognition, segmentation, or text processing, motivated its usage in graph embedding. Due to their architecture and known properties, deep neural networks are able to

capture non-linearities in the graph structures. One of the most popular neural network architectures – the autoencoder – was the foundation for models like **SDNE** (Structural deep network embedding) [97], which preserves first and second-order proximities by jointly optimizing these measures in both unsupervised (neighborhood reconstruction) and supervised (Laplacian Eigenmaps) manner. **DNGR** (Deep neural networks for learning graph representations) [16] uses random surfing and the positive pointwise mutual information (PPMI) measure along with stacked denoising autoencoders to build representation vectors. **VAE** (Variational Auto-Encoders) [52] were adopted into the graph domain as the **VGAE** (Variational Graph Auto-Encoder) [54] model. It combines the VAE network with a **Graph Convolutional Network** [55] and an inner product decoder. There are also generative models, such as **GraphGAN** [98], **Adversarial Network Embedding** [23] and **ProGAN** [32], which utilize the Generative Adversarial Networks architecture [37]. Other popular deep learning architectures, like multilayer perceptrons (**PALE** [69]) and recurrent neural networks (**DeepCas** [60]) are also used by graph representation learning models. Recent advancements focus on utilizing Normalizing Flows (**Graph Normalizing Flows** [63], **Equivariant Normalizing Flows** [33]) and **Diffusion Models** [62] architectures.

1.3.4 Graph neural networks

Although these are deep learning models, the recent exponential development of new methods should be discussed separately. **Graph neural networks (GNN)** [113] were designed to directly operate on graphs. They take as input the adjacency matrix of the graph and an attribute matrix (depending on the actual method, these might be node-, edge-, or graph-level attributes). One of the most popular architectures is the so-called **MPGNN** (Message Passing Graph Neural Network), where nodes have the ability to "pass" messages to their neighbors. Aggregating those messages in each layer, combined with appropriate transformation and activation functions, allows for efficient processing of even large graphs (assuming usage of sparse matrix implementations) and obtaining robust embeddings. In particular, a single layer in a message passing GNN performs the following three operations – message generation, feature aggregation, and feature update:

$$\mathbf{m}_v^{(k-1)} = \text{MESSAGE}(\mathbf{h}_v^{(k-1)}) \tag{1.1}$$

$$\mathbf{h}_{\mathcal{N}(u)}^{(k-1)} = \text{AGGREGATE}(\{\mathbf{m}_v^{(k-1)} : v \in \mathcal{N}(u)\}) \tag{1.2}$$

$$\mathbf{h}_u^{(k)} = \text{UPDATE}(\mathbf{h}_u^{(k-1)}, \mathbf{h}_{\mathcal{N}(u)}^{(k-1)}) \tag{1.3}$$

where $\mathbf{m}_v^{(k-1)}$ is the message generated by node v based on the input features $\mathbf{h}_v^{(k-1)}$, $\mathbf{h}_{\mathcal{N}(u)}^{(k-1)}$ denote the aggregated features of u's neighbors and $\mathbf{h}_u^{(k)}$ denote the output features of node u. In a GNN model with multiple layers, the first one operates on the attribute matrix, i.e., $\mathbf{h}^{(0)} = \mathbf{X}$. A single layer models a 1-hop neighborhood, and by adding additional layers to the model, one could also model k-hop neighborhoods (one hop for every layer).

Graph convolutional networks (GCN) [55] implement the idea of message passing by defining a convolution operator for graphs. For a given node, it is implemented as a weighted summation of neighbors' representations using a symmetrical degree-normalized adjacency matrix. **Graph Attention Networks (GAT)** [96] also computes a weighted sum over neighbor embeddings, but it uses the attention mechanism [94] to compute the weight coefficients. **Graph Isomorphism Networks (GIN)** [106] use an analogy to the Weisfeiler-Lehman isomorphism test to obtain the most expressive GNN in the class of message-passing models. **GraphSAGE** [44] addresses the problem of inductive learning of large-scale graphs by sampling only a part of the neighborhood. There are numerous other GNN architectures [105, 50, 8], but the four above are the most adopted.

Graph neural networks are often introduced either as standalone deep learning model layers or are trained in an end-2-end manner using the supervised setting on a particular classification task. Hence, the representation vectors are task-specific. Although the Graph Autoencoder can be paired with any existing GNN layer, the performed link prediction task does not provide the best overall generalization of the embedding vectors.

Recently, the self-supervised paradigm allowed to obtain state-of-the-art performance for graph neural networks without using labels [64]. Sometimes the performance was even better than for the supervised case. Instead of relying on autoencoders, contrastive methods produce multiple graph views, encode them and minimize embedding distances of matching nodes as well as maximize (contrast) distances to other nodes (negative samples), e.g., **GraphCL** [110], **GCA** [116], **GRACE** [115], or **DGI** [95]. Defining negative samples is a hard task, but some methods eliminate the need for negative samples, e.g., **BGRL** [89]. However, this particular method does require many epochs to converge, and it relies on many training tricks to overcome a trivial embedding collapse.

1.4 Identified research gaps

Although graph representation learning research is already quite developed, some problems and aspects have not been exploited sufficiently. Let's now briefly describe research gaps and problems present in unsupervised representation learning for graphs that were identified during the development of this Ph.D. thesis.

- **Self-supervised representation learning methods are computationally expensive due to the utilization of negative samples**

  As discussed earlier, the self-supervised learning paradigm gathers current interest in the GraphML research community, with methods prominently developed around the contrastive learning approach (GCA [116], GraphCL [110], GRACE [115] or DGI [95]). Despite the success of contrastive methods in areas such as computer vision and natural language processing, their fundamental limitation is the need for **negative samples**. Consequently, their sampling procedure highly affects the overall quality of the representations. Regarding images or texts, the definition of negative samples might not seem that problematic, but in the case of graphs, there is no clear intuition. For instance, what is the negative counterpart for a particular node in the graph – should it be a node that is not a direct neighbor or a node that is in a different graph component? Multiple options are available, but the right choice strictly depends on the downstream task.

  A solution to this problem might be the application of so-called **negative-sample-free** methods. In computer vision, they obtained successful results with methods such as BYOL [40], SimSiam [19], or Barlow Twins [112]. Using siamese network architectures with various techniques, like gradient stopping, asymmetry, or batch and layer normalizations, they prevent collapsing to trivial solutions. Based on BYOL, the Bootstrapped Representation Learning on Graphs (BGRL) [89] framework was proposed. Its computed representation vectors achieved state-of-the-art performance in node classification using various benchmark datasets. Notwithstanding, assuming asymmetry between the network twins (such as the predictor network, gradient stopping, and a moving average on the weight updates), the method is conceptually complex. Moreover, as the paper shows, this method requires many epochs to converge to state-of-the-art performance.

  The question arises: how to build a self-supervised graph representation learning model that does not require defining negative samples (i.e., negative-sample-free method) and provides time efficiency without using any additional tricks in the architecture and training procedure? The thesis addressed this question by means of the **Research objective 1**.

- **Edge representation learning is underrepresented**

  Most graph representation learning methods are designed for learning embeddings of nodes. However, one could be interested in prediction tasks involving pairs of nodes instead of individual nodes, e.g., link prediction and edge classification/regression.

  The authors of Node2vec [41] defined a set of 4 independent **binary operators** (see Table 1.1), which for given two nodes u and v transform the corresponding feature vectors $f(u)$ and $f(v)$ into edge features $g(u, v)$, so that: $g : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^d$, where d is the node/edge embedding dimension (these operators preserve the size of the input embeddings). Such an approach is independent of the initial node embedding algorithm, so the user can employ the most appropriate one in a given case and use one of the operators to evaluate the produced embeddings against edge-specific tasks.

Operator	Symbol	Definition
Average	$\oplus$	$[f(u) \oplus f(v)]_i = \frac{f_i(u)+f_i(v)}{2}$
Hadamard	$\odot$	$[f(u) \odot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\|\| \cdot \|\|_{\overline{1}}$	$\|\|f(u) \cdot f(v)\|\|_{\overline{1}i} = \|f_i(u) - f_i(v)\|$
Weighted-L2	$\|\| \cdot \|\|_{\overline{2}}$	$\|\|f(u) \cdot f(v)\|\|_{\overline{2}i} = \|f_i(u) - f_i(v)\|^2$

Table 1.1: Proposed binary operators for transforming node feature vectors $f(u), f(v)$ into edge features. The i subscript denotes that these operations are applied element-wise to the feature vectors. *Source: Node2vec paper [41]*

Aggrawal et al. [2] proposed an algorithm for edge classification and regression using a Jaccard-based node similarity measure. In the simplest case (binary classification), there are separate measures for positive and negative edges, but the authors also showed how to extend this idea to numerical labels. Eventually, the predicted label is computed as an aggregation (maximum or average) over all considered edges. In this variant (exact algorithm), the whole procedure could be too time-consuming when dealing with a large-scale graph. The min-hash index algorithm was used to estimate the Jaccard coefficient similarities to obtain a fast probabilistic approximation. Importantly, these methods work in an end-2-end manner, i.e., no edge features are being extracted.

Graphs might have initially defined edge features. In such a case, both approaches are unsuitable, as they ignore these features, possibly missing discriminative power for downstream models. There are many solutions to this problem. One of them is to modify these algorithms to include initial edge features. In the case of Node2vec's binary operators, they should be generalized to ternary operators

with two inputs for node features and one for the edge features. Of course, there might be a problem with the different dimensionalities of node and edge features. Handling such cases is not trivial and may require using additional dimensionality reduction techniques or even the application of neural networks. In the case of Aggrawal's Jaccard-based method, the generalization is also not trivial. Still, one could try to average the edge features instead of the target labels and feed them into a downstream classifier/regression model.

Other solutions include using: (1) node embedding algorithms, which are capable of incorporating edge features and then using those enriched node features to obtain edge representations using the above-defined binary operators, or (2) edge embedding algorithms, which directly learn these features (although, there are only a few solutions available). GCNs could be employed for one-dimensional edge features, which is, in most cases, impractical. GATs (Graph Attention Networks) [96] may use multi-dimensional edge features when computing the attention scores. The authors of **NRIM** [53] use special kinds of message-passing graph neural networks that perform multiple node-to-edge and edge-to-node messages pass rounds. After training, this model could even be used to directly infer edge features by stopping the algorithm after a node-to-edge stage. The **EGNN** [36] paper claims that other approaches do not properly use edge features, and it proposes a modified attention layer, which includes and updates the whole edge feature tensor. The [101]paper considers the case of point clouds, where edge features are initially obtained using simple MLPs. Then these are aggregated using the proposed **EdgeConv** operator (sum or maximum over feature vectors). The authors of **[85]** introduce so-called **edge-conditioned convolutional filters**, which computes a weighted average of node vectors with edge features as weights. A survey of another edge-aware message passing architectures can be found in [35].

In summary, obtaining edge representation vectors often involves computing node embeddings beforehand and applying trainable or non-trainable binary operators. Other approaches that are designed for edges are either trained in a supervised manner or do not take edge attributes into account. The question arises: how to build an edge representation learning method that is trained in an unsupervised approach and utilizes existing attributes? This thesis addressed this question by means of the **Research objective 2**.

- **Learning representations of nodes in dynamic graphs is not fully benefiting from incremental learning**

Many real-world networks are naturally dynamic, meaning they evolve, and both nodes and edges may appear or disappear. Considering the temporal (dynamic) network information allows their better understanding and modeling [76, 102]. A **dynamic graph** is represented by a 2-tuple: $\mathcal{G} = (\mathcal{V}(t), \mathcal{E}(t))$, where the node $\mathcal{V}$ and edge $\mathcal{E}$ sets depend on time t. A dynamic graph is often modeled either as a *event stream* of (*sender*, *receiver*, *timestamp*) triplets (with potentially additional features) or more commonly as a *sequence of discrete graph snapshots*: $\mathcal{G}_{0,1}, \mathcal{G}_{1,2}, \ldots, \mathcal{G}_{T-1,T}$, where T is the number of timesteps and $\mathcal{G}_{t-1,t} = (\mathcal{V}_{t-1,t}, \mathcal{E}_{t-1,t})$ is the t-th graph snapshot with node and edges present between timestep $t-1$ and t.

One must provide an efficient and robust embedding algorithm when learning representations for large-scale, disk-resident dynamic graphs [71, 3]. Due to the nature of such graphs, a full re-computation of the embeddings at every timestep is not feasible. Embeddings should be obtained in an incremental manner. Note that in this approach, there is no need to keep previous graph snapshots during consecutive iterations, as the information is already encoded in the model parameters. Therefore, incremental learning helps to save both time and memory during training. An illustration of training node representation vectors in a dynamic graph using an incremental learning approach is presented in Fig. 1.3.

There exist methods for learning node representations in dynamic networks, most based on deep learning architectures. **DynGEM** [39] employs an autoencoder to reconstruct the adjacency matrix. Weights learned in one timestep are used as initialization in the next timestep. **Dyngraph2vec** [38] extends the DynGEM model and defines a family of deep neural network models (autoencoder, LSTM, and a hybrid approach) that are trained to predict the next snapshot based on multiple previous snapshots.

Some methods adapt existing representation learning approaches to the domain of dynamic graphs. **n2v-dynlink** [104] applies Node2vec on each snapshot and concatenates the resulting vectors to obtain the final representation. Such a method is hardly scalable, as the embedding dimension grows with each new graph snapshot. **tNodeEmbed** [86] also applies Node2vec on each snapshot, but then it uses an LSTM network to align them. The whole model is trained an end-2-end, task-specific manner. The alignment procedure is motivated by the stochastic nature of Node2vec and changes in the graph structure. **OnlineCTDNE** [59] reuses the CTDNE algorithm and computes temporal random walks for each edge. Similarly, **Streamwalk** [11] utilizes temporal random walks but only updates the vectors for a subsample of nodes.

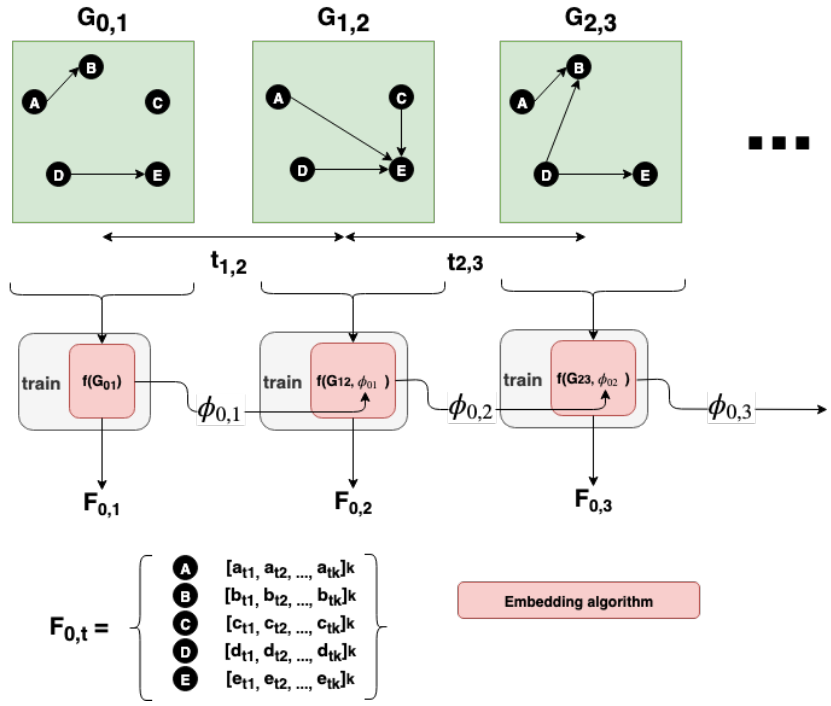


Figure 1.3: The idea of incremental learning for dynamic graph embedding. At each timestep t, the embedding $\mathbf{F}_{0,t}$ represents the whole evolution history of the nodes starting from the zeroth timestep. First, a representation learning model f is trained on the initial graph $\mathcal{G}_{0,1}$. In consecutive iterations, the model updates the embeddings based on the current graph snapshot and the previous model parameters.

Instead of reusing only selected node representation learning methods, it would be beneficial to let the user choose a method that is appropriate for the given use case. The question arises: how to design a node representation learning method for dynamic graphs which is trained in an unsupervised and incremental manner and is able to utilize any given static node embedding method (applied to consecutive graph snapshots)? This thesis addressed this question by means of the **Research objective 3**.

- **Existing precomputed structural embeddings can be fused together with node attributes**

  While numerous graph representation learning methods exist, either designed to reflect the graph structure or attributes in the context of the structure, there has been no work done on fusing (node) attribute information into existing precomputed structural (node) representation vectors. Such a situation might be useful in industrial applications where production systems were built using structural graph representation learning methods and, later on, additional node features were introduced. Full retraining might not be feasible due to the dataset's scale or other systems dependent on the current embedding vectors. Another scenario could be sensor networks, where the underlying graph's structure is static,

but the node attributes (sensor readings) are constantly changing. Recomputing the representations from scratch each time a new reading appears is inefficient. Embedding methods for dynamic graphs are also not applicable, as they often consider node attributes to be static while the structure changes. Moreover, note that the simplest solution of concatenating both structural and attribute vectors is also not feasible as the resulting representation would be neither consistent nor low-dimensional.

Henceforth, a method for computing representations in such a scenario requires using existing structural node representation vectors and updating them such that the resulting embeddings still resemble the original structural ones and contain information about the node attributes. The question arises: how to design a method that respects the above-mentioned assumption? This thesis addressed this question by means of the **Research objective 4**.

Chapter 2

Research goal

In this chapter, the main focus is given to stating the research hypothesis along with four research objectives. Next, the list of scientific papers that constitute this thesis is provided. Finally, for each research objective, it is shown how the particular research articles within the thesis addressed the objective.

2.1 Research hypothesis and objectives

The research hypothesis of this dissertation is given as follows:

> *There exist unsupervised representation learning methods for graphs, whose embeddings can outperform those of state-of-the-art unsupervised approaches evaluated by means of downstream tasks.*

In order to validate the research hypothesis, it was substantiated into the following research objectives **RO1** - **RO4**.

RO1 Verify whether the usage of the cross-correlation measure applied to augmented views of an attributed graph allows training a graph neural network in a self-supervised setting, such that this GNN computes better node representation vectors than existing self-supervised node representation learning methods, measured by the performance in downstream tasks and training time complexity.

RO2 Develop a deep neural network model for calculating edge representation vectors that is trained using a combination of a contrastive learning objective with a feature reconstruction loss, such that this method's embedding vectors are better in downstream tasks than those obtained as aggregations of source and target node representations.

RO3 Develop an incremental learning method for node representation vectors in dynamic graphs, which utilizes any kind of static node embeddings from consecutive graph snapshots and exhibits a lower time and memory complexity than contemporary methods for representation learning on dynamic graphs while providing competitive quality measure gains.

RO4 Develop a method for fusing together node attribute information with a given precomputed structural node embedding, resulting in a single low-dimensional embedding that performs better in downstream tasks than the structural embedding, the node attributes or other attributed node representation learning methods.

All of the above-mentioned research objectives result from an in-depth literature review that allowed to identify such research gaps in the current body of knowledge for unsupervised representation learning methods for graphs.

2.2 List of publications

This thesis is a series of thematically related works in the form of the following five scientific publications:

P1 Piotr Bielak, Tomasz Kajdanowicz, Nitesh V. Chawla, *Graph Barlow Twins: A self-supervised representation learning framework for graphs*, Knowledge-Based Systems, Volume 256, 2022, 109631, ISSN 0950-7051, `https://doi.org/10.1016/j.knosys.2022.109631`.

IF 8.139, MEIN 200

P2 Piotr Bielak, Tomasz Kajdanowicz, Nitesh V. Chawla, *AttrE2vec: Unsupervised attributed edge representation learning*, Information Sciences, Volume 592, 2022, Pages 82-96, ISSN 0020-0255, `https://doi.org/10.1016/j.ins.2022.01.048`.

IF 8.233, MEIN 200

P3 Piotr Bielak, Kamil Tagowski, Maciej Falkiewicz, Tomasz Kajdanowicz, Nitesh V. Chawla, *FILDNE: A Framework for Incremental Learning of Dynamic Networks Embeddings*, Knowledge-Based Systems, Volume 236, 2022, 107453, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2021.107453.

IF 8.139, MEIN 200

P4 Piotr Bielak, Daria Puchalska, Tomasz Kajdanowicz, *Retrofitting Structural Graph Embeddings with Node Attribute Information*. In: Groen, D., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds) Computational Science – ICCS 2022. ICCS 2022. Lecture Notes in Computer Science, vol 13350. Springer, Cham, https://doi.org/10.1007/978-3-031-08751-6_13.

CORE A, MEIN 140

P5 Piotr Bielak, Jakub Binkowski, Albert Sawczyn, Katsiaryna Viarenich, Daria Puchalska, Tomasz Kajdanowicz, *A deeper look at Graph Embedding RetroFitting*, Journal of Computational Science, Volume 68, 2023, 101979, ISSN 1877-7503, https://doi.org/10.1016/j.jocs.2023.101979.

IF 3.817, MEIN 100

2.3 Addressing the research objectives

Let's now discuss how each scientific article addresses its corresponding research objective.

2.3.1 Research objective 1

Objective: *Verify whether the usage of the cross-correlation measure applied to augmented views of an attributed graph allows training a graph neural network in a self-supervised setting, such that this GNN computes better node representation vectors than existing self-supervised node representation learning methods, measured by the performance in downstream tasks and training time complexity.*

Scientific article: **[P1]** – *Graph Barlow Twins: A self-supervised representation learning framework for graphs*

Details: This article proposes a novel self-supervised graph representation learning framework called **Graph Barlow Twins**. The loss function utilizes the embedding cross-correlation matrix of two distorted graph views to optimize the representation vectors. The framework neither requires using negative samples (as opposed to most other self-supervised approaches) nor introduces any kind of asymmetry in the network architecture (like state-of-the-art BGRL). Moreover, the model converges substantially faster than all other state-of-the-art methods.

In particular, given an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and an encoder graph neural network f_θ, the first step is to obtain two augmented views $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}$ of the input graph. Each of those views is generated by applying two functions: node feature masking (which randomly selects a set of node attributes and puts zeros in those attributes for all nodes) and edge dropping (which randomly removes edges from the graph), both parametrized by hyperparameters p_X and p_A which denote the probability of masking a single feature and dropping a single edge, respectively. Next, both views are processed using the same instance of the encoder GNN f_θ to compute the node representation vectors for each view, i.e., $\mathbf{Z}^{(1)} = f_\theta(\mathcal{G}^{(1)})$ and $\mathbf{Z}^{(2)} = f_\theta(\mathcal{G}^{(2)})$. Further, those embedding matrices are used to compute the empirical cross-correlation matrix $\mathcal{C}$, as it was assumed in the research objective. Using a stochastic gradient descent optimizer – AdamW [42] – this cross-correlation matrix is optimized to approach the identity matrix. The following loss function is applied:

$$\mathcal{L}_{\mathrm{BT}} = \sum_i (1 - \mathcal{C}_{ii})^2 + \lambda \sum_i \sum_{j \neq i} {\mathcal{C}_{ij}}^2, \tag{2.1}$$

where λ is a method hyperparameter. Such a learning objective enforces two effects: (1) as the diagonal elements of the cross-correlation matrix approach a value of one, the encoder model learns to ignore the applied augmentation functions, (2) by forcing the off-diagonal element toward zero (with a trade-off defined by λ) each pair of features in the representation vector is being decorrelated, i.e., features become independent and more knowledge about the graph structure, and node attributes can be stored in the representation vectors.

The experimental evaluation included a variety of downstream task scenarios: (1) transductive node classification for 5 smaller benchmark datasets, (2) transductive node classification using the medium-sized ogb-arxiv dataset from the Open Graph Benchmark [46], (3) inductive node classification for multiple graphs using the PPI (Protein-Protein Interaction) dataset, and (4) inductive node classification for the large-scale graph ogb-products dataset. Depending on the case, both GCN-based encoders and a GAT-based one were evaluated. The proposed method achieved analogous

results compared to state-of-the-art methods, however, it requires substantially fewer hyperparameters and converges in an order of magnitude training steps earlier, leading to an overall speedup of up to 42 times compared to state-of-the-art models.

Moreover, ablation studies revealed that using a projector network, which is often used in other approaches, does not improve the embedding quality and can be omitted, reducing the number of trainable model parameters. To obtain the augmented graph views, other methods use a different set of augmentation function hyperparameters for each augmentation branch (p_X, p_A in this case). In contrast, the proposed Graph Barlow Twins model computes robust embedding vectors even using the same augmentation function hyperparameters for both branches. This also allowed to reduce the number of model hyperparameters. Finally, choosing the λ hyperparameter in the loss function is non-trivial. Experiments revealed that using $\lambda = \frac{1}{d}$, where d is the embedding vector dimensionality, provides the best overall results regarding downstream task performance.

2.3.2 Research objective 2

Objective: *Develop a deep neural network model for calculating edge representation vectors that is trained using a combination of a contrastive learning objective with a feature reconstruction loss, such that this method's embedding vectors are better in downstream tasks than those obtained as aggregations of source and target node representations.*

Scientific article: **[P2]** – *AttrE2vec: Unsupervised attributed edge representation learning*

Details: This article proposes a novel inductive method, called **AttrE2vec**, for learning representations of edges in attributed graphs in an unsupervised manner. Using random walks and an aggregation function, it summarizes the neighborhoods of a given edge and passes those along with the edge features into a deep neural network encoder module that finally computes the edge representation vector. The loss function is twofold, where one term is a contrastive learning objective (allowing to model structural similarity), and the second term is a feature reconstruction loss (that, along with a trainable decoder module, ensures that the initial edge attribute information is preserved in the edge embedding vector). The model builds edge representation vectors without using intermediate node representations. The experimental evaluation shows that AttrE2vec builds representation vectors that achieve state-of-the-art performance on edge-related downstream tasks.

In particular, for a given attributed edge (u, v), the first step is to decompose the neighborhood of the edge into two smaller ones – one constituted by edges connected to node u and the other one by edges connected to node v. Next, the method performs several uniform random walks starting from node u and v, respectively. Such a procedure allows to explore the neighborhoods and scale up to large graphs. A walk aggregation function $\mathbf{Agg_w}(\cdot)$ is applied on each random walk to aggregate the attributes of visited edges. Several options for this aggregation function are considered: average, weighted average (with exponential decaying, where more distant edges get a lower weight), aggregation using Gated Recurrent Units (GRU) [21] over the edge attribute vectors, or aggregation using GRUs over the concatenated edge and node attributes. Having now one feature vector per random walk, another aggregation function $\mathbf{Agg_n}(\cdot)$ is applied. As the random walks have no particular ordering, this aggregation function is a permutation-invariant function in the form of an average over the vectors. At this point, each edge is described by three vectors: summary vector of the neighborhood defined by node u, i.e., $\mathbf{S}_u$, summary vector of v's neighborhood, i.e., $\mathbf{S}_v$, and the edge features itself $\mathbf{f}_{uv}$. These three vectors are passed into a deep neural network encoder, which is built as a multi-layer perceptron (MLP) with a self-attention mechanism. It outputs the final edge representation vector $\mathbf{h}_{uv}$. As mentioned above, the model is trained using a twofold loss function:

$$\mathcal{L} = \lambda * \mathcal{L}_{\cos} + (1 - \lambda) * \mathcal{L}_{\text{MSE}}, \tag{2.2}$$

where $\mathcal{L}_{\cos}$ is a contrastive loss function, which utilizes the cosine similarity measure to increase the similarity between anchor examples and positive samples while reducing the similarity between anchor examples and negative samples. In the case of AttrE2vec, for each edge, the positive samples are edges sampled from the set of edges visited by random walks, while negative samples are edges from the graph that were not visited by random walks. The term $\mathcal{L}_{\text{MSE}}$ denotes a mean squared error computed between the original edge features and the output of a decoder module that takes in the edge representation vector $\mathbf{h}_{uv}$. To define a trade-off between both terms, the λ hyperparameter is introduced.

Compared to contemporary approaches, the experimental evaluation shows that AttrE2vec builds more powerful edge vector representations, reflected both by low-dimensional embedding projections and higher quality measures (AUC, accuracy) in downstream tasks, such as edge classification and clustering. In particular, the proposed method achieves better performance than approaches that first compute node representations and then transform them into edge-level representation vectors.

To sum up, the article proposed a novel unsupervised method (AttrE2vec) for learning low-dimensional vector representation for attributed edges. The method is inductive and allows getting the representation of edges not present in the training phase. Various experiments show that AttrE2vec has superior performance over all of the baseline methods on edge classification and clustering tasks.

2.3.3 Research objective 3

Objective: *Develop an incremental learning method for node representation vectors in dynamic graphs, which utilizes any kind of static node embeddings from consecutive graph snapshots and exhibits a lower time and memory complexity than contemporary methods for representation learning on dynamic graphs while providing competitive quality measure gains.*

Scientific article: **[P3]** – *FILDNE: A Framework for Incremental Learning of Dynamic Networks Embeddings*

Details: This article proposes a novel framework for learning node representations in dynamic graphs using an incremental learning approach called **FILDNE**. It is trained in an unsupervised manner and integrates representation vectors computed using static node representation learning methods over different timesteps into a single representation by developing a convex combination function and alignment mechanism. The combination weights are learned using a Bayesian inference mechanism.

There are two proposed framework variants: *FILDNE*, which considers exactly two graph snapshots (embedding matrices) at each iteration, and *k-FILDNE*, which generalizes to k graph snapshots at each iteration. The framework is incrementally applied to the embeddings of the graph snapshot sequence so that the first embedding is always an aggregation from the last iteration. In particular, given a dynamic graph $\mathcal{G}_{0,T}$ which is modeled as a sequence of graph snapshots $[\mathcal{G}_{0,1}, \mathcal{G}_{1,2}, \ldots, \mathcal{G}_{T-1,T}]$, the first step is to apply an existing node representation learning method, such as Node2vec [78] or LINE [87], on each snapshot to obtain a sequence of embeddings $[\mathbf{F}_{0,1}, \mathbf{F}_{1,2}, \ldots, \mathbf{F}_{T-1,T}]$. Next, due to the stochastic nature of some representation learning methods, an embedding alignment method, based on the Orthogonal Procrustes method, is applied to the embedding matrices. The aligned embedding vectors of a given node are combined into a single representation using a convex linear combination function. For the basic *FILDNE* model, this function is given as follows:

$$\tilde{\mathbf{F}}_{0,t} = \alpha\tilde{\mathbf{F}}_{0,t-1} + (1-\alpha)\mathbf{F}^*_{t-1,t}, \tag{2.3}$$

where $\tilde{\mathbf{F}}_{0,t-1}$ is the node embedding for the interval $[0, t-1]$ (computed by the framework in the previous iteration), $\mathbf{F}^*_{t-1,t}$ is the node embedding of the current graph snapshot (after embedding alignment), and α is the model's hyperparameter, which can be found either using a grid search or using expert knowledge.

In the case of the *k-FILDNE* model the combination function is given as follows:

$$\tilde{\mathbf{F}}_{0,t} = \alpha_1\tilde{\mathbf{F}}_{0,t-k+1} + \alpha_2\mathbf{F}^*_{t-k+1,t-k+2} + \ldots + \alpha_k\mathbf{F}^*_{t-1,t}, \tag{2.4}$$

where $\tilde{\mathbf{F}}_{0,t-k+1}$ denotes the node embedding for the interval $[0, t-k+1]$ (computed by the framework in the previous iteration), $[\mathbf{F}^*_{t-k+1,t-k+2}, \ldots, \mathbf{F}^*_{t-1,t}]$ are aligned node embeddings for their respective time intervals, and $[\alpha_1, \alpha_2, ..., \ldots, \alpha_k]$ are the combination parameters. For *k-FILDNE*, these parameters are computed automatically based on the graph data. Namely, they are the Maximum A Posteriori estimate of the Dirichlet-Multinomial model, where the likelihood is modeled using an unsupervised link prediction task over the graph snapshots.

The experimental evaluation was performed on several downstream tasks (link prediction, edge classification, graph reconstruction) over seven real-world datasets. The results show that FILDNE is able to reduce memory (up to 6x) and computational time (up to 50x) costs while providing competitive quality measure gains (e.g., improvements up to 19 pp AUC on link prediction and up to 33 pp mAP on graph reconstruction) with respect to the contemporary methods.

2.3.4 Research objective 4

Objective: *Develop a method for fusing together node attribute information with a given precomputed structural node embedding, resulting in a single low-dimensional embedding that performs better in downstream tasks than the structural embedding, the node attributes or other attributed node representation learning methods.*

Scientific articles:

- **[P4]** – *Retrofitting Structural Graph Embeddings with Node Attribute Information*
- **[P5]** – *A deeper look at Graph Embedding RetroFitting*

Details: The first article ([**P4**]) proposes a novel method, called **GERF** (Graph Embedding RetroFitting), for updating existing structural node representation vectors using node attribute information. Instead of utilizing naive approaches, such as concatenation of the structural vectors with the node attribute vectors, which would create a non-consistent vector space and also result in potentially high-dimensional vectors, GERF learns a new node embedding $\mathbf{Z}^*$ that is optimized using stochastic gradient descent optimizers to preserve the information encoded in both the structural embeddings and node attributes.

In particular, GERF randomly initializes the $\mathbf{Z}^* \in \mathbb{R}^{|\mathcal{V}| \times d}$ matrix, where d is the dimensionality of the structural representation vectors, and then optimizes the following graph retrofitting objective:

$$\begin{aligned}\mathcal{L}(\mathbf{Z}^*) = (1 - \lambda_G - \lambda_X) \sum_{i=1}^{n} \|\mathbf{z}_i^* - \mathbf{z}_i\|^2 \\ + \lambda_G \sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}(v_i)} \frac{\|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2}{|\mathcal{N}(v_i)|} + \lambda_X \sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}_{\mathbf{X}}(v_i)} \frac{\|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2}{|\mathcal{N}_{\mathbf{X}}(v_i)|},\end{aligned} \tag{2.5}$$

where $\mathbf{Z}$ is the structural node embedding, $\mathcal{N}(u)$, is the neighborhood of node u defined by edges (the graph structure), $\mathcal{N}_{\mathbf{X}}(u)$ is the so-called attribute neighborhood of node u (it is defined as the K nearest neighbors of u in the node attribute space; K is set equal to the number of structural neighbors of this node in the graph), and λ_G, λ_X are method hyperparameters that define a trade-off between the three components of the objective function.

The first component $\sum_{i=1}^{n} \|\mathbf{z}_i^* - \mathbf{z}_i\|^2$ (invariance loss) ensures that the new representation vectors are close to the structural ones (so the information is preserved). The second term $\sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}(v_i)} \frac{\|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2}{|\mathcal{N}(v_i)|}$ (graph neighbor loss) ensures that nodes connected by edges obtain similar representation vectors, whereas the third component $\sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}_{\mathbf{X}}(v_i)} \frac{\|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2}{|\mathcal{N}_{\mathbf{X}}(v_i)|}$ (attribute neighbor loss) moves representation vectors of nodes with similar attributes closer together.

Experiments on four real-world benchmark datasets show that the representation vectors obtained using GERF achieve better results in a node classification task compared to: (1) purely structural embeddings, (2) naive approaches to fusing structural and attribute information, (3) node representation learning methods designed for attributed graphs.

However, as mentioned above the proposed GERF method requires two hyperparameters λ_G, λ_X to be defined, either by a grid search (that depending on the dataset size and grid resolution might be time-consuming) or by means of expert knowledge. Moreover, by a closer inspection of the objective function, one might claim that the graph neighbor loss is redundant, as the information about the graph structure (edges) is already encoded in the structural representation vectors. In the second article ([**P5**]) these issues were addressed.

In particular, an updated version of the model was introduced – **GERF++**. Its objective function was simplified by removing the graph neighbor loss:

$$\mathcal{L}(\mathbf{Z}^*) = (1 - \lambda_X) \sum_{i=1}^{n} \|\mathbf{z}_i^* - \mathbf{z}_i\|^2 + \lambda_X \sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}_{\mathbf{X}}(v_i)} \frac{\|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2}{|\mathcal{N}_{\mathbf{X}}(v_i)|}, \tag{2.6}$$

where all symbols are the same as in the original GERF model. Note that now, there is only one hyperparameter λ_X. To address the second issue of finding the best value for that hyperparameter, an estimation algorithm that is based on the Dirichlet-Multinomial model was proposed. The idea is similar to the one utilized in the FILDNE framework, but in this case, the classifier models are applied to the structural node representation vectors and the node attributes, respectively. Based on the number of successful classification attempts and the Maximum A Posteriori of the Dirichlet-Multinomial model, the λ_X hyperparameter is computed. Therefore, the GERF++ model can be applied to any dataset without the need to define hyperparameters.

In the experimental evaluation, the proposed model was not only evaluated in the node classification task but also in a link prediction task. Again, the GERF++ method achieved the best performance compared to structural node representation learning methods, naive fusion approaches, and node representation learning methods for attributed graphs.

Chapter 3

Conclusions

The studies conducted within this dissertation explored the realm of unsupervised methods for representation learning of graphs. An in-depth literature review allowed to identify a total of four research problems and gaps. Based on those, four research objectives were defined and executed. The results are presented in five scientific articles, published in high-impact journals, and a highly ranked conference. In particular, they showed that:

- the utilization of the cross-correlation measure between node representation vectors of two augmented views of a graph allowed to train a graph neural network in a self-supervised manner without requiring negative samples, such that the embeddings achieved analogous quality to state-of-the-art methods but converging in significantly fewer training iterations,
- training a deep neural networks model using a compound loss function, consisting of a contrastive learning objective and a feature reconstruction loss, allowed building edge representation vectors that achieved better results in edge-related downstream tasks than state-of-the-art methods and did not require the calculation of source and target node representations aggregations,
- node representation vectors for dynamic graphs can be obtained as a linear convex combination of static embeddings with combination parameters estimated by a Dirichlet-Multinomial model using the outcomes of an unsupervised link prediction task; the resulting embeddings achieve competitive quality measure gains with respect to the contemporary methods, and the method reduces memory and computational time costs,

- using a method based on a compound objective function, consisting of an invariance loss and attribute neighbor loss, allows updating precomputed structural node representation vectors with node attribute information, such that the resulting embedding vectors achieve better results in downstream tasks than existing attributed node representation learning methods.

Chapter 4

Publications

4.1 Graph Barlow Twins: A self-supervised representation learning framework for graphs

Authors:	Piotr Bielak, Tomasz Kajdanowicz, Nitesh V. Chawla
Published at:	Knowledge-Based Systems (IF 8.139)
MEIN points:	200
Citations:	Web of Science: 3
(as of 05.06.2023)	Scopus: 4
	Google Scholar: 50
Credit:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization

Knowledge-Based Systems 256 (2022) 109631

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Graph Barlow Twins: A self-supervised representation learning framework for graphs

Piotr Bielak [a], Tomasz Kajdanowicz [a,*], Nitesh V. Chawla [b]

[a] Department of Artificial Intelligence, Wroclaw University of Science and Technology, Poland
[b] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA

ARTICLE INFO

Article history:
Received 14 April 2022
Received in revised form 3 August 2022
Accepted 4 August 2022
Available online 17 August 2022

Keywords:
Representation learning
Self-supervised learning
Graph embedding

ABSTRACT

The self-supervised learning (SSL) paradigm is an essential exploration area, which tries to eliminate the need for expensive data labeling. Despite the great success of SSL methods in computer vision and natural language processing, most of them employ contrastive learning objectives that require negative samples, which are hard to define. This becomes even more challenging in the case of graphs and is a bottleneck for achieving robust representations. To overcome such limitations, we propose a framework for self-supervised graph representation learning — *Graph Barlow Twins*, which utilizes a cross-correlation-based loss function instead of negative samples. Moreover, it does not rely on non-symmetric neural network architectures — in contrast to state-of-the-art self-supervised graph representation learning method *BGRL*. We show that our method achieves as competitive results as the best self-supervised methods and fully supervised ones while requiring fewer hyperparameters and substantially shorter computation time (ca. 30 times faster than BGRL).

1. Introduction

Graph representation learning has been intensively studied for the last few years, having proposed various architectures and layers, like GCN [1], GAT [2], GraphSAGE [3] etc. A substantial part of these methods was introduced in the semi-supervised learning paradigm, which requires the existence of expensive labeled data (e.g. node labels or whole graph labels). To overcome this, the research community has been exploring unsupervised learning methods for graphs. This resulted in a variety of different approaches including: shallow ones (DeepWalk [4], Node2vec [5], LINE [6]) that ignore the feature attribute richness, focusing only on the structural graph information; and graph neural network methods (DGI [7], GAE, VGAE [8]) that build representations upon node or graph features, achieving state-of-the-art performance in those days.

Recently self-supervised paradigm is the most emerging branch of unsupervised graph representation learning and gathers current interest and strenuous research effort towards better results. The most prominent methods were developed around the contrastive learning approach, such as GCA [9], GraphCL [10], GRACE [11] or DGI [7]. Although contrastive methods are popular in many machine learning areas, including computer vision and natural language processing, their fundamental limitation is the need for negative samples. Consequently, the sampling procedure for negative examples highly affects the overall quality of the embeddings. In terms of images or texts, the definition of negative samples might seem not that problematic, but in the case of graphs there is no clear intuition. For instance, what is the negative counterpart for a particular node in the graph, should it be a node that is not a direct neighbor, or a node that is in a different graph component? There are multiple options available, but the right choice strictly dependent on the downstream task.

Researchers have already tackled the problem of building so-called *negative-sample-free* methods. In research being conducted in computer vision they obtained successful results with methods such as BYOL [12], SimSiam [13] or Barlow Twins [14]. These models utilize siamese network architectures with various techniques, like gradient stopping, asymmetry or batch and layer normalizations, to prevent collapsing to trivial solutions. Based on BYOL, [15] proposed the Bootstrapped Representation Learning on Graphs (BGRL) framework. It utilizes two graph encoders: an online and a target one. The former one passes the embedding vectors to a predictor network, which tries to predict the embeddings from the target encoder. The loss is measured as the cosine similarity and the gradient is backpropagated only through the online network (gradient stopping mechanism). The target encoder is updated using an exponential moving average of the online encoder weights. Such setup has been shown to produce graph representation vectors that achieve state-of-the-art performance in node classification using various benchmark datasets. Notwithstanding, assuming asymmetry between

* Corresponding author.
E-mail address: tomasz.kajdanowicz@pwr.edu.pl (T. Kajdanowicz).

https://doi.org/10.1016/j.knosys.2022.109631

the network twins (such as the predictor network, gradient stopping, and a moving average on the weight updates) the method is conceptually complex.

Employing a symmetric network architecture would seem more intuitive and reasonable, hence in this paper, we propose **Graph Barlow Twins (G-BT)**, a self-supervised graph representation learning framework, which computes the embeddings cross-correlation matrix of two distorted views of a single graph. The approach was firstly introduced in image representation learning as the Barlow Twins model [14] but was not able to handle graphs. The utilized network architecture is fully symmetric and does not need any special techniques to build non trivial embedding vectors. The distorted graph views are passed through the same encoder which is trained using the backpropagated gradients (in a symmetrical manner).

Our approach differs from previous application of Barlow Twins cost function in terms of: pivoting the Barlow Twins loss on graph data and selection of appropriate encoder for such case, data augmentation functions (and their hyperparameters), simplified part of neural network structure that is not necessary to apply in graph case, and provided experimental results for the batched scenario.

Our main contributions can be summarized as follows:

1. We propose a self-supervised graph representation learning framework Graph Barlow Twins. It is built upon the recently proposed Barlow Twins loss, which utilizes the embedding cross-correlation matrix of two distorted views of a graph to optimize the representation vectors. Our framework neither requires using negative samples (opposed to most other self-supervised approaches) nor it introduces any kind of asymmetry in the network architecture (like state-of-the-art BGRL). Moreover, our architecture is converges substantially faster than all other state-of-the-art methods.
2. We evaluate our framework in node classification tasks: (1) for 5 smaller benchmark datasets in a transductive setting, (2) using the ogb-arxiv dataset from the Open Graph Benchmark (also in the transductive setting), (3) for multiple graphs in the inductive setting using the PPI (Protein–Protein Interaction) dataset, and finally (4) for the large-scale graph dataset ogb-products in the inductive setting. We use both GCN-based encoders as well as a GAT-based one. We observe that our method achieves analogous results compared to state-of-the-art methods.
3. We ensure reproducibility by making the code of both our models as well as experimental pipeline available: https://github.com/pbielak/graph-barlow-twins.

2. Related works

Self-supervised learning. The idea of self-supervised learning (SSL) has a long history. Introduced in the early work of Schmidhuber [16] has more than 30 years of exploration and research now. Recently self-supervised learning was again rediscovered and found a broad interest, especially in computer vision and natural language processing. One of the most prominent SSL methods for image representation learning, Bootstrap Your Own Latent, BYOL [12], performs on par or better than the current state of the art on both transfer and semi-supervised benchmarks. It relies on two neural networks that interact and learn from each other. From an augmented view of an image, it trains the first one to predict the target network representation of the same image under a different view. At the same time, it updates the second network with a slow-moving average of the first network. Another approach to image representation SSL implements simple siamese networks, namely SimSiam [13]. It achieves comparative results while not demanding negative samples, large batches, nor momentum encoders. Authors emphasize collapsing solutions for the loss and structure but show how a stop-gradient operation plays an essential role in preventing it. Recent method, Barlow Twins [14], advances the SSL field with a new objective function that naturally avoids collapses by measuring the cross-correlation matrix between the outputs of two twin, identical networks fed with distorted versions of a sample, and makes it as close to the identity matrix as possible. Representations of distorted versions of samples are then expected to be similar, reducing the redundancy between them. What differentiates the method is that it does not require large batches or asymmetry between the network twins. It outperforms previous methods on ImageNet for semi-supervised classification.

Graph representation learning. Learning the representation also spreads to other domains. The graph embedding problem has also attracted much attention from the research community worldwide in recent years. Plenty of methods have been developed, each focused on a different aspect of network embeddings, such as proximity, structure, attributes, learning paradigm or scalability. There exist plenty of shallow methods, among others DeepWalk [4], Node2vec [5] or LINE [6], that use a simple notion of graph coding through random walks or on encoder–decoder objectives that optimize first and second-order node similarity. More complex graph neural networks, such as GCN [1] or GraphSAGE [3] implements the basic encoder algorithm with various neighborhood aggregation. Following the extension, graph attention network GAT [2] leverages masked self-attentional layers to address the shortcomings of graph convolutions and their troublesome approximations.

Self-supervised graph representation learning. Inspired by the success of contrastive methods in vision and NLP, the procedures were also adapted to graphs. Early DGI [7] employs GNN to learn node embeddings and obtains the graph embedding via a readout function and maximizes the mutual information between node embeddings and the graph embedding by discriminating nodes in the original graph from nodes in a corrupted graph. GCA [9] studied various augmentation procedures. GRACE [11] creates two augmented versions of a graph, pulls together the representation of the same node in both graphs, and pushes apart representations of every other node. Recent GraphCL [10] method is another example of representative approach using contrastive learning. All the previous methods use negative sampling approaches for the embedding optimization, yet such setting has a high complexity. To overcome this, BGRL [15] proposed to use an approach that does not rely on negative samples. It uses two kinds of encoder networks (online and target), introducing a non-intuitive asymmetric pipeline architecture, but provides state-of-the-art SSL results. Moreover, it relies on several techniques to prevent trivial solutions (gradient stopping, momentum encoder). A concurrent approach to BGRL is DGB [17].

3. Proposed framework

Motivated by the emerging self-supervised learning paradigm and its recent applications in graph representation learning (BGRL [15]), we propose **Graph Barlow Twins** — a framework that builds node embeddings using a symmetric network architecture and an empirical cross-correlation based loss function.

We start with an attributed graph $\mathcal{G}$. Using carefully selected augmentation functions $\mathcal{A}_{p_X, p_A}(\cdot)$, we compute two augmented versions of this graph, i.e., $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$. Each time the augmentation function is applied, it yields a different graph version. This is due the fact that it is performed using parameters sampled according to p_A and p_X. Next, we apply the same encoder network

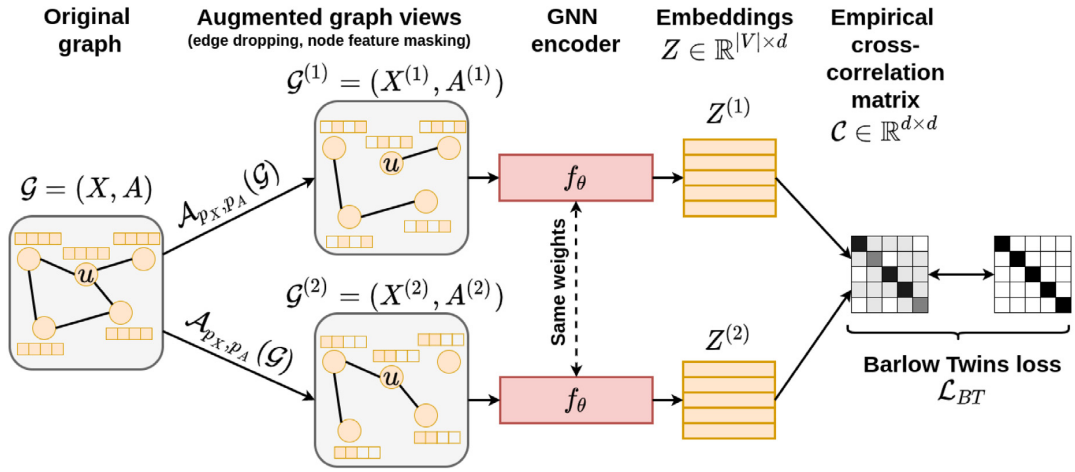


Fig. 1. Overview of our proposed Graph Barlow Twins framework. We transform an input graph $\mathcal{G}$ using an augmentation function and obtain two views: $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$. We pass both of them through the same GNN encoder f_θ to compute two embedding matrices $Z^{(1)}$, $Z^{(2)}$. We build a loss function such that the embeddings' empirical cross-correlation matrix $\mathcal{C}$ is optimized into the identity matrix.

(which is being pretrained using our proposed framework) on both created graphs and obtain two node embedding matrices — $Z^{(1)}$ and $Z^{(2)}$. Finally, we compute the empirical cross-correlation matrix $\mathcal{C}$ of these node embeddings and compute a loss function which forces the cross-correlation matrix to be as close as possible to the identity matrix. The encoder is trained by backpropagating the gradient of such loss function.

The overall pipeline of our framework is visually shown in Fig. 1, and the precise algorithm is presented in Algorithm 1. Let us now describe each framework step in detail:

Algorithm 1: Graph Barlow Twins training

Input: attributed graph $\mathcal{G}$,
augmentation function $\mathcal{A}_{p_X, p_A}(\cdot)$,
encoder network $f_\theta(\cdot)$,
number of epochs N,
learning rate η
Output: trained encoder network f_θ

for $i \leftarrow 1$ **to** N **do**
  // Generating graph views via augmentation
  $\mathcal{G}^{(1)} \leftarrow \mathcal{A}_{p_X, p_A}(\mathcal{G})$ // Eq. (1)
  $\mathcal{G}^{(2)} \leftarrow \mathcal{A}_{p_X, p_A}(\mathcal{G})$ // Eq. (1)
  // Computing node embeddings
  $Z^{(1)} \leftarrow f_\theta(\mathcal{G}^{(1)})$
  $Z^{(2)} \leftarrow f_\theta(\mathcal{G}^{(2)})$
  // Loss function
  $\mathcal{C} \leftarrow$ `cross-correlation`$(Z^{(1)}, Z^{(2)})$ // Eq. (2)
  // Optimizing encoder weights
  $\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \mathcal{L}_{BT}(\mathcal{C})$ // Eq. (3)
end

Graph data. We represent a graph $\mathcal{G}$ with nodes $\mathcal{V}$ and edges $\mathcal{E}$ as the tuple: (X, A), where $X \in \mathbb{R}^{|\mathcal{V}| \times k}$ is the node feature matrix and k is the feature dimensionality; $A \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the adjacency matrix, such that $A_{i,j} = 1$ iff $(i, j) \in \mathcal{E}$. In the general case, a graph could also have associated edge features or graph level features, but for simplicity we omit those here. Nevertheless, these could also be used in our framework, as long as the encoder can make use of such features.

Generating graph views via augmentation. Following other works [9–11,15], we select two kinds of augmentations – *edge dropping* and *node feature masking* – and generate two views of the input graph $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$ (see Eq. (1)). In the edge dropping case $\mathcal{A}_{p_A}(A)$, we remove edges according to a generated mask of size $|\mathcal{E}|$ (number of edges in the graph) with elements sampled from the Bernoulli distribution $\mathcal{B}(1 - p_A)$, where p_A is the probability of dropping an edge. When it comes to masking node features $\mathcal{A}_{p_X}(X)$, we employ a similar scheme and generate a mask of size k also sampled from the Bernoulli distribution $\mathcal{B}(1 - p_X)$, where p_X is the probability of masking features at nodes. Note that we mask node features at the scale of the whole graph, i.e. the same features are masked for each node. Other works apply different augmentation parameters p_X, p_A for each generated view, but as our framework is fully symmetrical, we postulate that it is enough to use the same parameters to generate both augmentations (see Section 5.1).

$$\mathcal{A}_{p_X, p_A}(\mathcal{G}) := (\mathcal{A}_{p_X}(X), \mathcal{A}_{p_A}(A)) \tag{1}$$

Encoder network for node embeddings. The main component of the proposed framework is the encoder network $f_\theta : \mathcal{G} \to \mathbb{R}^{|\mathcal{V}| \times d}$. It takes an augmented graph as the input and computes (in our case) a d-dimensional representation vector for each node in the graph. Note that we do not specify any particular encoder network and one may use even encoders that construct embeddings for edges or whole graphs. In our experiments, we will show the application of GCN [1] and GAT [2] based encoder networks. Both augmented graph views $\mathcal{G}^{(1)}$, $\mathcal{G}^{(2)}$ are *passed through the same encoder*, resulting in two embedding matrices $Z^{(1)}$ and $Z^{(2)}$, respectively. The original Barlow Twins method specified also a projector network (implemented as an MLP) to reduce high embedding dimensionality (of the ResNet encoder). Our approach eliminates that step as it uses GNNs with low dimensional embeddings.

Loss function. In our work, we propose to use a *negative-sample-free* loss function to train the encoder network. We first normalize the embedding matrices $Z^{(1)}$ and $Z^{(2)}$ along the batch dimension (a mean of zero and a standard deviation equal to one), and then

we compute the empirical cross-correlation matrix $\mathcal{C} \in \mathbb{R}^{d \times d}$:

$$\mathcal{C}_{ij} = \frac{\sum_b Z^{(1)}_{b,i} Z^{(2)}_{b,j}}{\sqrt{\sum_b (Z^{(1)}_{b,i})^2} \sqrt{\sum_b (Z^{(2)}_{b,j})^2}}, \tag{2}$$

where b are the batch indexes and i, j are the indexes of embeddings. Such setting was originally proposed under the name **Barlow Twins**. Neuroscientist H. Barlow's *redundancy-reduction principle* has motivated many methods both in supervised and unsupervised learning [18–20]. Recently, [14] has employed this principle to build a self-supervised image representation learning algorithm (we bring this idea to the domain of graph-structured data).

The cross-correlation matrix $\mathcal{C}$ is optimized by the Barlow Twins loss function $\mathcal{L}_{\text{BT}}$ (see Eq. (3)) to be equal to the identity matrix. The loss is composed of two parts: (I) the invariance term and (II) the redundancy reduction term. The first one forces the on diagonal elements $\mathcal{C}_{ii}$ to be equal to one, hence making the embeddings invariant to the applied augmentations. The second term optimizes the off-diagonal elements $\mathcal{C}_{ij}$ to be equal to zero — this results in decorrelated components of the embedding vectors.

$$\mathcal{L}_{\text{BT}} = \sum_i (1 - \mathcal{C}_{ii})^2 + \lambda \sum_i \sum_{j \neq i} \mathcal{C}_{ij}{}^2 \tag{3}$$

The $\lambda > 0$ parameter defines the trade-off between the invariance and redundancy reduction terms when optimizing the overall loss function. In [21], the authors proposed to use $\lambda = \frac{1}{d}$, which we employ in our experimental setting (see Section 5.3). Otherwise, one can perform a simple grid search to find the best λ value in a particular experiment.

Please note, that in such setting the gradient is symmetrically backpropagated through the encoder network. We do not rely on any special techniques, like momentum encoders, gradient stopping, or predictor networks. In preliminary experiments, we also investigated the Hilbert–Schmidt Independence Criterion (due to its relation to the Barlow Twins objective [21]), but we did not observe any performance gain.

4. Experiments

We evaluate the performance of our model using a variety of popular benchmark datasets, including smaller ones, such as WikiCS, Amazon-Photo or Coauthor-CS, as well as larger ones, such as ogb-arxiv, ogb-products, provided by the Open Graph Benchmark [22]. In this section, we will provide an overview of the utilized datasets and the experimental scenario details, as well as the discussion of the results. Overall, we use a similar experimental setup, as the state-of-the-art self-supervised graph representation learning method BGRL [15], so we can perform a fair comparison to this method. To track our experiments and provide a simple way for reproduction, we employ the Data Version Control tool (DVC [23]) - for details see Appendix D. We perform all experiments on a TITAN RTX GPU with 24 GB RAM.

4.1. Datasets

Following, we provide brief descriptions for each dataset, including the basic statistics (see Table 1) and the examined dataset split type for the node classification downstream task:

- **WikiCS** [24] is a network of Computer Science related Wikipedia articles with edges denoting references between those articles. Each article belongs to one of 10 subfields (classes) and has features computed as averaged GloVe embeddings of the article content. We use the provided 20 train/val/test data splits without any modifications.

Table 1
Dataset statistics. We use small to medium sized standard datasets together with the larger ogb-arxiv dataset in the transductive setting. We also evaluate the inductive setting using the ogb-products and PPI (multiple graphs) dataset.

Name	Nodes	Edges	Features	Classes
WikiCS	11,701	216,123	300	10
Amazon Computers	13,752	245,861	767	10
Amazon Photos	7,650	119,081	745	8
Coauthor CS	18,333	81,894	6805	15
Coauthor Physics	34,493	247,962	8415	5
ogb-arxiv	169,343	1,166,243	128	40
PPI (24 graphs)	56,944	818,716	50	121 (multilabel)
ogb-products	2,449,029	61,859,140	100	47

- **Amazon Computers, Amazon Photos** [25] are two networks extracted from Amazon's co-purchase data. Nodes are products and edges denote that these products were often bought together. Each product is described using a Bag-of-Words representation (node features) based on the reviews. There are 10 and 8 product categories (node classes), respectively. For these datasets there are no data splits available, so similar to BGRL, we generate 20 random train/val/test splits (10%/10%/80%).
- **Coauthor CS, Coauthor Physics** are two networks extracted from the Microsoft Academic Graph [26]. Node are authors and edges denote a collaboration of two authors. Each author is described by the keywords used in their articles (Bag-of-Words representation; node features). There are 15 and 5 author research fields (node classes), respectively. Similarly to the Amazon datasets there are no data splits provided, so we generate 20 random train/val/test splits (10%/10%/80%).
- **ogb-arxiv** is a larger graph from the Open Graph Benchmark [22] with about 170 thousand nodes and about 1.1 million edges. The graph was extracted from the Microsoft Academic Graph [26], where nodes represents a Computer Science article on the arXiv platform and edges denote citations across papers. The node features are build as word2vec embeddings of the whole article content. There are 40 subject areas a node can be classified into (node label/class). The ogb-arxiv dataset provides a single train/val/test split, so we use it without any modifications, but we retrain the whole framework 20 times.
- **Protein–Protein Interaction (PPI)** [27] consists of 24 separate graphs. Each node in a single graph represents a protein, described by 50 biological features, and edges denote interactions among those proteins. There are 121 node labels, but note that contrary to other cases, PPI uses multilabel classification, i.e. a single protein can be assigned with multiple labels. Aligned with other methods, we provide results in terms of the Micro-F1 score. For PPI, there exists a predefined data split, where 20 graphs are used for training, 2 graphs for validation and 2 graphs for testing. Note that the validation and test graphs are completely unobserved during training time, hence the model is more challenged during inference time.
- **ogn-products** is a large-scale graph from the Open Graph Benchmark [22] with about 2.4 million nodes and 62 million edges. The graph was extracted from the Amazon product co-purchasing network. Nodes represent products from the Amazon store and edges denote whether two products were bought together. There are 100 node features, which are obtained from bag-of-words products descriptions reduced using PCA. Each product (node) can be classified into one of 47 categories (node labels). This dataset comes with a predefined data split, so we use as is.

Table 2
Mean and std accuracy of transductive node classification over 20 data splits and initializations obtained in BGRL paper [15] and our experiment (∗) within the same experimental setup. *OOM* denotes running out of memory on a 16GB V100 GPU.

	WikiCS	Am-CS	Am-Photo	Co-CS	Co-Physics
Raw features	71.98 ± 0.00	73.81 ± 0.00	78.53 ± 0.00	90.37 ± 0.00	93.58 ± 0.00
DeepWalk	74.35 ± 0.06	85.68 ± 0.06	89.44 ± 0.11	84.61 ± 0.22	91.77 ± 0.15
DeepWalk + fts	77.21 ± 0.03	86.28 ± 0.07	90.05 ± 0.08	87.70 ± 0.04	94.90 ± 0.09
DGI	75.35 ± 0.14	83.95 ± 0.47	91.61 ± 0.22	92.15 ± 0.63	94.51 ± 0.52
MVGRL	77.52 ± 0.08	87.52 ± 0.11	91.74 ± 0.07	92.11 ± 0.12	95.33 ± 0.03
GRACE (10k epochs)	80.14 ± 0.48	89.53 ± 0.35	92.78 ± 0.45	91.12 ± 0.20	OOM
BGRL (10k epochs)	79.36 ± 0.53	89.68 ± 0.31	92.87 ± 0.27	93.21 ± 0.18	95.56 ± 0.12
G-BT∗ (≤1k epochs)	76.65 ± 0.62	88.14 ± 0.33	92.63 ± 0.44	92.95 ± 0.17	95.07 ± 0.17
BGRL∗ (1k epochs)	73.24 ± 0.62	87.37 ± 0.40	91.77 ± 0.57	92.07 ± 0.06	OOM∗
GCA	78.35 ± 0.05	88.94 ± 0.15	92.53 ± 0.16	93.10 ± 0.01	95.73 ± 0.03
Supervised GCN	77.19 ± 0.12	86.51 ± 0.54	92.42 ± 0.22	93.03 ± 0.31	95.65 ± 0.16

4.2. Evaluation protocol

Self-supervised framework training. We start the evaluation procedure by training the encoder networks using our proposed **Graph Barlow Twins** framework. In all scenarios, we use the AdamW optimizer [28] with weight decay equal to 10^{-5}. The learning rate is updated using a cosine annealing strategy with a linear warmup period. Our framework uses a single set of augmentation parameters for both graph views. Therefore we do not use reported values of these parameters from other publications that use two different sets. Instead we perform a grid search over the range: $p_A, p_X : \{0, 0.1, \ldots, 0.5\}$ for 500 epochs with a warmup time of 50 epochs. We implement our experiment using the PyTorch Geometric [29] library. All datasets are available in this library as well. The details about the used augmentation hyperparameters, node embedding dimensions and the encoder architecture are given in Appendices A and C.

Node embedding evaluation. We follow the linear evaluation protocol proposed by [7]. We use the trained encoder network, freeze the weights and extract the node embeddings for the original graph data without any augmentations. Next, we train a L_2-regularized logistic regression classifier from the Scikit learn [30] library. We also perform a grid search over the regularization strength using following values: $\{2^{-10}, 2^{-9}, \ldots, 2^{9}, 2^{10}\}$. In the case of the larger ogb-arxiv, ogb-products and the PPI datasets, the Scikit implementation takes too long to converge. Hence, we implement the logistic regression classifier in PyTorch and optimize it for 1000 steps using the AdamW optimizer. We check various weight decay values using a smaller grid search: $\{2^{-10}, 2^{-8}, \ldots, 2^{8}, 2^{10}\}$. We use these classifiers to compute the classification accuracy and report mean and standard deviations over 20 model initializations and splits, except for the ogb-arxiv, ogb-products and PPI datasets, where we there is only one data split provided — we only re-initialize the model weights 20 times (5 times for ogb-products due to long training time).

4.3. Transductive experiments

We evaluate and compare our framework to other graph representation learning approaches on 6 real-world datasets using the transductive setting. The whole graph including all the node features is observed during the encoder training. The node labels (classes) are hidden at that moment (unsupervised learning). Next, we use the frozen embeddings and labels of training nodes to train the logistic regression classifier.

4.3.1. Small and medium sizes benchmark datasets

Our first experiment uses 5 small and medium sized popular benchmark datasets, namely: WikiCS, Amazon Computers, Amazon Photos, Coauthor CS and Coauthor Physics.

Encoder model. Similarly to [15], we build our encoder f_θ as a 2-layer GCN [1] network. After the first GCN layer we apply a batch normalization layer (with momentum equal to 0.01) and the PReLU activation function. Accordingly to the original Barlow Twins method, we do not apply any normalization or activation to the final layer. A graph convolution layer (GCN) uses the diagonal degree matrix $\mathbf{D}$ to apply a symmetrical normalization to the adjacency matrix with added self-loops $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. Hence the propagation rule of such layer is defined as follows:

$$\mathrm{GCN}(X, A) = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}X\mathbf{W} \tag{4}$$

Note that we do not include the activation $\sigma(\cdot)$ in this definition, as we first apply the batch normalization and then the activation function.

Results and discussion. We train our framework for a total of 1000 epochs, but we observe that our model converges earlier at about 500–900 epochs (depending on the dataset; see Appendix B). This is significantly faster than the state-of-the-art method BGRL, which converges and reports results for 10 000 epochs. Additionally, we reproduce the results of BGRL and provide values for BGRL at 1000 epochs. In Table 2 we report the mean node classification accuracy along with the standard deviations. As our experimental scenario was aligned with BGRL one, we re-use their reported scores and compare them to our results. We observe that our proposed method outperforms other baselines and achieves comparable results to state-of-the-art methods. Moreover, our G-BT model outperforms BGRL at 1000 epochs.

4.3.2. ogb-arxiv dataset

In the next experiment, we use ogb-arxiv — a larger graph from the Open Graph Benchmark [22] with about 170 thousand nodes and about 1.1 million edges.

Encoder model. Due to the larger size of the graph, we extend the encoder f_θ to a 3-layer GCN model. We employ batch normalization and PReLU activations after the first and second layer, leaving the final layer as is (i.e. without any activation of normalization). In the BGRL paper, the authors suggested to use layer normalization together with weight standardization [31], yet we did not observe any performance gain, but more importantly the training procedure was unstable, with many peaks in the loss function.

Results and discussion. In Table 3 we report the mean classification accuracy along with the standard deviations. Note that we provide values for both validation and test splits, as the provided data splits are build according to chronological order. Hence, any model will be more affected with the out-of-distribution error on further (in time) away data samples. We evaluate our model for 500 epochs but it converges as fast as about 300–400 epochs (further training did not give any improvements). The model

Table 3
Mean and std accuracy of transductive node classification the **ogb-arxiv** dataset over 20 data splits and initializations obtained in BGRL paper [15] and our experiment (∗) within the same experimental setup.

	Validation	Test
MLP	57.65 ± 0.12	55.50 ± 0.23
node2vec	71.29 ± 0.13	70.07 ± 0.13
DGI	71.26 ± 0.11	70.34 ± 0.16
GRACE (10k epochs)	72.61 ± 0.15	71.51 ± 0.11
BGRL (10k epochs)	72.53 ± 0.09	71.64 ± 0.12
G-BT∗ (300 epochs)	71.16 ± 0.14	70.12 ± 0.18
Supervised GCN	73.00 ± 0.17	71.74 ± 0.29

Table 4
Mean and std Micro-F1 of multilabel node classification the **PPI dataset** over 20 model initializations obtained in BGRL paper [15] and our experiment (∗) within the same experimental setup.

	PPI (test set)
Raw features	42.20
DGI	63.80 ± 0.20
GMI	65.00 ± 0.02
GRACE	66.20 ± 0.10
GRACE GAT encoder (1k epochs)	69.71 ± 0.17
BGRL GAT encoder (1k epochs)	70.49 ± 0.05
G-BT∗ (500 epochs)	70.49 ± 0.19
Supervised GAT	97.30 ± 0.20

achieves results which are only 1.5 pp off to the state-of-the-art method BGRL, which in turn takes 10 000 epochs to converge to such solution.

4.4. Inductive experiments

We evaluate our proposed **G-BT** framework in inductive tasks over a single and multiple graphs.

4.4.1. PPI

For the inductive learning case with multiple graphs, we employ the **Protein–Protein Interaction (PPI)** dataset [27]. Aligned with other methods, we provide results for multilabel node classification in terms of the Micro-F1 score.

Encoder model. We employ a Graph Attention (GAT) [2] based encoder model, as previous works have shown better results of such network compared to standard GCN layers on PPI. Specifically, we build our encoder f_θ as a 3-layer GAT network with skip connections. The first and second layer uses 4 attention heads of size 256 which are concatenated, and the final layer uses 6 attention heads of size 512, whose outputs are averaged instead of applying concatenation. In the GAT model, an attention mechanism learns the weights that are used to aggregate information from neighboring nodes. The attention weights α_{ij} are computed according to the following equation:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}h_i \parallel \mathbf{W}h_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}h_i \parallel \mathbf{W}h_k\right]\right)\right)} \tag{5}$$

where $\mathcal{N}_i$ are the neighbors of node i, $\mathbf{W}$ is a trainable matrix to transform node attributes, $\mathbf{a}$ is the trainable attention matrix, and $\parallel$ denotes the concatenation operation.

We use the ELU activation for the first and second layer, leaving the last layer without any activation function. We do not apply any normalization techniques in the model as preliminary experiments showed no performance improvement.

Results and discussion. We train our framework using a batch size of 1 graph for a total of 500 epochs, which turned out to be enough for the model to converge (we conducted some preliminary experiments). In Table 4, we report the mean Micro-F1 score along with the standard deviations over 20 model initialization, as this dataset provided only one data split. Training for only 500 epochs provided results on par with SOTA method – BGRL – our model achieves 70.49 using a GAT encoder.

4.4.2. ogb-products

We study the applicability of our proposed model in the case of large-scale graphs. We select the ogb-products dataset, which has about 2.5 million nodes and 61 million edges.

Table 5
Mean and std accuracy of inductive node classification on the **ogb-products** dataset over 5 model initializations obtained in the OGB leaderboard and our experiment (∗) within the same experimental setup.

	Validation	Test
Features∗	63.18 ± 0.01	50.93 ± 0.01
DeepWalk∗	87.42 ± 0.09	73.11 ± 0.44
DeepWalk + fts∗	87.84 ± 0.09	73.38 ± 0.11
BGRL∗ (100 epochs)	78.06 ± 2.12	63.97 ± 1.62
G-BT∗ (100 epochs)	85.04 ± 0.23	70.46 ± 0.38
Supervised GCN	92.00 ± 0.03	75.64 ± 0.21

Encoder model and setup. We utilize the same GAT-based encoder as for PPI. Due to the size of this dataset and the resulting training time, we decide to perform inductive node classification, i.e., during training we use only the nodes from the training set and edges among them. Moreover, as this graph does not fit into GPU memory, we selected a batched setting with neighbor sampling (as proposed in [3]) instead of the full-batch scenario. We train our model with a batch size of 512 for 100 epochs.

Results and discussion. BGRL does not report results for this dataset, so we modify the implementation of the BGRL method to accept batches instead of whole graphs and evaluate it on this dataset. We also include results from the OGB leaderboard, but note that virtually all methods reported there are trained in a semi-supervised setting, contrary to our approach in the self-supervised setting. Therefore, we may expect worse results. We summarize the mean and std node classification accuracy values in Table 5. We observe that G-BT highly outperforms BGRL on both validation and test sets.

4.5. Training time comparison

We compare the training time of all considered models by the duration of single epoch (the evaluation phase is the same in all models). We run each model for 10 training epochs and report the mean and standard deviation of the time measurements (Table 6). In virtually all cases our model takes the least time for a single training iteration due to the simple symmetrical architecture. Compared to BGRL our method speeds up computations about 17–42 times.

4.6. Batched processing

Our proposed method allows working in both full-batch and mini-batch settings. For most considered datasets, splitting them up into batches is not required as these fit completely into the GPU's memory. Nevertheless, we run additional experiments where we train our G-BT model on these datasets in a batched manner. Batches are created using neighbor sampling, i.e. for a

Table 6
Single epoch running time (in seconds) averaged over 10 training epochs.

	WikiCS	Am-CS	Am-Photo	Co-CS	Co-Phy
DeepWalk	0.83 ± 0.02	0.96 ± 0.02	0.62 ± 0.02	1.22 ± 0.03	2.25 ± 0.03
DGI	0.06 ± 0.03	0.09 ± 0.00	0.05 ± 0.00	0.19 ± 0.00	0.50 ± 0.21
MVGRL	OOM	OOM	OOM	OOM	OOM
GRACE	0.33 ± 0.10	0.43 ± 0.03	0.15 ± 0.01	OOM	OOM
BGRL	0.12 ± 0.01	0.18 ± 0.01	0.08 ± 0.00	0.35 ± 0.01	OOM
Epochs | training time [s]:	10 000 | 1200	10 000 | 1800	10 000 | 800	10 000 | 3500	10 000 | –
G-BT	0.05 ± 0.00	0.07 ± 0.00	0.04 ± 0.00	0.22 ± 0.05	0.44 ± 0.01
Epochs | training time [s]:	900 | 45	600 | 42	500 | 20	900 | 198	900 | 396
Speedup (vs. BGRL):	**26x**	**42x**	**40x**	**17x**	–

Table 7
Evaluation of G-BT model in batched setting.

Batch size	WikiCS	Am-CS	Am-Photo	Co-CS	Co-Phy
Full-batch	76.65 ± 0.62	88.14 ± 0.33	92.63 ± 0.44	92.95 ± 0.17	95.07 ± 0.17
256	75.69 ± 1.02	87.93 ± 0.39	91.24 ± 0.46	91.82 ± 0.22	94.98 ± 0.14
512	75.83 ± 0.64	88.21 ± 0.44	91.21 ± 0.44	91.62 ± 0.22	94.95 ± 0.12
1024	75.79 ± 0.77	87.94 ± 0.50	91.24 ± 0.47	91.54 ± 0.31	94.91 ± 0.12
2048	75.58 ± 0.52	87.92 ± 0.29	91.21 ± 0.40	91.43 ± 0.28	94.84 ± 0.11

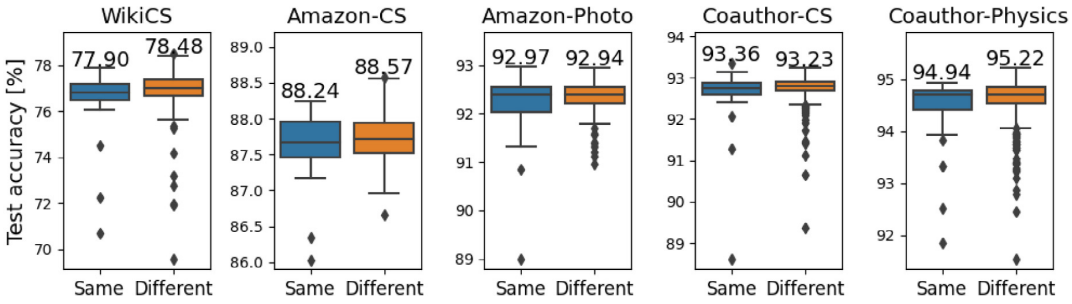


Fig. 2. Comparison of using the "Same" and "Different" augmentation hyperparameter sets.

k-layer encoder model, we sample the k-hop neighborhood of a node. More specifically, we first subsample the direct neighbors, then we sample neighbors of those, etc (as proposed in [3]). We re-use the augmentation hyperparameter values and number of epochs found in the full-batch case and retrain the G-BT model for each batch size 5 times (Table 7). We observe an expected decrease in performance when using the batched scenario (subject to further finetuning).

5. Ablation and hyperparameter sensitivity study

To gain a better understanding of our proposed method, we conduct an ablation and hyperparameter sensitivity study. In particular, we focus on the augmentation functions (types and hyperparameters) and the encoder architectures.

5.1. Augmentation hyperparameter sets

In our model, we postulate to use the same augmentation function hyperparameters to generate both graph views. This is motivated by the symmetrical architecture of our model, and hyperparameter search complexity. Performing a simple grid search over the value space yields in our case a total number of $6^2 = 36$ evaluated combinations (values: $\{0, 0.1, \ldots, 0.5\}$). In contrary, usage of different parameter sets for both graph views, would generate $(6^2)^2 = 1296$ combinations, which can be further reduced by exploiting the symmetrical architecture, yielding a final value of $630 = \binom{36}{2}$ combinations to evaluate. To demonstrate the impact of using both the same and different augmentation hyperparameter value sets we provide the results in Fig. 2. There is no substantial difference in terms of test accuracy.

5.2. Ablation study

For the ablation study, we examine the utilized augmentation functions and the encoder architectures. We considered four literature-based configurations of augmentation functions [15] and four encoders as following:

- for the augmentation functions:
  - without any augmentation functions
  - using only node feature masking
  - using only edge dropping
  - using both augmentation functions (node feature masking and edge dropping) as in the original version of our method
- for the encoder architectures:
  - MLP-based encoder (in contrast to a graph neural network) — a three-layer model with batch normalization and PReLU activations after the first two layers, whereas the output of the third linear layer is unmodified (aligned with other models in our paper);
  - one-layer GCN encoder — a single GCN layer without any normalization or activation functions;
  - two-layer GCN encoder — as in the original version of our method;
  - three-layer GCN encoder — the same architecture as used for the ogb-arxiv dataset.

We conducted all the ablation study experiments with the WikiCS dataset. We trained each model version for 1000 epochs (with

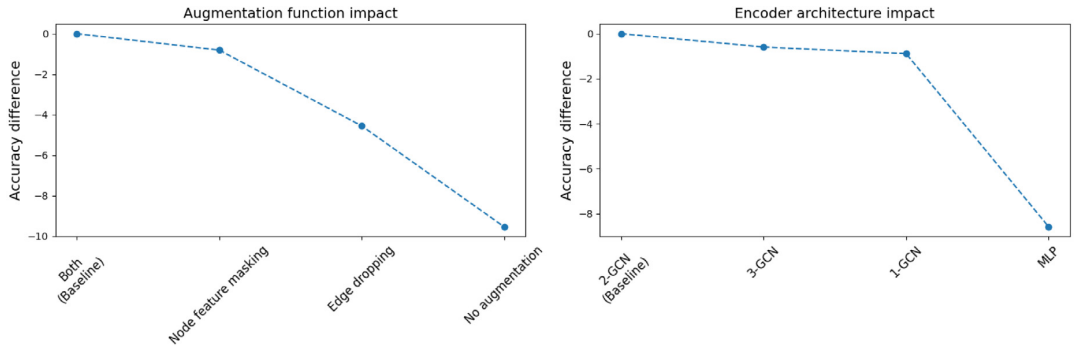


Fig. 3. Visualization of the ablation study. We consider the two most important parts of our model: the augmentation functions and the encoder architecture. We evaluate different settings and conclude that: (1) using both augmentation methods (node feature masking and edge dropping) at once, yields the best results, (2) the two-layer GCN model works the best, whereas a deeper one (three layers) suffers from the oversmoothing issue.

100 warmup epochs). All the remaining hyperparameters were borrowed from the best performing G-BT model on the WikiCS dataset, as reported in our paper.

We visualize the influence of both the augmentation functions and encoder architectures in Fig. 3. Let us notice that using both augmentation functions provides the best results. However, using only node feature masking already leads to decent results (1pp difference; 75.9% acc). We conclude that node feature masking is more expressive than edge dropping, as using only edge dropping provides a smaller results boost (72% accuracy). Without any augmentation functions, we observe a noticeably lower quality (67% accuracy). In fact, no augmentation with large enough number of training epochs should result in very poor quality (the representation collapses into a constant embedding).

For the encoder architectures, we notice that using the two-layer GCN model (as evaluated in our main experimental pipeline) leads to the best results. One might expect a larger model (three-layer GCN) to work better, yet we observe a performance drop when using such an encoder. This may be related to the over-smoothing issue in Graph Neural Networks. The one-layer GCN is also a reasonable choice in comparison to the two-layer model as its accuracy is only 1.1pp worse than the baseline. Moreover, the time and space complexity of deeper GNN models (two, three, ...) tend to explode in the case of highly dense graphs. Ignoring the graph structure by using an MLP-based encoder leads to results of about 68% accuracy.

5.3. Loss function trade-off parameter

The λ parameter in the loss function (see: Eq. (3)) controls the trade-off between the invariance and redundancy reduction term. We evaluate multiple choices for this parameter and report the results in Fig. 4. We use a similar setup as in the ablation study (see: Section 5.2) – i.e., we utilize the WikiCS dataset and train a two-layer GCN encoder for 1000 epochs with 100 warmup epochs, whereas other hyperparameters (except the λ parameter) are borrowed from the best performing G-BT model on the WikiCS dataset. We evaluate the following values for $\lambda = \{0, 0.001, \frac{1}{256}, 0.01, 0.1, 0.5, 1.0, 2.0\}$. Note that the value $\lambda = \frac{1}{d} = \frac{1}{256}$ corresponds to the default choice used throughout all our other experiments. Using this value was initially suggested in [21]. Let us notice that the results in Fig. 4 show that our proposed G-BT model achieves the best performance for exactly such λ. Moreover, one can conclude that smaller values yield higher performance than larger ones. In particular, when using $\lambda = 1$ (both invariance and redundancy reduction terms are equally important) the performance deteriorates and settles at about 55% accuracy, whereas using values smaller than 0.01 results in a much better embeddings — about 75% accuracy.

5.4. Impact of projector network

In our proposed G-BT method, we omit the so-called projector network, which was utilized in the original Barlow Twins method. Its main purpose is to reduce high embedding dimensionality (of the ResNet encoder), whereas in our approach that step is solved by utilizing GNNs with low dimensional embeddings. We evaluate multiple choices of the projector network dimensionality (including our base case with no projector network at all) and report the results in Fig. 5. We use the same setup as previously, namely in the ablation study (see: Section 5.2) and loss function trade-off parameter study (see: Section 5.3) — i.e., we utilize the WikiCS dataset and train a two-layer GCN encoder for 1000 epochs with 100 warmup epochs, whereas other hyperparameters are borrowed from the best performing G-BT model on the WikiCS dataset. We evaluate the following values for the projector dimensionality $\{128, 256, \ldots, 8192, 16348\}$. We did not observe any significant differences in the model performance, regardless of whether we employ a projector network or we omit it completely. We performed a Friedman test and the computed p-value confirmed our observation.

6. Conclusions

In this work we presented Graph Barlow Twins, a self-supervised graph representation learning framework, which utilizes the embeddings' cross-correlation matrix computed from two distorted views of a particular graph. The framework is fully symmetric and does not need any special techniques to build non trivial embedding vectors. It builds representations that are invariant to the applied augmentations and reduces the redundancy in the representation vectors by enforcing the cross-correlation matrix to be equal to the identity matrix (Barlow Twins loss). Using 8 real-world datasets we evaluate our model in node classification tasks, both transductive and inductive, and achieve results that are on par or better than SOTA methods in SSL graph representation learning. We also show that our model converges an order of magnitude faster than other approaches.

Overall, our method allows to reduce the computation cost (faster convergence) while keeping a decent performance in downstream tasks. Consequently, it can be used to process larger graph datasets and efficiently perform tasks such as node classification, link prediction or graph classification. These tasks have crucial impact on various machine learning areas where graph structured data is used, e.g. detection of bots or hate speech in social networks, or building graph based recommendation engines.

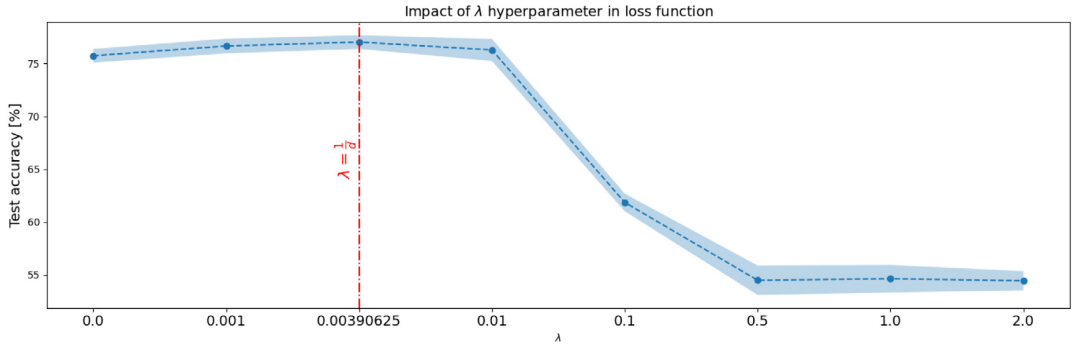


Fig. 4. Evaluation of the impact of the loss function's λ parameter on the overall model performance. The more the redundancy reduction term is present, the lower the accuracy is. Empirically the best performing contribution is for $\lambda = \frac{1}{d}$, where d is the dimensionality of the embedding vectors.

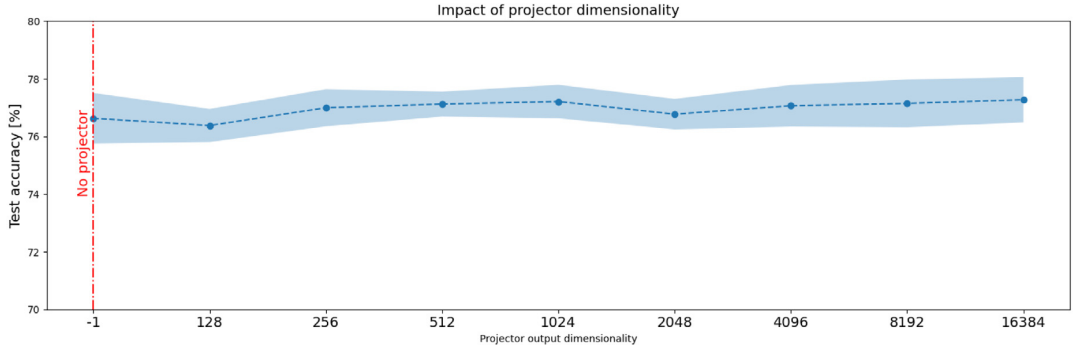


Fig. 5. Evaluation of the impact of the projector output dimensionality on the overall model performance. We performed a Friedman test and confirmed that the results are not significantly different.

Further studies can address the utilization of other negative-sample free approaches and applications of the proposed model in further graph-related tasks, such as link prediction or graph classification, and extensions to other types of data that are more specific than graphs (e.g., texts, tabular data).

CRediT authorship contribution statement

Piotr Bielak: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Tomasz Kajdanowicz:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Supervision, Funding acquisition. **Nitesh V. Chawla:** Conceptualization, Formal analysis, Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The project was partially supported by the National Science Centre, Poland, grants numbers: 2021/41/N/ST6/03694, 2016/21/D/ST6/02948, as well as statutory funds from the Department of Artificial Intelligence.

Appendix A. Augmentation hyperparameters

Our proposed framework uses a single pair of augmentation hyperparameters $p_A \in \mathbb{R}, p_X \in \mathbb{R}$ compared to other methods that use different values to generate both graph views. We show that a single set is enough to achieve a decent performance in a symmetrical network architecture like ours. Therefore, we cannot use the reported values of other works. We instead perform a grid search over these hyperparameters and use those where the model performs the best (in terms of classification accuracy or Micro-F1 score, for PPI). We do not evaluate the model during training and just use the final version after training. We use the following setting:

- the framework is trained for **500 epochs**,
- we set the learning rate warmup time to **50 epochs**,
- for both hyperparameters p_A and p_X we check following values: $\{0, 0.1, \ldots, 0.5\}$.

For values greater than 0.5 the augmentation removes too much information from the graph. In the case of the ogb-products dataset, due to its large size, we trained our model only for 10 epochs with a warmup period of 2 epochs, but we evaluated the same augmentation hyperparameter values. We summarize the augmentation hyperparameters of the best performing models in Table A.8.

Appendix B. Training setup

For all datasets, we train our framework using the AdamW [28] optimizer with a weight decay of 10^{-5}. The learning rate is adjusted using a cosine annealing strategy with a linear warmup

Table A.8
Augmentation hyperparameters. Ogb-products was trained in the batched setting with a batch size of 512.

	G-BT	
	p_A	p_X
WikiCS	0.2	0.1
Amazon-CS	0.4	0.1
Amazon-Photo	0.0	0.5
Coauthor-CS	0.5	0.1
Coauthor-Physics	0.1	0.4
ogb-arxiv	0.2	0.0
PPI	0.1	0.1
ogb-products∗	0.2	0.1

period up to the base learning rate. During training we set a total number of epochs and an evaluation interval, after which the frozen embeddings are evaluated in downstream tasks (using either the l_2 regularized logistic regression from Scikit learn [30] with liblinear solver, or the custom PyTorch version with AdamW for ogb-arxiv and PPI). For instance, if we set the total number of epochs to 1000 and the evaluation interval to 500, the model will be evaluated at epochs: 0, 500 and 1000 (three times in total). We report the values for the best performing model found during those evaluations. We summarize these training statistics in Table B.9.

Appendix C. Encoder architecture

We compare our framework against the state-of-the-art self-supervised graph representation learning method BGRL [15]. To provide a fair comparison, we use similar encoder architectures to the ones presented in their paper. We do not use any predictor networks in our framework, so we need to slightly modify the encoders to be better suited for the loss function (as given in the Barlow Twins paper [14]), i.e. we do not apply any normalization (like batch or layer normalization) or activation function in the final layers of the encoder. Note that the lack of predictor network and batch normalization in the final layer, reduces the overall number of trainable network parameters. In all cases, we use a batch normalization with the momentum equal to 0.01 (as in BGRL [15], where they use the equivalent weight decay equal to 0.99).

For the small up to medium sized datasets, i.e. WikiCS, Amazon-CS, Amazon-Photo, Coauthor-CS, Coauthor-Physics, we use a 2-layer GCN [1] based encoder with the following architecture:

- GCN(k, $2d$),
- BatchNorm($2d$),
- PReLU(),
- GCN($2d$, d),

where k is the number of node features and d is the embedding vector size.

For the ogb-arxiv dataset, we use a slightly larger model — a 3-layer GCN [1] based encoder. We tried to utilize weight standardization [31] and layer normalization, but our model did not benefit from those techniques (as it helped in BGRL [15]). The training procedure under this setting was unstable with various fluctuations and peaking of the loss function. The final architecture is summarized as follows:

- GCN(k, d),
- BatchNorm(d),
- PReLU(),
- GCN(d, d),
- BatchNorm(d),
- PReLU(),
- GCN(d, d).

In the inductive experiment with the PPI dataset, we use a 3-layer GAT [2] based encoder. Graph Attention network are known to perform better on this dataset compared to GCNs. This was also showed in BGRL [15], where their approach with GAT layers provided state-of-the-art performance in self-supervised graph representation learning for PPI. Our architecture can be summarized as follows:

- GAT(k, 256, heads $= 4$) + Linear(k, $4 * 256$)
- ELU(),
- GAT($4 * 256$, 256, heads $= 4$) + Linear($4 * 256$, $4 * 256$)
- ELU(),
- GAT($4 * 256$, d, heads $= 6$) + Linear($4 * 256$, d)

The outputs of the attention heads in the first and second layer are concatenated and for the last GAT layer, the attention heads outputs are averaged. In every layer, we utilize skip connections using linear layers to project the outputs of the previous layer (features in the case of the first layer) to the desired dimensionality.

The exact values for the input feature dimension k and the embedding dimension d are given in Table C.10.

Appendix D. Code and reproducibility

We implement all our models using the PyTorch-Geometric library [29]. The experimental pipeline is built using the Data Version Control tool (DVC [23]). It enables to run all experiments in a single command and ensure better reproducibility. We attach the code available at https://github.com/pbielak/graph-barlow-twins. To reproduce the whole pipeline run: `dvc repro` and to execute a single stage use: `dvc repro <stage name>`. There are following stages:

- `preprocess_dataset@<dataset_name>` — downloads the `<dataset_name>` dataset; if applicable, generates the node splits for train/val/test,
- `full_batch_hps@<dataset_name>` — runs the augmentation hyperparameter search for a given dataset in the full-batch case,
- `full_batch_train@<dataset_name>`, `batched_train@<dataset_name>` — trains and evaluates the G-BT model for a given dataset in the full-batch case and the batched scenario, respectively,
- `batched_hps_ogbn_products` — runs the augmentation hyperparameter search for the ogb-products dataset in the batched scenario,
- `batched_train_ogbn_products` — trains and evaluated the G-BT model for the ogb-products dataset in the batched scenario,
- `compare_augmentation_hyperparameter_sets` — loads all full-batch augmentation hyperparameter results, compares the case when using the same or different sets of hyperparameters to generate both graph views (outputs Fig. 2),
- `compare_running_times` — computes the average running time of a training epoch for the following methods: DeepWalk, DGI, MVGRL, GRACE, BGRL and G-BT,
- `train_bgrl_full_batch@<dataset_name>` — trains and evaluates the BGRL model in the full-batch case for WikiCS, Amazon-CS, Amazon-Photo, and Coauthor-CS,
- `bgrl_hps_batched@ogbn-products` — runs the augmentation hyperparameter search for BGRL using the ogb-products dataset,

Table B.9
Training hyperparameters.

	G-BT				
	Total epochs	Warmup	Evaluation interval	Base learning rate	Best model found at
WikiCS	1000	100	100	$5 * 10^{-4}$	900
Amazon-CS	1000	100	100	$5 * 10^{-4}$	600
Amazon-Photo	1000	100	100	$1 * 10^{-4}$	500
Coauthor-CS	1000	100	100	$1 * 10^{-5}$	900
Coauthor-Physics	1000	100	100	$1 * 10^{-5}$	900
ogb-arxiv	500	100	100	$1 * 10^{-3}$	300
PPI	500	50	100	$5 * 10^{-3}$	500
ogb-products	100	10	10	$1 * 10^{-3}$	100

Table C.10
Encoder layer size parameters.

	G-BT	
	Number of node features k	Embedding dimensionality d
WikiCS	300	256
Amazon-CS	767	128
Amazon-Photo	745	256
Coauthor-CS	6805	256
Coauthor-Physics	8415	128
ogb-arxiv	128	256
PPI	50	512
ogb-products	100	128

- `bgrl_batched_train@ogbn-products` — trains and evaluates the BGRL model for the ogb-products dataset,
- `evaluate_features_products` — evaluates the performance of ogb-products' raw node features,
- `evaluate_deepwalk_products` — evaluates the performance of DeepWalk on the ogb-products dataset; additionally the case of DeepWalk features concatenated with raw node features is also evaluated.

All hyperparameters described in this Appendix are stored in configuration files in the `experiments/configs/` directory, whereas the experimental Python scripts are placed in the `experiments/scripts/` directory.

References

[1] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: International Conference on Learning Representations (ICLR), 2017.
[2] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, in: International Conference on Learning Representations, 2018, https://openreview.net/forum?id=rJXMpikCZ, (in press) as poster.
[3] W.L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, in: NIPS, 2017.
[4] B. Perozzi, R. Al-Rfou, S. Skiena, DeepWalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, in: KDD '14, ACM, New York, NY, USA, 2014, pp. 701–710, http://dx.doi.org/10.1145/2623330.2623732, http://doi.acm.org/10.1145/2623330.2623732.
[5] A. Grover, J. Leskovec, Node2vec: Scalable feature learning for networks, in: KDD, 2016.
[6] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, LINE: Large-scale information network embedding, in: WWW, ACM, 2015.
[7] P. Veličković, W. Fedus, W.L. Hamilton, P. Liò, Y. Bengio, R.D. Hjelm, Deep graph infomax, in: International Conference on Learning Representations, 2019, https://openreview.net/forum?id=rklz9iAcKQ.
[8] T.N. Kipf, M. Welling, Variational graph auto-encoders, in: NIPS Workshop on Bayesian Deep Learning, 2016.
[9] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, L. Wang, Graph contrastive learning with adaptive augmentation, 2020, CoRR abs/2010.14945 https://arxiv.org/abs/2010.14945.
[10] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, Y. Shen, Graph contrastive learning with augmentations, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 5812–5823, https://proceedings.neurips.cc/paper/2020/file/3fe230348e9a12c13120749e3f9fa4cd-Paper.pdf.
[11] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, L. Wang, Deep graph contrastive representation learning, in: ICML Workshop on Graph Representation Learning and beyond, 2020, http://arxiv.org/abs/2006.04131.
[12] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, M. Valko, Bootstrap your own latent - A new approach to self-supervised learning, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 21271–21284, https://proceedings.neurips.cc/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf.
[13] X. Chen, K. He, Exploring simple siamese representation learning, 2020.
[14] J. Zbontar, L. Jing, I. Misra, Y. LeCun, S. Deny, Barlow twins: Self-supervised learning via redundancy reduction, 2021, arXiv:2103.03230.
[15] S. Thakoor, C. Tallec, M.G. Azar, R. Munos, P. Veličković, M. Valko, Bootstrapped representation learning on graphs, in: ICLR 2021 Workshop on Geometrical and Topological Representation Learning, 2021, https://openreview.net/forum?id=QrzVRAA49Ud.
[16] J. Schmidhuber, Making the world differentiable: On Using Fully Recurrent Self-Supervised Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments, Tech. Rep. FKI-126-90 (revised), Institut für Informatik, Technische Universität München, 1990.
[17] F. Che, G. Yang, D. Zhang, J. Tao, P. Shao, T. Liu, Self-supervised graph representation learning via bootstrapping, 2020, arXiv preprint arXiv:2011.05126.
[18] G. Deco, L. Parra, Non-linear feature extraction by redundancy reduction in an unsupervised stochastic neural network, Neural Netw. 10 (4) (1997) 683–691, http://dx.doi.org/10.1016/S0893-6080(96)00110-4.
[19] J. Schmidhuber, M. Eldracher, B. Foltin, Semilinear predictability minimization produces well-known feature detectors, Neural Comput. 8 (4) (1996) 773–786, http://dx.doi.org/10.1162/neco.1996.8.4.773.
[20] J. Ballé, V. Laparra, E.P. Simoncelli, End-to-end optimized image compression, in: Int'L Conf on Learning Representations (ICLR), Toulon, France, 2017, Available at http://arxiv.org/abs/1611.01704.
[21] Y.-H.H. Tsai, S. Bai, L.-P. Morency, R. Salakhutdinov, A note on connecting barlow twins with negative-sample-free contrastive learning, 2021, arXiv:2104.13712.
[22] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec, Open graph benchmark: Datasets for machine learning on graphs, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 22118–22133, https://proceedings.neurips.cc/paper/2020/file/fb60d411a5c5b72b2e7d3527cfc84fd0-Paper.pdf.
[23] R. Kuprieiev, D. Petrov, S. Pachhai, P. Redzyński, C. da Costa-Luis, A. Schepanovski, P. Rowlands, I. Shcheklein, J. Orpinel, F. Santos, B. Taskaya, A. Sharma, Zhanibek, Gao, D. Hodovic, A. Grigorev, Earl, nik123, N. Dash, G. Vyshnya, maykulkarni, M. Hora, Vera, S. Mangal, W. Baranowski, C. Wolff, A. Maslakov, A. Khamutov, K. Benoy, DVC: Data version control - git for data & models, 2021, http://dx.doi.org/10.5281/zenodo.4733984.
[24] P. Mernyei, C. Cangea, Wiki-CS: A wikipedia-based benchmark for graph neural networks, 2020, arXiv preprint arXiv:2007.02901.
[25] J. McAuley, C. Targett, Q. Shi, A. van den Hengel, Image-based recommendations on styles and substitutes, in: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information

Retrieval, in: SIGIR '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 43–52, http://dx.doi.org/10.1145/2766462.2767755.

[26] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J.P. Hsu, K. Wang, An overview of microsoft academic service (MAS) and applications, in: Proceedings of the 24th International Conference on World Wide Web, in: WWW '15 Companion, Association for Computing Machinery, New York, NY, USA, 2015, pp. 243–246, http://dx.doi.org/10.1145/2740908.2742839.

[27] M. Zitnik, J. Leskovec, Predicting multicellular function through multi-layer tissue networks, Bioinformatics 33 (14) (2017) i190–i198, http://dx.doi.org/10.1093/bioinformatics/btx252.

[28] S. Gugger, J. Howard, Adamw and super-convergence is now the fastest way to train neural nets., 2018, https://www.fast.ai/2018/07/02/adam-weight-decay/.

[29] M. Fey, J.E. Lenssen, Fast graph representation learning with PyTorch Geometric, in: ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[31] S. Qiao, H. Wang, C. Liu, W. Shen, A. Yuille, Weight standardization, 2019, arXiv preprint arXiv:1903.10520.

4.2 AttrE2vec: Unsupervised attributed edge representation learning

Authors:	Piotr Bielak, Tomasz Kajdanowicz, Nitesh V. Chawla
Published at:	Information Sciences (IF 8.233)
MEIN points:	200
Citations:	Web of Science: 2
(as of 05.06.2023)	Scopus: 2
	Google Scholar: 4
Credit:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization

Information Sciences 592 (2022) 82–96

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

AttrE2vec: Unsupervised attributed edge representation learning

Piotr Bielak [a,*], Tomasz Kajdanowicz [a,*], Nitesh V. Chawla [b]

[a] Department of Artificial Intelligence, Wroclaw University of Science and Technology, Wroclaw, Poland
[b] Department of Computer Science and Engineering and Lucy Family Institute for Data and Society, University of Notre Dame, Notre Dame, IN, United States

ARTICLE INFO

Article history:
Received 29 December 2020
Received in revised form 21 October 2021
Accepted 22 January 2022
Available online 29 January 2022

Keywords:
Representation learning
Graphs
Edge embedding
Random walk
Neural network
Attributed graph
Graph neural network

ABSTRACT

Representation learning has overcome the often arduous and manual featurization of networks through (unsupervised) feature learning as it results in embeddings that can apply to a variety of downstream learning tasks. The focus of representation learning on graphs has focused mainly on shallow (node-centric) or deep (graph-based) learning approaches. While there have been approaches that work on homogeneous and heterogeneous networks with multi-typed nodes and edges, there is a gap in learning edge representations. This paper proposes a novel unsupervised inductive method called `AttrE2Vec`, which learns a low-dimensional vector representation for edges in attributed networks. It systematically captures the topological proximity, attributes affinity, and feature similarity of edges. Contrary to current advances in edge embedding research, our proposal extends the body of methods providing representations for edges, capturing graph attributes in an inductive and unsupervised manner. Experimental results show that, compared to contemporary approaches, our method builds more powerful edge vector representations, reflected by higher quality measures (AUC, accuracy) in downstream tasks as edge classification and edge clustering. It is also confirmed by analyzing low-dimensional embedding projections.

1. Introduction

Complex networks, included attributed and heterogeneous networks, are ubiquitous—from recommender systems to citation networks and biological systems [1]. These networks present a multitude of machine learning problem statements, including node classification, link prediction, and community detection. A fundamental aspect of any such machine learning (ML) task, transductive or inductive, is the availability of featurized data. Traditionally, researchers have identified several network characteristics suited to specific ML tasks and used them for the learning algorithm. This practice is arduous as it often entails customizing to each specific ML task, and also is limited to the computable characteristics.

This has led to a surge in (unsupervised) algorithms and methods that learn embeddings from the networks, such that these embeddings form the featurized representation of the network for the ML tasks [2–6]. This area of research is generally notated as representation learning in networks. Generally, these embeddings generated by representation learning methods are agnostic to the end use-case, as they are generated in an unsupervised fashion. Traditionally, the focus was on representation learning on homogeneous networks, i.e. the networks that have singular type of nodes and edges, and also do not have attributes attached to the nodes and edges [4].

Existing representation learning models mainly focus on transductive learning, where a model can only be trained using the entire input graph. It means that the model requires all the nodes and a fixed structure of the network in the training

* Corresponding authors.
E-mail addresses: piotr.bielak@pwr.edu.pl (P. Bielak), tomasz.kajdanowicz@pwr.edu.pl (T. Kajdanowicz), nchawla@nd.edu (N.V. Chawla).

https://doi.org/10.1016/j.ins.2022.01.048
0020-0255/

phase, e.g., Node2vec [7], DeepWalk [8] and GCN [9], to some extent. Besides, there have been methods focused on heterogeneous networks that incorporate different typed nodes and edges in a network, as well as content at each node [10,11].

On the other hand, a less explored and exploited approach is the inductive setting. In this approach, only a part of the network is used to train the model to infer embeddings for new nodes. Several attempts have been made in the inductive setting including EP-B [12], GraphSAGE [13], GAT [14], SDNE [15], TADW [16], AHNG[17] or PVECB [18]. There is also recent progress on heterogeneous graph embedding, e.g., MIFHNE [19] or models based on graph neural networks [20].

State-of-the-art network embedding techniques are mostly unsupervised, i.e., aim at learning low-dimensional representations that preserve the structure of an input graph, e.g., GraphSAGE [13], DANE [21], line2vec [22], RCAN [23]. Nevertheless, semi-supervised or supervised methods can learn vector representations but for a specific downstream prediction task, e.g., TADW [16] or FSCNMF [24]. Hence it has been shown in the literature that not much supervision is required to learn the embeddings.

In recent years, proposed models mainly focus on the graphs that do not contain attributes related to nodes and edges [4]. It is especially noticeable for edge attributes. The majority of proposed approaches consider node attributes only, omitting the richness of edge feature space while learning the representation. Nevertheless, there have been successfully introduced such models as DANE [21], GraphSAGE [13], SDNE [15] or CAGE [25] which make use of node features and EGNN [26], NEWEE [27], EGAT [28] that consume edge attributes.

Both node-based embedding methods and graph neural network inspired methods do not generalize effectively to both transductive and inductive settings, especially when there are attributes associated with edges. This work is motivated by the idea of unsupervised learning on networks with attributed edges such that the embeddings are generalizable across tasks and are inductive.

To that end, we develop a novel `AttrE2vec`, an unsupervised learning model that adapts auto-encoder and self-attention network with the use of feature reconstruction and graph structural loss. To learn edge representation, `AttrE2vec` splits edge neighborhood into two parts, separately for each node endings of the edge, and then generates random edge walks in both neighborhoods. All walks are then aggregated over the node and edge attributes using one of the proposed strategies (Avg, Exp, GRU, ConcatGRU). These are accumulated with the original nodes and edge features and then fed to attention and dense layer to encode the edge. The embeddings are subsequently inferred via a two-step loss function—for both feature reconstruction and graph structural loss. As a consequence, `AttrE2vec` can explicitly incorporate feature information from nodes and edges at many hops away to effectively produce the plausible edge embeddings for the inductive setting.

In summary, our main contributions are as follows:

- we propose a novel unsupervised `AttrE2vec` method, which learns a low-dimensional vector representation for edges that are attributed
- we exploit the concept of a graph-topology-driven edge feature aggregation, from simple ones to learnable GRU based, that captures edge topological proximity and similarity of edge features
- the proposed method is inductive and allows getting the representation for edges not present in the training phase
- we conduct various experiments and show that our `AttrE2vec` method has superior performance over all of the baseline methods on edge classification and clustering tasks.

2. Related work and research gap

Embedding information networks has received significant interest from the research community. We refer the readers to the survey articles for a comprehensive overview of network embedding [4,5,3,2] and cite only some of the most prominent works that are relevant.

Unsupervised network embedding methods use only the network structure or original attributes of nodes and edges to construct embeddings. The most common method is DeepWalk [8], which in two-phases constructs node neighborhoods by performing fixed-length random walks and employs the skip-gram [7] model to preserve the co-occurrences between nodes and their neighbors. This two-phase framework was later an inspiration for learning network embeddings by proposing different strategies for constructing node neighborhoods or modeling co-occurrences between nodes, e.g., node2vec [7], Struc2-vec [35], GraphSAGE [13], line2vec [22] or NEWEE [27]. Another group of unsupervised methods utilizes auto-encoder or graph neural networks to obtain embedding. SDNE [15] uses auto-encoder architecture to preserve first and second-order proximities by jointly optimizing the loss in neighborhood reconstruction. Another auto-encoder based representatives are EP-B [12] and DANE [21].

Supervised network embedding methods are constructed as an end-to-end methods for particular tasks like node classification or link prediction. These methods require network structure, attributes of nodes and edges (if method is capable of using) and some annotated target like node class. The representatives are ECN [29], ECC [30], FSCNMF [24], GAT [14], planetoid [31], EGNN [26], GCN [9], EdgeConv [32], EGAT [28], Attribute2vec [33].

Edge representation learning has been already tackled by several methods, i.e. ECN [29], EGNN [26], line2vec [22], EdgeConv [32], EGAT [28]. However, non of these methods was able to directly take into account attributes of edges as well as perform the learning in an unsupervised manner.

All the characteristics of the representative node and edge representation learning methods are grouped in Table 1.

Table 1
Comparison of most representative graph embedding methods with their abilities to learn the representation, with or without attributes, reasoning types and short characteristics. The most prominent and appropriate methods selected to compare to `AttrE2vec` in experiments are marked with bold text.

	Method	Representation		Attributed		Reasoning		Family
		Nodes	Edges	Nodes	Edges	Transduct.	Induct.	
Supervised	ECN [29] (2016)		✓			✓		neigh. aggr.
	GCN [9] (2017)	✓		✓		✓	✓	GCN/GNN
	ECC [30] (2017)	✓		✓		✓		GCN, DL
	FSCNMF [24] (2018)	✓		✓		✓		GCN
	GAT [14] (2018)	✓		✓		✓	✓	AE, DL
	Planetoid [31] (2018)	✓		✓		✓	✓	GNN
	EGNN [26] (2019)	✓	✓	✓	✓	✓	✓	GNN
	EdgeConv [32] (2019)		✓			✓		GNN
	EGAT [28] (2019)	✓	✓	✓	✓	✓	✓	GNN
	Attribute2vec [33] (2020)	✓	✓			✓		GCN
Unsupervised	**DeepWalk** [8] (2014)	✓				✓		RW, skip-gram
	TADW [16] (2015)	✓	✓			✓		RW, MF
	LINE [34] (2015)	✓				✓		RW, skip-gram
	Node2vec [7] (2016)	✓				✓		RW, skip-gram
	SDNE [15] (2016)	✓		✓		✓	✓	AE
	GraphSAGE [13] (2017)	✓		✓		✓	✓	RW
	EP-B [12] (2017)	✓		✓		✓	✓	AE
	Struc2vec [35] (2017)	✓				✓		RW, skip-gram
	DANE [21] (2018)	✓		✓		✓	✓	AE
	Line2vec [22] (2019)		✓			✓		RW, skip-gram
	NEWEE [27] (2019)	✓		✓	✓	✓		RW, skip-gram
	AttrE2vec (2020)		✓	✓	✓	✓	✓	RW, AE, DL

3. Method

3.1. Motivation

In the following paragraphs, we explain our threefold motivation to propose the `AttrE2vec`.

Edge embeddings For a decade, network processing approaches gather more and more attention as graph data is produced in an increasing number of systems. Network embedding traditionally provided the notion of vectorizing nodes that was used in node classification or clustering. However, the edge representation learning did not gather enough attention and was accomplished through node embedding transformation [36]. Nevertheless, such an approach is problematic. For instance, inferring edge type from neighboring nodes' embeddings may not be the best choice for edge type classification in heterogeneous social networks. We claim that efficient edge clustering, edge attribute regression, or link prediction tasks require dedicated and specific edge representations. We expect that the representation learning approach devoted strictly to edges provides more powerful vector representations than traditional methods that require node embeddings trained upfront and transform nodes' embedding to represent edges.

Inductive embedding methods A vast majority of contemporary network representation learning methods is transductive (see Table 1). It means that any change to the graph requires the whole retraining of the method to provide predictions for unseen cases-such property limits the applicability of methods due to high computational costs. Contrary, the inductive approach builds a predictive ability that can be applied to unseen cases and does not need retraining – in general, inductive methods have a lower computation cost. Considering these advantages, we expect modern edge embedding methods to be inductive.

Encoding graph attributes in embeddings Much of the real-world data exhibits rich attribute sets or meta-data that contain crucial information, e.g., about the similarity of nodes or edges. Traditionally, graph representation learning has been focused on exploiting the network structure, omitting the related content. Thus, we may expect to consume attributes as a regularizer over the structure. It would allow overcoming the limitation when the only edge discriminating ability is encoded in the edges' attributes, not in the graph's structure. Relying only on the network would produce inconclusive embeddings.

3.2. Attributed graph edge embedding

We denote an attributed graph as $G = (V, E)$, where V is a set of nodes and $E = \{(u, v) \in V \times V\}$ a set of edges. Every node u and every edge $e = (u, v)$ has associated features: $m_u \in \mathbb{R}^{d_V}$ and $f_{uv} \in \mathbb{R}^{d_E}$, where $\mathcal{M} \in \mathbb{R}^{|V| \times d_V}$ and $\mathcal{F} \in \mathbb{R}^{|E| \times d_E}$ are node and edge feature matrices, respectively. By d_V we denote dimensionality of node feature space and d_E dimensionality of edge feature space. The edge embedding task is defined as learning a function $g : E \to \mathbb{R}^d$, which takes an edge and outputs its low-dimensional vector representation. Note that the embedding dimension d should be much less than the original edge feature

dimensionality d_E, i.e.: $d \ll d_E$. More specifically, we aim at using the topological structure of the graph and node and edge attributes: $f : (E, \mathcal{F}, \mathcal{M}) \rightarrow \mathbb{R}^d$.

3.3. AttrE2vec

In contrast to traditional node embedding methods, we shift the focus from nodes to edges and consider a graph from an edge perspective. Given any edge $e = (u, v)$, we can observe three natural sources of knowledge: the edge attributes itself and the two neighborhoods - N_u and N_v, located behind nodes u and v, respectively. In AttrE2vec, we exploit all three sources jointly.

First, we obtain aggregations (summaries) S_u, S_v of the both neighborhoods N_u, N_v. We want to capture the topological structure of the neighborhood, so we perform k **edge random walks** of length L, which start from node u (or v, respectively) and use a uniformly distributed neighbor sampling approach (DeepWalk-like) to obtain the next edge. Each ith walk w_u^i started from node u is hence a sequences of edges.

$$\mathbf{RW}(G, k, L, u) \rightarrow \{w_u^1, w_u^2, \ldots, w_u^k\}$$

$$w_u^i \equiv (u, u_2), (u_3, u_4), \ldots, (u_{L-1}, u_L)$$

Next, we take the attributes of the edges (and nodes, if applicable) in each random walk and aggregate them into a single vector using the **walk aggregation model $\mathbf{Agg_w}$**.

$$S_u^i = \mathbf{Agg_w}(w_u^i, \mathcal{F}, \mathcal{M})$$

Later, aggregated walks are combined using the **neighborhood aggregation model $\mathbf{Agg_n}$**, which summarizes the neighborhood S_u (and S_v, respectively). The proposed implementations of these aggregation are given in Section 3.4.

$$S_u = \mathbf{Agg_n}(\{S_u^1, S_u^2, \ldots, S_u^k\})$$

Finally, we obtain the low dimensional edge embedding h_{uv} using an **encoder Enc** module. It combines the edge attributes f_{uv} with the summarized neighborhood information S_u, S_v. We employ a simple Multilayer Perceptron (MLP) with 3 inputs (each of size equal to the edge features dimensionality) and an attention mechanism over these inputs, to check how much of the information of each input is used to create the embedding vector (see Fig. 3):

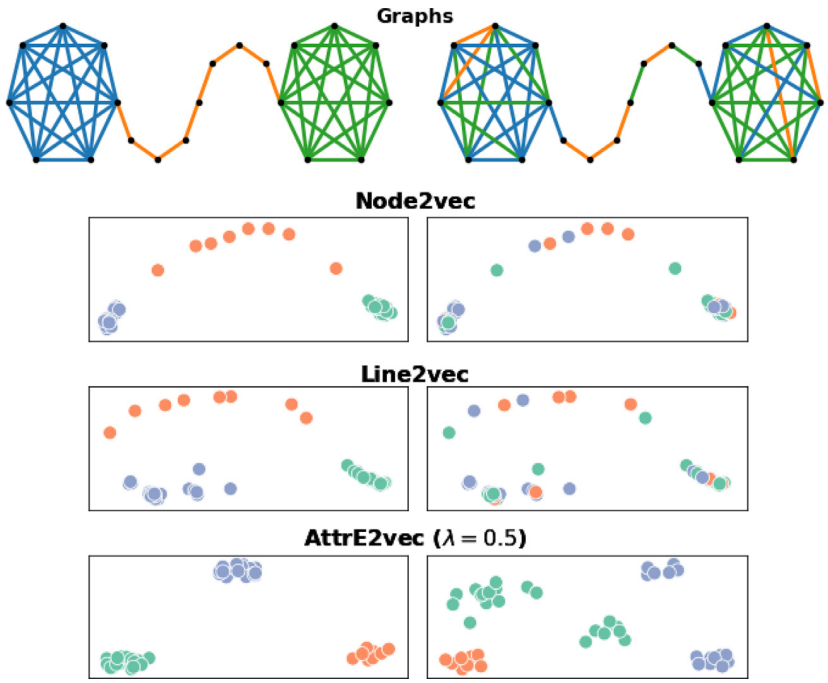


Fig. 1. Our proposed AttrE2vec model compared to other methods in the task of an attributed graph embedding. Colors denote edge features. On the left we can see a graph, where the features are aligned to substructures of the graph. On the right, the features were shuffled (ca. 50%). Traditional approaches fail to build robust representations, whereas our method includes features information to construct the embedding vectors.

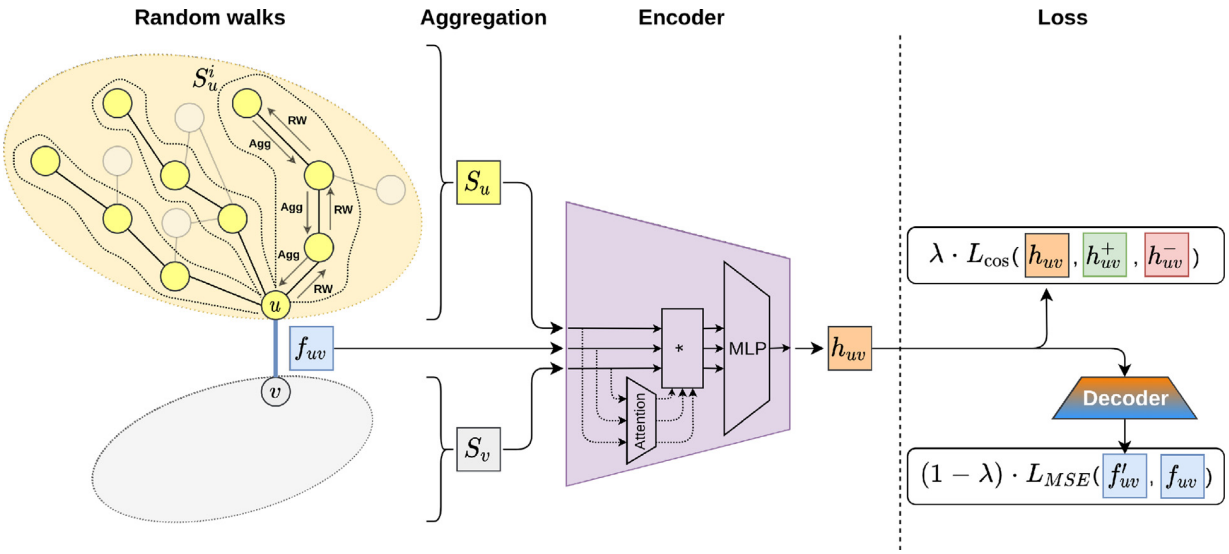


Fig. 2. Overview of the `AttrE2vec` model. The model first computes edge random walks on two neighborhoods of a given edge (u, v). Each neighbourhood walks are aggregated into S_u, S_v. Both are combined with the edge features f_{uv} using an Encoder module, which results into the edge embedding vector h_{uv}. The loss function consists of two parts: structural loss ($L_{\cos}$) and feature reconstruction loss (L_{MSE}).

$$h_{uv} = \mathbf{Enc}(f_{uv}, S_u, S_v)$$

The overall illustration of the method is contained in Fig. 2 and the inference algorithm is shown in Algorithm 1.

Algorithm 1: AttrE2vec inference algorithm

Data: graph G, edge list xe, edge features $\mathcal{F}$, node features $\mathcal{M}$
Params: number of random walks per node k, random walk length L
Result: edge embedding vectors h_{uv}
begin
  foreach *(u, v) in xe* **do**
    foreach *i in (1...k)* **do**
      $w_u^i = \mathbf{RW}(G, L, u)$
      $S_u^i = \mathbf{Agg_w}(w_u^i, \mathcal{F}, \mathcal{M})$
      $w_v^i = \mathbf{RW}(G, L, v)$
      $S_v^i = \mathbf{Agg_w}(w_v^i, \mathcal{F}, \mathcal{M})$
    end
    $S_u = \mathbf{Agg_n}(\{S_u^1, \ldots, S_u^k\})$
    $S_v = \mathbf{Agg_n}(\{S_v^1, \ldots, S_v^k\})$
    $h_{uv} = \mathbf{Enc}(f_{uv}, S_u, S_v)$
  end
end

3.4. Aggregation models

For the purpose of the neighborhood aggregation model $\mathbf{Agg_n}$, we use an average over vectors S_u^i, as there is no particular ordering of these vectors (each one was generated by an equally important random walk). In the case of walk aggregation, we propose the following:

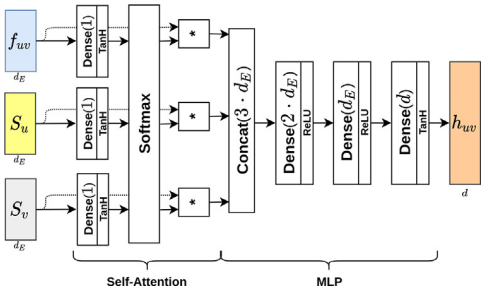


Fig. 3. Encoder module architecture.

- **average** – that computes a simple average of the edge attribute vectors in the random walk;

$$S_u^i = \frac{1}{L}\sum_{n=1}^{L} f_{u_n u_{n+1}}$$

- **exponential** – that computes a weighted average, where the weights are *exp*onents of the "minus" position in the random walk so that further away edges are less important than the near ones;

$$S_u^i = \frac{1}{L}\sum_{n=1}^{L} e^{-n} f_{u_n u_{n+1}}$$

- **GRU** – that uses a Gated Recurrent Unit [37] architecture, where hidden and input dimension is equal to the edge attribute dimension; the aggregated representation is the output of the last hidden vector; the aggregation process starts here at the end of the random walk and proceeds to the beginning;

$$S_u^i = \text{GRU}(\{f_{u_n u_{n+1}}, f_{u_{n-1} u_n}, \ldots, f_{u_1 u_2}\})$$

- **ConcatGRU** – that is similar to the GRU-based aggregator, but here we also use the node feature information by concatenating the node attributes with the edge attributes; hence the GRU input size is equal to the sum of the edge and node dimensions; in case there are not any node features available, one could use network-specific features, like degree, betweenness or more advanced techniques like Node2vec; the hidden dimension size and the aggregation direction is unchanged;

$$S_u^i = \text{ConcatGRU}(\{f_{u_n u_{n+1}} \oplus m_{u_n}, \ldots, f_{u_1 u_2} \oplus m_{u_1}\})$$

3.5. Learning AttrE2vec's parameters

`AttrE2vec` is designed to make the most use of edge attributes and information about the structure of the network. Therefore we propose a loss function, which consists of two main parts:

- structural loss $L_{\cos}$ – computes a **cosine embedding loss**; such function tries to minimize the cosine distance between a given embedding h and embeddings of edges sampled from the random walks h^+ (positive), and simultaneously to maximize a cosine distance between an embedding h and embeddings of edges sampled from a set of all edges in the graph h^- (negative), except for these in the random walks:

$$L_{\cos} = \frac{1}{|B|}\sum_{h_{\text{uv}} \in B}\left(\sum_{h_{\text{uv}}^+}(1 - \cos(h_{\text{uv}}, h_{\text{uv}}^+)) + \sum_{h_{\text{uv}}^-}\cos(h_{\text{uv}}, h_{\text{uv}}^-)\right)$$

  where B denotes a minibatch of edges and $|B|$ the minibatch size,
- feature reconstruction loss L_{MSE} – computes a **mean squared error** of the actual edge features and the outputs of a **decoder** (implemented as a 3-layer MLP – see Fig. 4), that reconstruct the edge features based on the edge embeddings;

$$L_{\text{MSE}} = \frac{1}{|B|}\sum_{(h_{\text{uv}} f_{\text{uv}}) \in B}(\text{DEC}(h_{\text{uv}}) - f_{\text{uv}})^2$$

  where B denotes a minibatch of edges and $|B|$ the minibatch size.

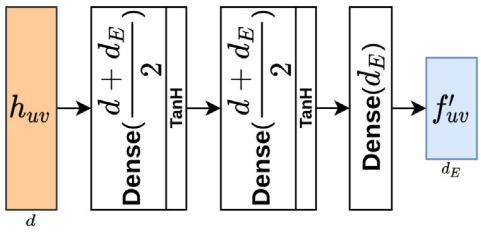


Fig. 4. Decoder module architecture.

We combine the values of the above loss functions using a mixing parameter $\lambda \in [0, 1]$. The higher the value of this parameter is, the more structural information is preserved and less focus is one the feature reconstruction. The total loss of AttrE2vec is given as follows:

$$L = \lambda * \ L_{\cos} + (1 - \lambda) * \ L_{\text{MSE}}$$

4. Experiments

To evaluate the proposed model's performance, we perform three tasks: edge classification, edge clustering, and embedding visualization on three real-world datasets. We first train our model on a small subset of edges (inductive setting). Then we use the model to infer embeddings for edges from the test set. Finally, we evaluate them in all downstream tasks: by predicting the class of edges in citation graphs (**edge classification**), by applying the K-means++ algorithm (**edge clustering**; as defined in [22]) and by the dimensionality reduction method T-SNE (**embedding visualization**). We compare our model to several baselines and contemporary methods in all experiments, see Table 1. Eventually, we check the influence of AttrE2vec's hyperparameters and perform an ablation study on artificially generated datasets. We implement our model in the popular deep learning framework PyTorch. All experiments were performed on NVIDIA GTX1080Ti. We make our code available at `https://github.com/attre2vec/attre2vec` and include our DVC [38] pipeline so that all experiments can be easily reproduced.

4.1. Datasets

In order to compare gathered evaluation evidence we focused on well known datasets, that appear in the literature, namely: Cora [39], Citeseer [39] and Pubmed [40]. These are citation networks of scientific papers in several research areas, where nodes are the papers and edges denote citations between papers. We summarize basic statistics about the datasets before and after pre-processing steps in Table 2. Raw datasets contain node features only in the form of high dimensional sparse bags of words. For Cora and Citeseer, these are binary vectors, showing which of the most popular words were used in a given paper, and for Pubmed, the features are in the form of TF-IDF vectors. To adjust the datasets to our problem setting, we apply the following pre-processing steps to obtain edge level features, which are used to train and evaluate our `AttrE2vec` model:

- we create dense vector representations of the nodes' features by applying Doc2vec [41] in the PV-DBOW variant with a target dimension size of 128;
- for each edge (u, v) and its symmetrical version (v, u) (necessary to perform uniform, undirected random walks) we extract the following features:
  - 1 feature – cosine similarity of raw node features for nodes u and v (binary BoW; for Pubmed transformed from TF-IDF to binary BoW),
  - 2 features – the ratios of the number of used words (number of ones in the BoW) to all possible words in the document (length of BoW vector) in each paper u and v,

Table 2
Datasets used in the experiments.

Name	Features				Number of		Training instances		
	initial		pre-processed						
	node	edge	node	edge	nodes	edges	classes	inductive	transductive
Cora	1 433	0	32	260	2 485	5 069	7 + 1	160	5 069
Citeseer	3 703	0	32	260	2 110	3 668	6 + 1	140	3 668
Pubmed	500	0	32	260	19 717	44 324	3 + 1	80	44 324

  - 256 features – concatenation of Doc2vec features for nodes u and v,
  - 1 feature – a binary indicator, which denotes whether this is an original edge (1) or its symmetrical counterpart (0),
- we apply standardization (StandardScaler in Scikit-Learn [42]) of the edge feature matrix.

Moreover, we extracted new node features as 32-dimensional Node2vec embeddings to provide the evaluation possibility for one of our model versions (AttrE2vec with ConcatGRU aggregator), which generalizes upon both edge and nodes attributes.

Raw datasets provide each node labeled by the research area the paper comes from. To apply this knowledge in the edge classification problem setting, we applied the following rule: if an edge has two nodes from the same class (research area), the edge receives this class; if two nodes have different classes, the edge between these nodes is assigned with a cross-domain citation class.

To ensure a fair comparison method, we follow the dataset preparation scheme from EP-B [12], i.e., for each dataset (Cora, Citeseer, Pubmed) we sample 10 train/validation/test sets, where the train set consists of 20 edges per class and the validation and test sets to contain 1 000 randomly chosen edges each. While reporting the resulting metrics, we show the mean values over these ten sampled sets (together with the standard deviation).

4.2. Baselines

We compare our method against several baseline methods. In the most simple case, we use the edge features obtained during the pre-processing phase for all datasets (further referred to as *Doc2vec*).

Many standard approaches employ simple node embedding transformations to obtain edge embeddings. The authors of Node2vec [36] proposed binary operators like averaging, Hadamard product, or L1 and L2 norms of vector differences. Here, we will use following methods to obtain node embeddings: DeepWalk [8], Node2vec [36], SDNE [43] and Struc2vec [35]. In preliminary experiments, we evaluated these methods and checked that the Average operator and an embedding size of 64 gives the best results. We will use these models in 2 setups: (a) **Avg(**$\mathcal{M}, \mathcal{M}$**)** – using only the averaged node features, (b) **Avg (**$\mathcal{M}, \mathcal{M}$**)**$\oplus\mathcal{F}$ – like previously but concatenated with the edge features from the dataset (in total 324-dim vectors).

We also checked a scheme to compute a 64-dim PCA reduction of the concatenated features to have comparable vector sizes with the 64-dimensional embedding of our model, but these turned out to perform poorly. Note that SDNE has the capability of inductive reasoning, but due to the non-availability of such implementation, we decided to evaluate this method in the transductive scheme (which works in favor of the method).

We also extend our body of baselines by more sophisticated approaches – two dense autoencoder architectures. In the first setting **MLP(**$\mathcal{M}, \mathcal{M}$**)**, we train a model (see Fig. 5), which reconstructs concatenated embeddings of connected nodes. In the second baseline **MLP(**$\mathcal{M}, \mathcal{M}, \mathcal{F}$**)**, the autoencoder (see Fig. 6) is extended by edge attributes. In both settings, we employ the mean squared error as the model loss function. The output of the encoders (embeddings) is used in the downstream tasks. The input node embeddings are obtained using the methods mentioned above, i.e., DeepWalk, Node2vec, SDNE, and Struc2vec.

The last baseline is Line2vec [22], which is directly dedicated for edges - we use an embedding size of 64.

4.3. Edge classification

To evaluate our model in an inductive setting, we need to make sure that test edges are unseen during the model training procedure – we remove them from the graph. Note that all baselines (except for GraphSage, see 1) require all edges during the training phase (i.e., these are transductive methods).

After each training epoch of `AttrE2vec`, we evaluate the embeddings using L2-regularized Logistic Regression (LR) classifier and compute AUC. The regression model is trained on edge embeddings from the train set and evaluated on edge embeddings from the validation set. We take the model with the highest AUC value on the validation set. Moreover, an early stopping strategy is implemented– if the validation AUC metric does not improve for more than 15 epochs, the learning is terminated. Our approach to model selection is aligned with the schema proposed in [44] because this approach is more

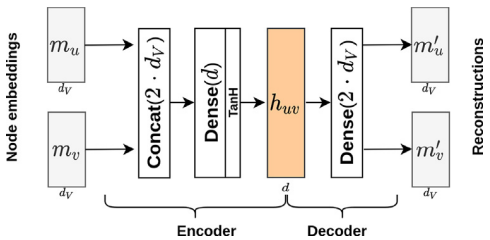


Fig. 5. Architecture of the **MLP(**$\mathcal{M}, \mathcal{M}$**)**.

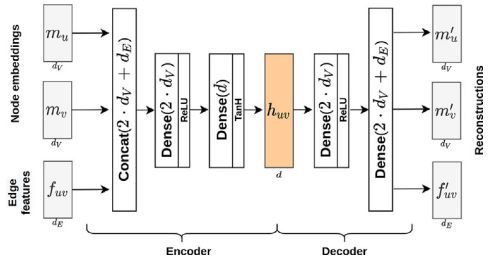


Fig. 6. Architecture of the **MLP**($\mathcal{M}, \mathcal{M}, \mathcal{F}$).

natural than relying on the loss function. This is repeated for all 10 data splits (see: Section 4.1 for details). We report a mean and std AUC measures for 10 test sets (see Table 3).

We choose AdamW [45] with a learning rate of 0.001 to optimize our model's parameters. We also set the size of positive samples to $|h^+| = 5$ and negative samples to $|h^-| = 10$ in the cosine embedding loss. The mixing coefficient is set to $\lambda = 0.5$, equally including the influence of features and topological graph structure. We choose an embedding size of 64 as a reasonable value while dealing with edge features of size 260.

In Table 3, we summarize the AUC values for baseline methods and for our model. Even though vectors' original dimensionality is relatively high (260), good results are already yielded using only the edge features (Doc2vec). However, adding structural information about the graph could further improve the results.

Using representations from node embedding methods, which are transformed to edge embeddings using the average operator **Avg**($\mathcal{M}, \mathcal{M}$), achieve poor results of about 50–60% AUC. However, if these are combined with the edge features from the datasets **Avg**($\mathcal{M}, \mathcal{M}$)$\oplus\mathcal{F}$, the AUC values increase significantly to about 86%, 88% and 79% for Citeseer, Cora, and Pubmed, respectively. Unfortunately, this results in an even higher vector dimensionality (324).

The MLP-based approach results lead to similar conclusions. Using only node embeddings **MLP**($\mathcal{M}, \mathcal{M}$) we achieve quite poor results of about 50% (on Pubmed) up to 60% (on Cora). With **MLP**($\mathcal{M}, \mathcal{M}, \mathcal{F}$) approach we observe that edge features improve the classification results. The AUC values are still slightly worse than concatenation operator (**Avg**($\mathcal{M}, \mathcal{M}$)$\oplus\mathcal{F}$), but we can reduce the edge embedding size to 64.

Table 3
AUC values for edge classification. $\mathcal{F}$ denotes the edge attributes (also referred to as "Doc2vec"), $\mathcal{M}$ – node attributes (e.g., embeddings computed using "Node2vec"), $\oplus$ – concatenation operator, **Avg**($\mathcal{M}, \mathcal{M}$) – average operator on node embeddings, **MLP**($\cdot$) – encoder output of MLP autoencoder trained on given attributes. **AUC** in bold shows the highest value and *AUC* in italic – the second highest value.

	Method group/name	Vector size	AUC			
			Citeseer	Cora	Pubmed	
Transductive	**Edge features only;** $\mathcal{F}$ (Doc2vec)		260	86.13 ± 0.95	88.67 ± 0.51	79.15 ± 1.41
	Line2vec		64	86.19 ± 0.28	91.75 ± 1.07	84.88 ± 1.19
	Avg($\mathcal{M}, \mathcal{M}$)	DeepWalk	64	58.40 ± 1.08	59.98 ± 1.32	51.04 ± 1.23
		Node2vec	64	58.26 ± 0.89	59.59 ± 1.11	51.03 ± 1.01
		SDNE	64	54.28 ± 1.57	55.91 ± 1.11	50.00 ± 0.00
		Struc2vec	64	61.29 ± 0.86	61.30 ± 1.58	54.67 ± 1.46
	MLP($\mathcal{M}, \mathcal{M}$)	DeepWalk	64	55.88 ± 1.68	57.87 ± 1.53	51.23 ± 0.77
		Node2vec	64	55.35 ± 2.26	57.44 ± 0.87	51.48 ± 1.55
		SDNE	64	55.56 ± 0.93	56.02 ± 1.22	50.00 ± 0.00
		Struc2vec	64	59.93 ± 1.43	59.76 ± 1.80	53.27 ± 1.32
	Avg($\mathcal{M}, \mathcal{M}$)$\oplus\mathcal{F}$	DeepWalk	324	86.13 ± 0.95	88.67 ± 0.51	79.15 ± 1.41
		Node2vec	324	86.13 ± 0.95	88.67 ± 0.51	79.15 ± 1.41
		SDNE	324	86.14 ± 1.03	88.70 ± 0.51	79.15 ± 1.41
		Struc2vec	324	86.21 ± 0.97	88.73 ± 0.48	79.24 ± 1.36
	MLP($\mathcal{M}, \mathcal{M}, \mathcal{F}$)	DeepWalk	64	84.58 ± 1.11	86.47 ± 0.87	78.60 ± 1.84
		Node2vec	64	84.65 ± 1.05	86.71 ± 0.68	78.84 ± 1.71
		SDNE	64	84.32 ± 1.13	85.99 ± 0.77	78.34 ± 1.07
		Struc2vec	64	83.95 ± 1.16	85.54 ± 0.96	77.19 ± 1.42
Inductive	**Avg**($\mathcal{M}, \mathcal{M}$)	GraphSage	64	54.84 ± 1.90	55.16 ± 1.36	51.14 ± 1.64
	MLP($\mathcal{M}, \mathcal{M}$)	GraphSage	64	55.19 ± 1.04	55.47 ± 1.66	50.36 ± 1.54
	Avg($\mathcal{M}, \mathcal{M}$)$\oplus\mathcal{F}$	GraphSage	324	86.14 ± 0.95	88.68 ± 0.51	79.16 ± 1.41
	MLP($\mathcal{M}, \mathcal{M}, \mathcal{F}$)	GraphSage	64	84.63 ± 1.11	86.14 ± 0.45	78.00 ± 1.85
	AttrE2vec (our)	Avg	64	**88.97 ± 0.82**	**93.43 ± 0.56**	**87.68 ± 1.25**
		Exp	64	88.91 ± 1.10	92.80 ± 0.38	86.18 ± 1.41
		GRU	64	88.92 ± 1.13	93.06 ± 0.63	86.39 ± 1.21
		ConcatGRU	64	88.56 ± 1.34	92.93 ± 0.61	86.34 ± 1.18

The Line2vec [22] algorithm achieves very good results, without considering edge features information – we get about 86%, 92% and 85% AUC for Citeseer, Cora, and Pubmed, respectively. These values are higher than for any other baseline approach.

Our model performs the best among all evaluated methods. For Citeseer, we gain about 3 percent points compared to the best baselines: Line2vec, Struc2vec (**Avg(**$\mathcal{M},\mathcal{M}$**)**$\oplus\mathcal{F}$) or GraphSage (**Avg(**$\mathcal{M},\mathcal{M}$**)**$\oplus\mathcal{F}$). Note that the algorithm is trained only on 140 edges in the inductive setting, whereas all transductive baselines require the whole graph for training. The gains on Cora are 2 pp, and on Pubmed we achieve up to 4 pp (and up to 8 pp compared only to GraphSage (**Avg(**$\mathcal{M},\mathcal{M}$**)**$\oplus\mathcal{F}$**)**). Our model with the Average (Avg) aggregator works the best, whereas the Gated Recurrent Unit (GRU) aggregator achieves the second-best results.

4.4. Edge clustering

Similarly to Line2vec [22], we apply the K-Means++ algorithm on the resulting embedding vectors and compute an unsupervised clustering accuracy [46]. We summarize the results in Table 4. Our model performs the best in all but one case and achieves significantly better results than other baseline methods. The only exception is for the Pubmed dataset, where Line2vec achieves the best clustering accuracy. Other baseline methods perform similarly as in the edge classification task. Hence, we will not discuss the details, and we encourage the reader to go through the detailed results.

4.5. Embedding visualization

For all tested baseline methods and our proposed `AttrE2vec` method, we compute 2-dimensional projections of the produced embeddings using T-SNE [47] method. We visualize them in Fig. 7. In our subjective opinion, these plots correspond to the AUC scores reported in Table 3-the higher the AUC, the better the group separation. In details, for Doc2vec raw edge features seem to form groups, but unfortunately overlap to some degree. We cannot observe any pattern in the node embedding-based settings (**Avg(**$\mathcal{M},\mathcal{M}$**)** and **MLP(**$\mathcal{M},\mathcal{M}$**)**), they tempt to be quasi-random. When concatenated with the edge attributes (**Avg(**$\mathcal{M},\mathcal{M}$**)**$\oplus\mathcal{F}$ and **MLP(**$\mathcal{M},\mathcal{M},\mathcal{F}$**)**) we observe a slightly better grouping, but yet non satisfying. `AttrE2vec` model produces much more formed groups, with only a little overlapping. To summarize, based on the observed groups' separability and AUC metrics, our approach works the best among all methods.

Table 4
Accuracy on edge clustering. $\mathcal{F}$ denotes the edge attributes (also referred to as "Doc2vec"), $\mathcal{M}$ – node attributes (e.g., embeddings computed using "Node2vec"), $\oplus$ – concatenation operator, **Avg(**$\mathcal{M},\mathcal{M}$**)** – average operator on node embeddings, **MLP(**$\cdot$**)** – encoder output of MLP autoencoder trained on given attributes. **AUC** in bold shows the highest value and *AUC* in italic—the second highest value.

	Method group/name	Vector size	Accuracy			
			Citeseer	Cora	Pubmed	
Transductive	**Edge features only;** $\mathcal{F}$ (Doc2vec)		260	54.13 ± 2.73	54.64 ± 5.86	46.33 ± 1.53
	Line2vec		64	54.73 ± 2.56	63.50 ± 1.92	**55.26** ± **1.36**
	Avg($\mathcal{M},\mathcal{M}$**)**	DeepWalk	64	28.89 ± 1.06	21.93 ± 0.86	27.24 ± 0.50
		Node2vec	64	26.82 ± 0.67	21.32 ± 0.62	27.17 ± 0.74
		SDNE	64	21.01 ± 0.50	17.97 ± 0.47	31.38 ± 0.69
		Struc2vec	64	25.21 ± 1.33	20.15 ± 0.64	32.02 ± 1.49
	MLP($\mathcal{M},\mathcal{M}$**)**	DeepWalk	64	26.36 ± 1.37	21.06 ± 0.57	27.40 ± 0.93
		Node2vec	64	26.37 ± 1.64	21.31 ± 0.98	27.67 ± 0.78
		SDNE	64	22.27 ± 0.76	17.15 ± 0.36	28.44 ± 1.21
		Struc2vec	64	24.22 ± 0.83	19.56 ± 0.49	31.31 ± 1.70
	Avg($\mathcal{M},\mathcal{M}$**)**$\oplus\mathcal{F}$	DeepWalk	324	54.13 ± 2.73	54.70 ± 5.85	46.33 ± 1.53
		Node2vec	324	54.13 ± 2.73	54.70 ± 5.85	46.33 ± 1.53
		SDNE	324	55.29 ± 2.06	55.43 ± 4.63	46.33 ± 1.53
		Struc2vec	324	55.59 ± 1.51	52.47 ± 6.52	46.32 ± 1.29
	MLP($\mathcal{M},\mathcal{M},\mathcal{F}$**)**	DeepWalk	64	48.74 ± 4.03	47.38 ± 4.72	46.49 ± 1.20
		Node2vec	64	50.80 ± 2.30	48.48 ± 3.38	46.15 ± 1.43
		SDNE	64	46.17 ± 3.15	44.87 ± 3.54	45.74 ± 1.89
		Struc2vec	64	47.35 ± 3.73	44.38 ± 3.04	45.40 ± 1.72
Inductive	**Avg(**$\mathcal{M},\mathcal{M}$**)**	GraphSage	64	18.79 ± 0.62	17.70 ± 1.05	27.04 ± 0.71
	MLP($\mathcal{M},\mathcal{M}$**)**	GraphSage	64	18.92 ± 0.98	17.89 ± 0.85	27.09 ± 0.81
	Avg($\mathcal{M},\mathcal{M}$**)**$\oplus\mathcal{F}$	GraphSage	324	54.06 ± 2.54	54.82 ± 6.86	46.49 ± 1.64
	MLP($\mathcal{M},\mathcal{M},\mathcal{F}$**)**	GraphSage	64	48.79 ± 4.04	47.49 ± 5.41	45.15 ± 1.54
	AttrE2vec (our)	Avg	64	59.82 ± 3.30	65.42 ± 1.71	48.86 ± 2.46
		Exp	64	59.07 ± 4.65	**66.36** ± **3.62**	48.02 ± 2.55
		GRU	64	60.16 ± 2.25	66.15 ± 3.71	49.41 ± 1.49
		ConcatGRU	64	**60.71** ± **2.75**	66.00 ± 2.21	50.27 ± 3.75

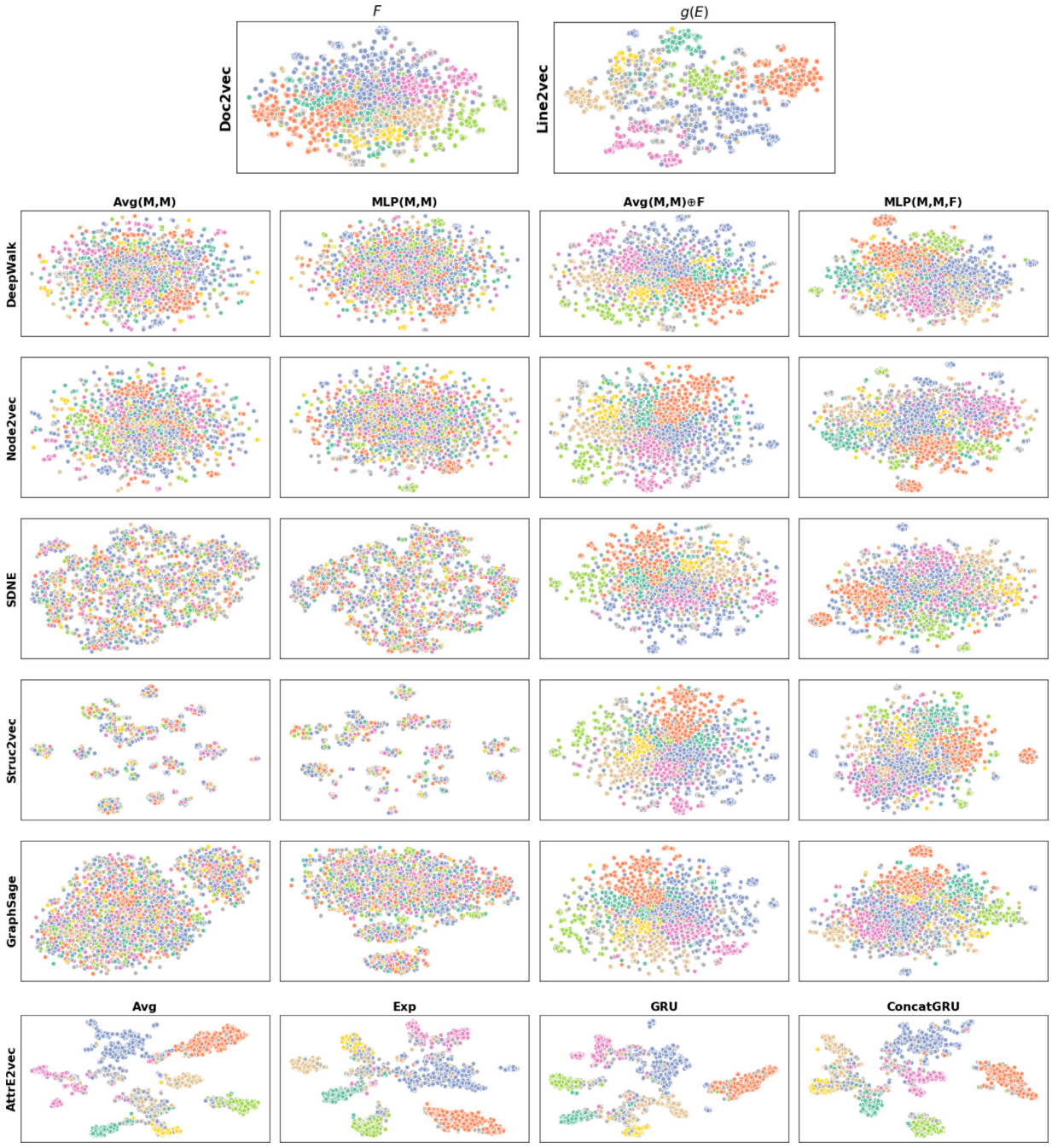


Fig. 7. 2-D T-SNE projections of embedding vectors for all evaluated methods. Columns denotes aggregation approach, beside F that denotes the edge attributes and $g(E)$ that is an edge embedding obtained with graph structure only. Rows gather particular methods.

5. Hyperparameter Sensitivity of AttrE2vec

We investigate hyperparameters' effect considering each of them independently, i.e., setting a given parameter and preserving default values for all other parameters. The evaluation is applied for our model's two inductive variants: with the Average aggregator and with the GRU aggregator. We use all three datasets (Cora, Citeseer, Pubmed) and report the AUC values. We choose following hyperparameter value sets (values with an asterisk denote the default value for that parameter):

- length of random walk: $L = \{4, \mathbf{8}^*, 16\}$,

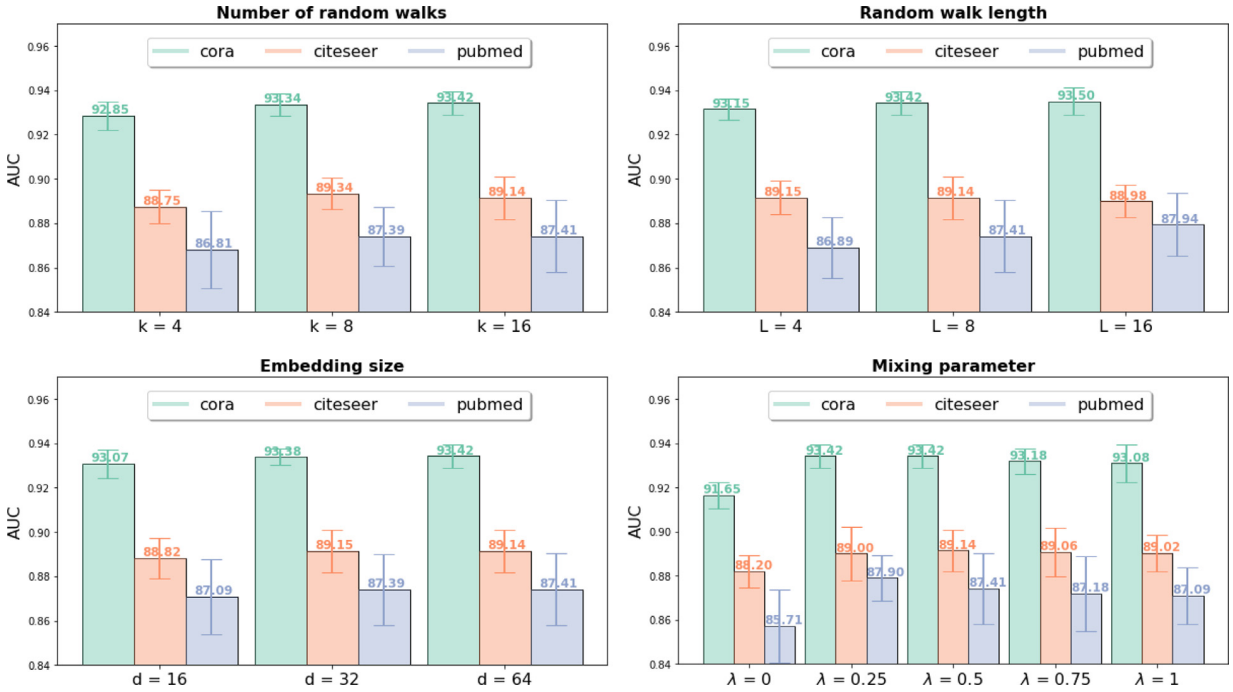


Fig. 8. Effects of hyperparameters on Cora, Citeseer and Pubmed datasets.

- number of random walks: $k = \{4, 8, \mathbf{16}^*\}$,
- embedding size: $d = \{16, 32, \mathbf{64}^*\}$,
- mixing parameter: $\lambda = \{0, 0.25, \mathbf{0.5}^*, 0.75, 1\}$.

The results of all experiments are summarized in Fig. 8. We observe that for both aggregation variants, Avg and GRU, the trends are similar, so we will include and discuss them based only on the Average aggregator.

In general, the higher the number of random walks k and the length of a single random walk L, the better results are achieved. One may require higher values of these parameters, but it significantly increases the random walk computation time and the model training itself.

Unsurprisingly, the embedding size (embedding dimension) also follows the same trend. With more dimensions, we can fit more information into the created representations. However, as an embedding goal is to find **low-dimensional** vector representations, we should keep reasonable dimensionality. Our chosen values (16, 32, 64) seem plausible while working with 260-dimensional edge features.

As for loss mixing parameter λ, we observe that too high values negatively influence the model performance. The greater the value, the more critical the structural loss becomes. Simultaneously the feature loss becomes less relevant. Choosing $\lambda = 0$ causes the loss function to consider feature reconstruction only and completely ignores the embedding loss. This yields significantly worse results and confirms that our approach of combining both feature reconstruction and structural embedding loss is justified. In general, the best values are achieved for setting an equal influence of both loss factors ($\lambda = 0.5$).

6. Ablation study

We performed an ablation study to check whether our method `AttrE2vec` is invariant to introduced noise in an artificially generated network. We use a barbell graph, which consists of two fully connected graphs and a path which connects them (see: Fig. 1). The graph has seven nodes in each full graph and seven nodes in the path – a total of 50 edges. Next, we generate features from 3 clusters in a 200-dimensional space using isotropic Gaussian blobs. We assign the features to 3 parts of the graph: the first to the edges in one of the full graphs, the second to the edges in the path and the third to the edges in the other full graph. The edge classes are matching the feature clusters (i.e., three classes). Therefore, the structure is aligned with the features, so any good structure based embedding method can fit this data very well (see: Fig. 1). A problem occurs when the features (and hence the classes) are shuffled within the graph structure. Methods that employ only a structural loss function will fail. We want to check how our model `AttrE2vec`, which includes both structural and feature-based loss, performs with different amount of such noise.

We will use the graph mentioned above and introduce noise by shuffling $p\%$ of all edge pairs, which are from different classes, i.e., an edge with class 2 (originally located in the path) may be swapped with one from the full graphs (classes 1 or

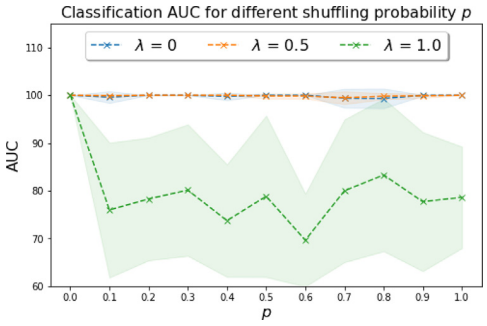


Fig. 9. `AttrE2vec` performance for various noise levels p and mixing parameter values $\lambda \in \{0, 0.5, 1\}$.

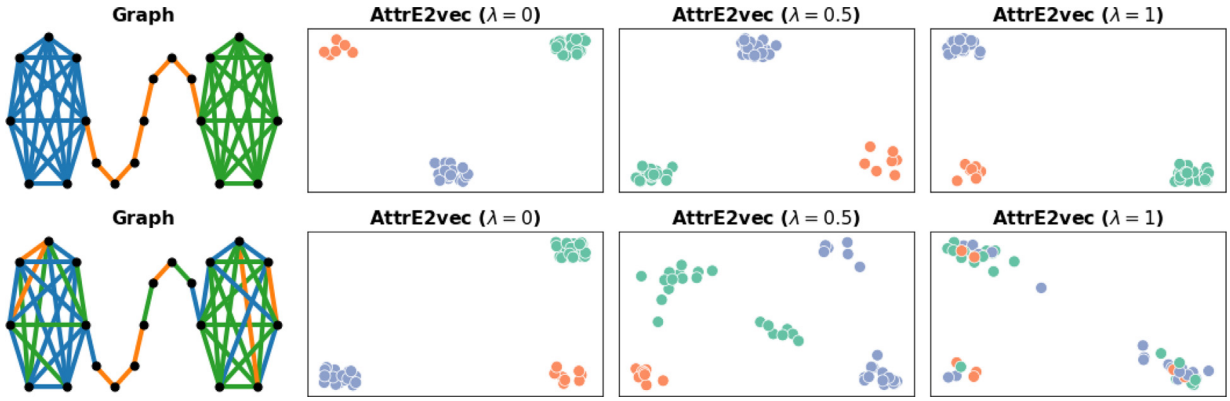


Fig. 10. 2-D representations of ideal and noisy graph edges using `AttrE2vec` with $\lambda \in \{0, 0.5, 1\}$.

3). We use our AttrE2vec model with an Average aggregator in the transductive setting (due to the graph size) and report the edge classification AUC for different values of $p \in \{0, 0.1, \ldots, 0.5, \ldots, 0.9, 1\}$ and $\lambda \in \{0, 0.5, 1\}$. The values of the mixing parameter λ allow us to check how the model behaves when working only with a feature-based loss ($\lambda = 0$), only with a structural loss ($\lambda = 1$), and with both losses at equal importance ($\lambda = 0.5$). We train our model for five epochs and repeat the computations ten times for every (p, λ) pair, due to the shuffling procedure's randomness. We report the mean and standard deviation of the AUC value in Fig. 9.

Using only the feature loss or a combination of both losses allows us to achieve nearly 100% AUC in the classification task. The fluctuations appear due to the low number of training epochs and the local optima problem. The performance of the model that uses only structural loss ($\lambda = 1$) decreases with higher shuffling probabilities, and from a certain point, it starts improving slightly because shuffling results in a complete swap of two classes, i.e., all features and classes from one graph part are exchanged with all features and classes from another part of the graph.

We also demonstrate how our method reacts on noisy data with various $\lambda \in \{0, 0.5, 1\}$. There are two graphs: one where the features are aligned to substructures of the graph and the second with shuffled features (ca. 50%), see Fig. 10. Keeping `AttrE2vec` with $\lambda = 0.5$ allows to represent noisy graphs fairly.

7. Conclusions and future work

We introduce `AttrE2vec` – the novel unsupervised and inductive embedding model to learn attributed edge embeddings by leveraging on the self-attention network with auto-encoder over attribute space and structural loss on aggregated random walks. *Attre2vec* can directly aggregate feature information from edges and nodes at many hops away to infer embeddings not only for present nodes, but also for new nodes. Extensive experimental results show that `AttrE2vec` obtains the state-of-the-art results in edge classification and clustering on CORA, PUBMED and CITESEER.

CRediT authorship contribution statement

Piotr Bielak: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Tomasz Kajdanowicz:** Conceptualization, Methodology, Software, Valida-

tion, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Supervision. **Nitesh V. Chawla:** Conceptualization, Formal analysis, Writing – review & editing, Supervision, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work was partially supported by the National Science Centre, Poland Grant No. 2016/21/D/ST6/02948, and 2016/23/B/ST6/01735, as well as by the Department of Computational Intelligence, Wrocław University of Science and Technology statutory funds.

References

[1] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec, R. Barzilay, P. Battaglia, Y. Bengio, M. Bronstein, S. Günnemann, W. Hamilton, T. Jaakkola, S. Jegelka, M. Nickel, C. Re, L. Song, J. Tang, M. Welling, R. Zemel, Open graph benchmark: Datasets for machine learning on graphs (may 2020). arXiv:2005.00687. http://arxiv.org/abs/2005.00687..
[2] D. Zhang, J. Yin, X. Zhu, C. Zhang, Network Representation Learning: A Survey, IEEE Trans. Big Data 6 (1) (2018) 3–28, https://doi.org/10.1109/tbdata.2018.2850013.
[3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P.S. Yu, A Comprehensive Survey on Graph Neural Networks, IEEE Trans. Neural Networks Learn. Syst. (2019) 1–21, https://doi.org/10.1109/TNNLS.2020.2978386.
[4] B. Li, D. Pi, Network representation learning: a systematic literature review, Neural Comput. Appl. 32 (21) (2020) 16647–16679, https://doi.org/10.1007/s00521-020-04908-5.
[5] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, K. Murphy, Machine Learning on Graphs: A Model and Comprehensive Taxonomy (2020), http://arxiv.org/abs/2005.03675.
[6] S. Bahrami, F. Dornaika, A. Bosaghzadeh, Joint auto-weighted graph fusion and scalable semi-supervised learning, Inf. Fusion 66 (2021) 213–228, URL: www.scopus.com.
[7] A. Grover, J. Leskovec, Node2vec: Scalable feature learning for networks, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Vol. 13–17-Augu, 2016, pp. 855–864. doi:10.1145/2939672.2939754..
[8] B. Perozzi, R. Al-Rfou, S. Skiena, DeepWalk: Online Learning of Social Representations Bryan, in: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14, ACM Press, New York, New York, USA, 2014, pp. 701–710. doi:10.1145/2623330.2623732. URL: http://dl.acm.org/citation.cfm?doid=2623330.2623732..
[9] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, International Conference on Learning Representations, ICLR, 2017, pp. 1–14, arXiv:1609.02907.
[10] Y. Dong, N.V. Chawla, A. Swami, Metapath2vec: Scalable representation learning for heterogeneous networks, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Vol. Part F1296, ACM, New York, NY, USA, 2017, pp. 135–144. doi:10.1145/3097983.3098036. https://dl.acm.org/doi/10.1145/3097983.3098036..
[11] S. Wang, V.V. Govindaraj, J.M. Górriz, X. Zhang, Y. Zhang, Covid-19 classification by fgcnet with deep feature fusion from graph convolutional network and convolutional neural network, Information Fusion 67 (2021) 208–229, cited By:1. URL: www.scopus.com..
[12] A. García-Durán, M. Niepert, Learning graph representations with embedding propagation, in: Advances in Neural Information Processing Systems, vol. 2017-Decem, 2017, pp. 5120–5131..
[13] W.L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in Neural Information Processing Systems, vol. 2017-Decem, 2017, pp. 1025–1035..
[14] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, Y. Bengio, Graph attention networks, in: 6th International Conference on Learning Representations, ICLR 2018 – Conference Track Proceedings, International Conference on Learning Representations, ICLR, 2018, pp. 1–12. arXiv:1710.10903..
[15] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, vol. 13–17-Augu, 2016, pp. 1225–1234. doi:10.1145/2939672.2939753..
[16] C. Yang, Z. Liu, D. Zhao, M. Sun, E.Y. Chang, Network representation learning with rich text information, IJCAI International Joint Conference on Artificial Intelligence (2015-, 2015,) 2111–2117.
[17] M. Liu, J. Liu, Y. Chen, M. Wang, H. Chen, Q. Zheng, Ahng: Representation learning on attributed heterogeneous network, Inf. Fusion 50 (2019) 221–230, cited By:3. URL: www.scopus.com..
[18] L. Lan, P. Wang, J. Zhao, J. Tao, J. Lui, X. Guan, Improving network embedding with partially available vertex and edge content, Inf. Sci. 512 (2020) 935–951, https://doi.org/10.1016/j.ins.2019.09.083.
[19] B. Li, D. Pi, Y. Lin, I. Khan, L. Cui, Multi-source information fusion based heterogeneous network embedding, Inf. Sci. 534 (2020) 53–71, https://doi.org/10.1016/j.ins.2020.05.012.
[20] C. Zhang, D. Song, C. Huang, A. Swami, N.V. Chawla, Heterogeneous graph neural network, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2019, pp. 793–803, https://doi.org/10.1145/3292500.3330961.
[21] H. Gao, H. Huang, Deep attributed network embedding, in: IJCAI International Joint Conference on Artificial Intelligence, vol. 2018-July, 2018, pp. 3364–3370. doi:10.24963/ijcai.2018/467..
[22] S. Bandyopadhyay, A. Biswas, N. Murty, R. Narayanam, Beyond node embedding: A direct unsupervised edge representation framework for homogeneous networks (2019). arXiv:1912.05140..
[23] Y. Chen, T. Qian, Relation constrained attributed network embedding, Inf. Sci. 515 (2020) 341–351, https://doi.org/10.1016/j.ins.2019.12.033.
[24] S. Bandyopadhyay, H. Kara, A. Kannan, M.N. Murty, FSCNMF: Fusing structure and content via non-negative matrix factorization for embedding information networks (2018). arXiv:1804.05313..
[25] D. Nozza, E. Fersini, E. Messina, CAGE: Constrained deep Attributed Graph Embedding, Inf. Sci. 518 (2020) 56–70, https://doi.org/10.1016/j.ins.2019.12.082.
[26] J. Kim, T. Kim, S. Kim, C.D. Yoo, Edge-labeling graph neural network for few-shot learning, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2019-June, 2019, pp. 11–20. arXiv:1905.01436, doi:10.1109/CVPR.2019.00010..
[27] Q. Li, Z. Cao, J. Zhong, Q. Li, Graph representation learning with encoding edges, Neurocomputing 361 (2019) 29–39, https://doi.org/10.1016/j.neucom.2019.07.076.
[28] L. Gong, Q. Cheng, Exploiting edge features for graph neural networks, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019, pp. 9203–9211, https://doi.org/10.1109/CVPR.2019.00943.

[29] C. Aggarwal, G. He, P. Zhao, Edge classification in networks, in: 2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016, Institute of Electrical and Electronics Engineers Inc, 2016, pp. 1038–1049, https://doi.org/10.1109/ICDE.2016.7498311.
[30] M. Simonovsky, N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, in: Proceedings – 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, vol. 2017-Janua, 2017, pp. 29–38. doi:10.1109/CVPR.2017.11..
[31] T.D. Bui, S. Ravi, V. Ramavajjala, Neural Graph Learning: Training Neural Networks Using Graphs, dl.acm.org 2018-Febua (2018) 64–71. doi:10.1145/3159652.3159731..
[32] Y. Wang, Y. Sun, M.M. Bronstein, J.M. Solomon, Z. Liu, S.E. Sarma, Dynamic Graph CNN for Learning on Point Clouds, ACM Trans. Graphics 38 (5) (2019) 146, https://doi.org/10.1145/3326362.
[33] T. Wanyan, C. Zhang, A. Azad, X. Liang, D. Li, Y. Ding, Attribute2vec: Deep network embedding through multi-filtering GCN (apr 2020). arXiv:2004.01375. http://arxiv.org/abs/2004.01375..
[34] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, LINE Large-scale information network embedding, in: WWW 2015 – Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 1067–1077, https://doi.org/10.1145/2736277.2741093.
[35] L.F. Ribeiro, P.H. Saverese, D.R. Figueiredo, Struc2vec: Learning node representations from structural identity, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, vol. Part F1296, 2017, pp. 385–394. doi:10.1145/3097983.3098061..
[36] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining ACM, 2016, pp. 855–864.
[37] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling (dec 2014). arXiv:1412.3555. http://arxiv.org/abs/1412.3555..
[38] R. Kuprieiev, D. Petrov, R. Valles, P. Redzyński, C. da Costa-Luis, A. Schepanovski, I. Shcheklein, S. Pachhai, J. Orpinel, F. Santos, A. Sharma, Zhanibek, D. Hodovic, P. Rowlands, Earl, A. Grigorev, N. Dash, G. Vyshnya, maykulkarni, Vera, M. Hora, xliiv, W. Baranowski, S. Mangal, C. Wolff, nik123, O. Yoktan, K. Benoy, A. Khamutov, A. Maslakov, Dvc: Data version control - git for data & models (May 2020). doi:10.5281/zenodo.3859749. doi: 10.5281/zenodo.3859749..
[39] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, T. Eliassi-Rad, Collective classification in network data, AI Magazine 29 (3) (2008) 93, https://doi.org/10.1609/aimag.v29i3.2157, URL: https://ojs.aaai.org/index.php/aimagazine/article/view/2157.
[40] G. Namata, B. London, L. Getoor, B. Huang, Query-driven Active Surveying for Collective Classification, in: Proceedings ofthe Workshop on Mining and Learn- ing with Graphs, Edinburgh, Scotland, UK., 2012, pp. 1–8..
[41] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: 31st International Conference on Machine Learning, ICML 2014, Vol. 4, 2014, pp. 2931–2939. arXiv:1405.4053. http://arxiv.org/abs/1405.4053..
[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[43] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, ACM, New York, NY, USA, 2016, pp. 1225–1234. doi:10.1145/2939672.2939753..
[44] D.Q. Nguyen, T.D. Nguyen, D. Phung, A self-attention network based node embedding model (jun 2020). arXiv:2006.12100. http://arxiv.org/abs/2006.12100..
[45] I. Loshchilov, F. Hutter, Decoupled Weight Decay Regularization (nov 2017). arXiv:1711.05101. http://arxiv.org/abs/1711.05101..
[46] J. Xie, R. Girshick, A. Farhadi, in: M.F. Balcan, K.Q. Weinberger (Eds.), Unsupervised deep embedding for clustering analysis Proceedings of The 33rd International Conference on Machine Learning, vol. 48 of Proceedings of Machine Learning Research, PMLR, New York, New York, USA, 2016, pp. 478–487.
[47] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (2008) 2579–2605, URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

4.3 FILDNE: A Framework for Incremental Learning of Dynamic Networks Embeddings

Authors:	Piotr Bielak, Kamil Tagowski, Maciej Falkiewicz, Tomasz Kajdanowicz, Nitesh V. Chawla
Published at:	Knowledge-Based Systems (IF 8.139)
MEIN points:	200
Citations: (as of 05.06.2023)	Web of Science: 5 Scopus: 8 Google Scholar: 12
Credit:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization

Knowledge-Based Systems 236 (2022) 107453

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

FILDNE: A Framework for Incremental Learning of Dynamic Networks Embeddings

Piotr Bielak [a], Kamil Tagowski [a], Maciej Falkiewicz [a], Tomasz Kajdanowicz [a,*], Nitesh V. Chawla [a,b]

[a] Department of Computational Intelligence, Wroclaw University of Science and Technology, Poland
[b] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA

ARTICLE INFO

Article history:
Received 17 November 2020
Received in revised form 27 February 2021
Accepted 26 August 2021
Available online 30 August 2021

Keywords:
Representation learning
Dynamic graph embedding
Incremental network embedding

ABSTRACT

Representation learning on graphs has emerged as a powerful mechanism to automate feature vector generation for downstream machine learning tasks. The advances in representation on graphs have centered on both homogeneous and heterogeneous graphs, where the latter presenting the challenges associated with multi-typed nodes and/or edges. In this paper, we consider the additional challenge of evolving graphs. We ask the question of whether the advances in representation learning for static graphs can be leveraged for dynamic graphs and how? It is important to be able to incorporate those advances to maximize the utility and generalization of methods. To that end, we propose the Framework for Incremental Learning of Dynamic Networks Embedding (FILDNE), which can utilize any existing static representation learning method for learning node embeddings while keeping the computational costs low. FILDNE integrates the feature vectors computed using the standard methods over different timesteps into a single representation by developing a convex combination function and alignment mechanism. Experimental results on several downstream tasks, over seven real-world datasets, show that FILDNE is able to reduce memory (up to 6x) and computational time (up to 50x) costs while providing competitive quality measure gains (e.g., improvements up to 19 pp AUC on link prediction and up to 33 pp mAP on graph reconstruction) with respect to the contemporary methods for representation learning on dynamic graphs.

1. Introduction

Learning embeddings from networks or graphs is pervasive with a sundry of applications across various fields, including social networks [1–4], biological networks [5,6], molecular networks [7,8], spatial networks [9,10], citation networks [1,11,12], transportation networks [13] and many others. These embeddings are generally learned in an unsupervised fashion, providing an automated way of discovering dynamic features and enabling several downstream inductive learning tasks (and in some cases transductive learning tasks as well).

The vast majority of graph embedding methods are devoted to so-called static networks, whose structure does not evolve over time [14–16]. However, in most real-world scenarios, one has to deal with changes in the data, e.g., updates of node attributes and structural adjustments, like addition or removal of edges (links) and nodes [17–19]. On the one hand, dynamics can occur infrequently, which can be easily addressed with static approaches. On the other hand, there are streams of temporal events that constitute constantly evolving networks. The latter case requires dedicated, computationally, and memory-wise efficient solutions. Thus, a key challenge remains: how to effectively learn network embeddings on dynamic networks? While recent works have addressed incremental learning of embeddings on dynamic graphs [20,21], in this paper, we consider a framework that encompasses existing methods for learning embeddings and enables them for a dynamic environment.

Let us consider two conceptually different aspects of *Dynamic Graph Embedding*. The first is a naive one, where each new data batch triggers the computation of a new representation for historical data, completely disregarding previous feature vectors. The second one is an incremental learning paradigm, where both the time and storage costs are reduced by updating embeddings based on new temporal events. Incremental learning algorithms might specify a new objective function and be constrained to the types of problems they could apply to [14]. However, there is also a need for a framework that is able to incorporate the existing works for embeddings and implement them in an incremental

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail address: tomasz.kajdanowicz@pwr.edu.pl (T. Kajdanowicz).

https://doi.org/10.1016/j.knosys.2021.107453
0950-7051/

List of abbreviations and acronyms

AUC	Area under the ROC Curve
CTDNE	Continuous-Time Dynamic Network Embedding
DGI	Deep Graph Infomax
EC	Edge Classification
FILDNE	Framework for Incremental Learning of Dynamic Networks Embedding
GAN	Generative Adversarial networks
GCN	Graph Convolutional Network
GR	Graph Reconstruction
GLODYNE	Global Topology Preserving Dynamic Network Embedding
HOPE	High Order Proximity preserved Embedding
LE	Laplacian Eigenmaps
LINE	Large-scale Information Network Embedding
LLE	Locally Linear Embedding
LP	Link Prediction
LR	Logistic Regression
MAP	mean Average Precision
MF	Matrix Factorization
n2v	Node2vec
Online-n2v	Online-Node2vec
RNN	Recurrent Neural Network
SVDX	Singular Value Decomposition

manner, allowing for time and space costs to be reduced while retaining the quality of the base method.

This work. In this paper, we propose FILDNE, for incremental learning of embeddings of a dynamic network. Our contribution can be summarized as follows:

1. We provide a method that utilizes historical embeddings and incrementally enhances them based on batched event stream (*non-overlapping snapshots* for keeping the embeddings current; see Fig. 2). We consider two variants: (a) the first one, FILDNE, recursively combines a pair of embeddings at each step using a hyper-parameter to steer the importance weighting (see Fig. 1; (b) the second one, k-FILDNE, combines a vector of k embeddings at once using an *automatic estimation method for importance weighting parameters*.
2. The proposed method can work with *any* graph embedding method, including static and temporal graph embedding methods. Using a novel *reference nodes selection scheme*, our method performs an *embedding alignment* step that allows us to apply convex combination despite rotations and translations of embedding spaces. Moreover, our framework is designed to work in an *unsupervised manner* that does not require obtaining any additional class labels.
3. Through comprehensive empirical analyses that include link prediction, edge classification, and graph reconstruction tasks, we demonstrate that FILDNE allows *reducing memory and computational time costs* while being competitive when compared to other streaming methods in terms of embedding quality. Our hyper-parameter sensitivity study (see Fig. 10) shows how the balance of importance of past and recent events (data batches) influences the performance of the representation.

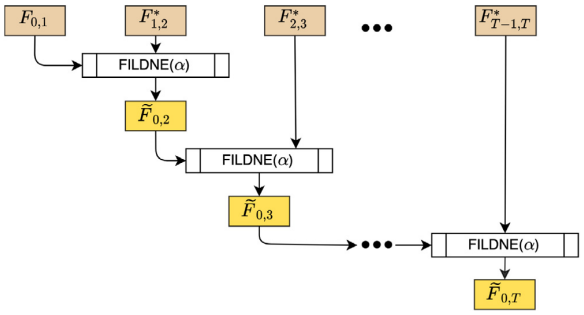


Fig. 1. The FILDNE method applied on graph stream. At the beginning, FILDNE composes of two embeddings computed by the Base embedding method. Next, with each new snapshot, FILDNE composes the aligned version of the current embedding $F^*_{t-1,t}$ with the output of the previous iteration $\widetilde{F}_{0,t}$.

The paper is organized as follows: in Section 2, we present an overview of the related work in the domain, whereas in Section 3, we introduce several formal definitions and notation. Next, in Section 4, we propose our Framework for Incremental Learning of Dynamic Networks Embeddings and provide its detailed description. Further, in Section 5 we report the results of our extensive experiments. Section 6 concludes our work and outlines the directions for future work.

2. Related work

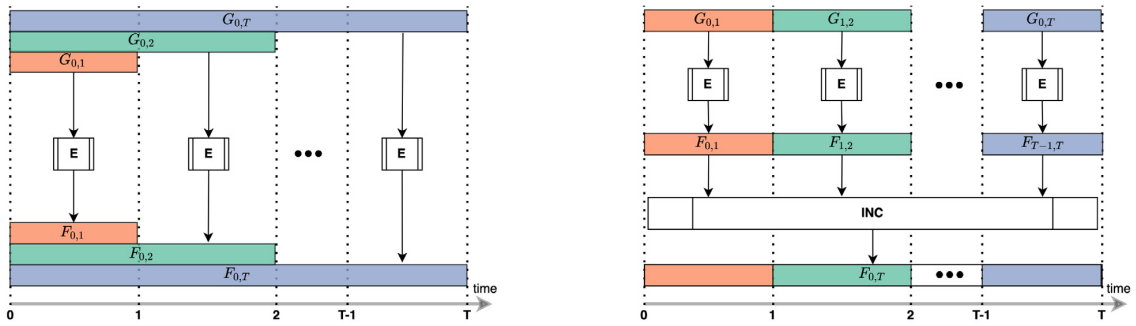
The network embedding problem has attracted much attention from the research community worldwide in recent years. Plenty of methods have been developed, each focused on a different aspect of network embedding, such as proximity, structure, attributes, learning paradigm, scalability, to name only a few [14, 15,22]. In this section, we discuss several methods that are relevant to the scope of our paper. We summarize these in Table 1, where we adapt the taxonomy introduced in [15].

2.1. Static network embedding

The topic is covered in various embedding method families, among which we first discuss *matrix factorization* based methods. **Locally Linear Embedding (LLE)** [23] and **Laplacian Eigenmaps (LE)** [24] both aim to map a high-dimensional data point space to low-dimensional one based on the neighborhood of points (first-order proximity). LLE reconstructs a linear weight matrix, and in LE eigenvectors over graph Laplacian are computed. **Large-scale Information Network Embedding (LINE)** [1] extended these approaches by additionally preserving second-order proximities in the graph. Nodes with similar neighborhoods end up lying closer in the embedding space. **High Order Proximity preserved Embedding (HOPE)** [25] aims to sustain asymmetric proximities of nodes in the graph, in contrast to LINE, where proximities were symmetric.

The next group is methods which are based on *random-walks*. In **DeepWalk** [26] random-walks sampled over the graph are fed to the skip-gram model adapted from Word2vec [27]. **Node2vec (n2v)** [2] was an improvement over DeepWalk, where authors introduced parameters p and q control both depth-first search and breadth-first search like behavior.

Another approach utilizes *Graph Convolutional Networks* architecture. **Deep Graph Infomax (DGI)** [28] is an unsupervised method, which relies on maximizing mutual information between patch representations (obtained by GCN-based encoder layer) and corresponding high-level summaries of graphs (obtained by readout function). All of the approaches mentioned above are capable of processing static networks only.



(a) Using a static or temporal embedding method (**E**) requires to fully retrain the method on every new batch using cumulative snapshots.

(b) Using an incremental embedding method (**INC**) it is possible to reuse the already computed embeddings of non-overlapping snapshots and combine those into an embedding that describes the full event stream history.

Fig. 2. Comparison of applications of traditional static network embedding methods and incremental methods in dynamic network embedding task.

2.2. Temporal and dynamic network embedding

In Temporal Network Embedding methods, we aim to preserve the network's temporal properties. On the other hand, Dynamic Network Embedding focuses on providing up-to-date embedding for evolving graphs. We can distinguish *online* approaches that update embedding with every new edge arrival and *incremental* approaches that process events in batches. The majority of the methods satisfy both of these objectives — temporal and dynamic. Here we give a brief overview of the most prominent ones. A popular trend found in the literature is to build upon the framework of random walks with the skip-gram model.

Continuous-Time Dynamic Network Embedding (CTDNE) [29] introduced temporal walks that traverse edges according to their timestamps instead of performing random-walks statically. The original version of the method focused on the temporal aspect of embedding, while in the follow-up work, the authors introduce an online version of the algorithm [30] that produces a new portion of temporal random walk for upcoming events and then updates the model.

This direction is further extended in **Dynnode2vec** [20] architecture, where for each timestamp, a random-walk is sampled, only for nodes marked as evolving in terms of new edges. **Global Topology Preserving Dynamic Network Embedding (GloDyNE)** [31] also follows a similar schema with incrementally updating skip-gram model, but they differ in the method of selecting nodes to perform new random-walks. They partition the graph into k sub-networks for each timestamp, and for each sub-network, they randomly select one node based on calculated probability distribution within the sub-network. After obtaining the representative nodes list, they perform new random-walks over a new snapshot for representative nodes and update the skip-gram model.

An interesting approach was introduced in **Online-Node2vec** models [32]: StreamWalk and SecondOrder. StreamWalk follows a temporal random walk procedure like CTDNE but differs in the edge sampling scheme. In their second model SecondOrder, they use MinHash fingerprinting to approximate Jaccard node similarity instead of performing temporal random-walks that reduce the method's complexity.

In **Weg2vec** [33], instead of embedding nodes, representations of events are learned by introducing a weighted neighborhood edge sampling strategy.

Finally, in **tNodeEmbed** [34] for each timestamp a new embedding is calculated using the Node2vec method. Embeddings are aligned between timestamps before they are fed as an input to the Long Short Term Memory (LSTM) model, which performs an end-to-end task.

In contrary to random-walk-based methods **Dyngraph2vec** [35] extends Autoencoder (AE) architecture to capture to the evolving structure of temporal networks providing a purely neural-network-based approach. They present three variants of the model: dyngraphAE, dyngraphRNN, and dyngraphAERNN, which differ in how they represent input neighbor vector — dyngraphAE uses fully connected layers, dyngraphRNN uses LSTM layers, and DyngraphAERNN uses fully connected layers followed by LSTM layers.

The **EvolveGCN** [36] paper utilizes Graph Convolutional Networks applied on each timestamp in the graph. The GCN layer weights from one timestamp are passed through a Recurrent Neural Network (GRU, LSTM) to obtain new weights in the next timestamp.

A completely different approach is proposed in the **DGNN** [37] paper. The authors consider single additions of edges, marking the nodes of this edge as *interacting* and their neighbors as *involved*. Using modified LSTM networks, the method updates node embeddings.

The authors of [38] propose a framework (further referred to as **LCF**) that is based on the linear combination of embeddings from consecutive snapshots. Building upon a similar paradigm, we provide an insightful framework with differences discussed in Section 4.5.

2.3. Network embedding alignment

Network embedding alignment problem arises when combining embeddings from subsequent runs or comparing representations from different graphs. [39,40] identify cross-graph node similarities by jointly solving two optimization problems: they use the Sinkhorn algorithm to match node correspondence and find a linear transformation of one of the embeddings by solving Orthogonal Procrustes. [34] uses Orthogonal Procrustes to see a transformation matrix between embeddings of two consecutive timestamps (as we do). They use all common nodes between timestamps to form the matrix (which differs from our method). [41] uses alignment as an integral part of the model, improving the learning process of embeddings. Embeddings are aligned at anchor nodes (that indicate the same users across two networks) and introduce soft-constraint for non-anchor nodes. Other approaches utilize generative adversarial networks [42,43] to align embeddings. [43] solution is based on Wasserstein GAN to produce cross-lingual embedding mapping. [42] utilized GAN architecture in which they aim to obtain both sides' transformation using cycle consistency loss.

Table 1
Graph node embedding methods comparison. Methods marked in bold are the ones evaluated in our experiments.

Method	Static	Temporal	Dynamic	Network alignment	Taxonomy
LLE'00 [23]	√	×	×	×	MF
LE'03 [24]	√	×	×	×	MF
LINE'15 [1]	√	×	×	×	Edge reconstruction
HOPE'16 [25]	√	×	×	×	MF
DGI'19 [34]	√	×	×	×	GCN
DeepWalk'14 [26]	√	×	×	×	Random walk, Skip-gram
Node2vec'16 [2]	√	×	×	×	Random walk, Skip-gram
GraphSAGE'17 [44]	√	×	×	×	Neighborhood sampling, Aggregation
CTDNE'18, 19 [29,30]	×	√	√	×	Temporal Random walk, Skip-gram
tNodeEmbed'19 [34]	×	√	√	√	RNN, Random walk, Skip-gram
StreamWalk'19 [30]	×	√	√	×	Temporal random walk
SecondOrder'19 [30]	×	√	√	×	Temporal random walk
DynGraph2vec'20 [35]	×	√	√	×	Neural networks
Dynnode2vec'18 [20]	×	√	√	×	Random walk, Skip-gram
Weg2vec'20 [33]	×	√	×	×	Temporal random walk, Skip-gram
GloDyNe'20 [31]	×	√	√	×	Random walk, Skip-gram
LCF'20 [38]	×	√	√	√	Dependent on base method
EvolveGCN'20 [36]	×	√	√	×	GCN, RNN
DGNN'20 [37]	×	√	√	×	RNN
FILDNE'20 (Our)	×	√	√	√	Dependent on base method

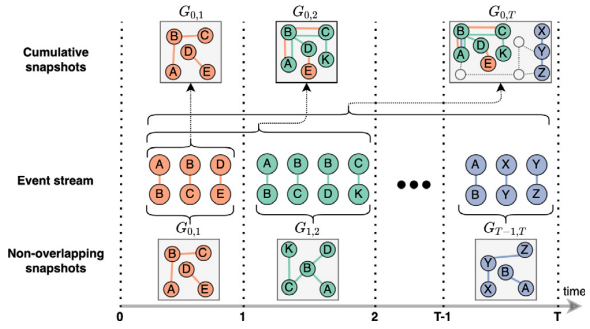


Fig. 3. Event streams can be saved as a series of cumulative or non-overlapping graph snapshots. The first hold the full history from the very beginning, but at the cost of a relatively high memory footprint. Contrary, the latter ones are restricted to a given time interval, hence requiring less memory.

3. Notation and problem definition

The notation introduced in these and all the following is summarized in Table 2.

Definition 1 (*Static Network*)**.** A Network (Graph) is a pair $G = (V, E)$, where V is a set of *vertices* and $E = \{(u, v) : (u, v) \in V \times V\}$, is a set of *edges* connecting vertices. Both, the nodes and edges can possess assigned *attributes*. A special kind of vertices' attributes are *timestamps*, which lead to the next definition.

Definition 2 (*Dynamic Network*)**.** A Dynamic Network (Graph) is a triple $G = (V, E, ts)$ where V and E are sets of vertices and edges respectively and $ts : E \to \mathcal{R}$ is a function assigning *timestamp* to each edge.

Working with such a network is inconvenient — whenever we want to check the state of the graph at a given time t we have to iterate over E. The solution would be to store it as a *snapshot* $G_{0,t} = (V_{0,t}, E_{0,t}, ts)$, where $E_{0,t}$ is a set of edges with timestamps up to time t, while $V_{0,t}$ is the set of vertices associated with them. Further we arrange them in a sequence $[G_{0,1}, G_{0,2}, \ldots, G_{0,T}]$ of cumulative graphs, each associated with a *time-index* in the range $[1; T]$, where T denotes the maximal time-index. One might be interested in non-overlapping batches $[G_{0,1}, G_{1,2}, \ldots, G_{T-1,T}]$, where $G_{t,t+1}$ consists only of edges from $E_{t,t+1}$, created between t and $t + 1$.

For simplicity, we will mark snapshots with the end of interval whenever they cover a single time window, that is $G_{t-1,t} \equiv G_t$.

Dynamic Networks can be attributed in the same way as Static Networks are.

Definition 3 (*Graph Stream*)**.** A Dynamic Network is defined for a limited time interval $[0; T]$ specified by the youngest snapshot's time-index. A Graph Stream expands this definition for a potentially infinite stream of events (each connecting two nodes and represented as an edge) forming an infinite sequence of cumulative graphs $[G_{0,1}, G_{0,2}, \ldots]$ or equivalently non-overlapping ones $[G_{0,1}, G_{1,2}, \ldots]$ (see Fig. 3). The real-world applications of Graph Streams have to take resources limitation into account. Therefore the oldest history has to be forgotten or compressed.

A graph stream can be observed in an online (one edge at a time) or batched manner. In this paper, we would like to focus on the batched setting, remembering that the online setting can be interpreted as single-event batches.

Definition 4 (*Static Network Embedding*)**.** The aim is to find a mapping $f_G : V \to R^d$, $d \ll |V|$ such that the topological (proximity or structural) similarity of vertices in a static G is preserved. The resulting embedding is marked as $F = f_G(V)$ and can be arranged as a $|V| \times d$ matrix, where each row denotes vector representation of a single node.

Definition 5 (*Temporal Network Embedding*)**.** The aim is to find a mapping $f_{G_{0,T}} : V_{0,T} \to R^d$, $d \ll |V|$ such that the temporal topological [45] (proximity or structural) similarity of vertices in a dynamic $G_{0,T}$ is preserved. The resulting embedding is marked as $F_{0,T} = f_{G_{0,T}}(V_{0,T})$ and can be arranged as a $|V_{0,T}| \times d$ matrix, where each row denotes vector representation of a single node.

Definition 6 (*Dynamic Network Embedding*)**.** As the network evolves one may be interested in evolving network embedding. We can distinguish two approaches — naive and incremental one (see Fig. 2). In the latter setting, we reuse previously computed embeddings $(F_{t-1}, f^{t-1}, \ldots, F_1, f^1)$ to obtain a representation $F_{0,t}$ updated with the most recent snapshot G_t, i.e. $F_{0,t} = f^t(G_t, G_{t-1}, F_{t-1}, f^{t-1}, \ldots, F_1, f^1)$. The motivation for incremental paradigm is to reduce computational cost of naive approach by updating nodes' embeddings. The training objective is preservation of the topological properties in $G_{0,t}$.

Table 2
Symbols and notations.

Symbol	Object
G	Network/graph
G_t	Snapshot of dynamic graph with nodes and edges occurring between $t-1$ and t
G_{t_1,t_2}	Snapshot of dynamic graph with nodes and edges occurring between t_1 and t_2
V	Set of nodes
V_{t_1,t_2}	Set of nodes in dynamic graph G_{t_1,t_2}
$V_{t-1\cap t}$	Set of common nodes between G_{t-1} and G_t
V_{ref}	Set of reference nodes
E	Set of edges
E_{t_1,t_2}	Set of edges in dynamic graph G_{t_1,t_2}
T	Maximal time-index in Dynamic Network
F	Embedding matrix
$F(V')$	Embedding matrix of a subset of nodes $V' \subseteq V$
F_t	Embedding matrix for graph $G_{t-1,t}$
F_{t_1,t_2}	Embedding matrix for graph G_{t_1,t_2}
$F^*_{t_1,t_2}$	Aligned embedding for graph G_{t_1,t_2}
$\boldsymbol{F}^*_{k\|t}$	Vector of k aligned embeddings up to time t
$\tilde{F}_{0,t}$	Embedding matrix generated by FILDNE
α	Convex combination weight in FILDNE
$\boldsymbol{\alpha}$	Vector of k convex combinations weights in k-FILDNE
$a(\cdot)$	Activity function
$a_t^{(v)}$	v's activity in G_t
$s(\cdot,\cdot)$	Scoring function
S	Scores of common nodes $V_{t-1\cap t}$

Definition 7 (*Matrix Alignment*)**.** This can be seen as an instance of the *orthogonal Procrustes* problem [40]. Given matrices $A \in R^{n\times d}$ and $B \in R^{n\times d}$ with matching rows, we are interested in finding transformation matrix Q that satisfies

$$\underset{Q:Q^\intercal Q=I}{\operatorname{argmin}} \|BQ - A\|_2^2. \tag{1}$$

The solution is easily found as $Q^* = UW^\intercal$, where $U\Sigma W^\intercal$ is the Singular Value Decomposition (SVD) of $B^\intercal A$.

Remark. For the completeness of the discussion on evolving networks, one should also consider such aspects as time attributes in the form of intervals, non-time attributes changing in time, and nodes appearing without an edge. However, these considerations go beyond the proposed method's scope, and we leave them as challenges for subsequent research.

4. Proposed framework for incremental learning of dynamic networks embeddings

The goal of Framework for Incremental Learning of Dynamic Networks Embeddings (FILDNE) is to find the embedding $F_{0,t}$ of the full graph $G_{0,t}$, without calculating it from all source data from period $[0, t]$, but based on already computed embeddings on historical data, i.e. $(F_{t-1}, \ldots, F_1)$ and the most recent snapshots G_{t-1} and G_t. Let us note that our method does not require the mapping functions $(f^{t-1}, \ldots, f^1)$. FILDNE consists of 3 consecutive steps that are repeated with each new data portion arrival $G_{t-1,t}$. The methods come in two versions – FILDNE and k-FILDNE – therefore, step 3. has two variants. The difference between FILDNE and k-FILDNE method is that the former works in a pairwise manner $F_{0,t} = \text{FILDNE}(G_t, G_{t-1}, F_{t-1})$, while the latter operates on a vector of past embeddings $F_{0,t} = \text{k-FILDNE}(G_t, G_{t-1}, F_{t-1}, \ldots, F_{t-k})$, where k is a parameter of the method.

Step 1. Batch embedding. For each new snapshot $G_{t-1,t} \equiv G_t$, we apply some network embedding technique – called *Base Method* – and obtain $F_{t-1,t} \equiv F_t$ (see Fig. 4a). The choice of the Base Method is entirely up to the user. It can be an instance of Static Network Embedding (e.g. node2vec, LINE, DGI) and Temporal Network Embedding (e.g. CTDNE), both handling additional node/edge attributes if such are observed.

We focus on transductive Base methods, for which inferring representation for unseen examples is not possible in any other way than full retraining of the model. One could also use inductive graph embedding methods (such as GraphSAGE or GCN-based approaches), but these do not suit our problem setting (in fact, our framework provides a way to inductive learning itself).

Step 2. Alignment. The random initialization and stochastic optimization adopted in embedding methods result in non-comparable realizations of representation vectors even for the same input data. The same applies when matching embeddings between two timesteps, forcing us to run network embedding alignment before we go to the next step. The problem appears to be straightforward — arranging embeddings in matrices with matching rows lets us apply matrix alignment techniques. However, the problem's nature to be solved should not be forgotten. Network evolution is not only about the appearance of new nodes but also about the change in neighborhoods of existing ones. We are not interested in vertices whose structural neighborhood topology was completely altered. Therefore, not all vectors should be considered when aligning embedding matrices. We introduce the concept of *reference nodes* (later discussed in Section 4.1) or anchor nodes [41] marked as V_{ref}. They are supposed to be (relatively) static over two neighboring batches. Having such a reference we can calculate the transformation matrix Q(as described in Definition 7) only on vectors representing those nodes and then apply in on the entire embedding matrix F_t what results in F_t^* (see Algorithm 1 and Fig. 4c).

In case of the k-FILDNE method F_t is aligned to the previous embedding F^*_{t-1} which has been aligned in the former iteration. We store a vector of k aligned embeddings $\boldsymbol{F}^*_{k|t}$.

Algorithm 1: Embedding Alignment

Input: $F_{t-1}, F_t, \{(a_{t-1}^{(v)}, a_t^{(v)}) : v \in V_{t-1\cap t}\}$, select$(\cdot,\cdot)$
$S = \{s(a_{t-1}^{(v)}, a_t^{(v)}) : v \in V_{t-1\cap t}\}$ ▷ Calculate node scores
$V_{\text{ref}} = \text{select}(S, V_{t-1\cap t})$ ▷ Obtain reference nodes
$U\Sigma W^\intercal = \text{SVD}(F_{t-1}(V_{\text{ref}})^\intercal F_t(V_{\text{ref}}))$
$Q = UW^\intercal$ ▷ Compute transformation matrix
return $\underline{F_tQ}$

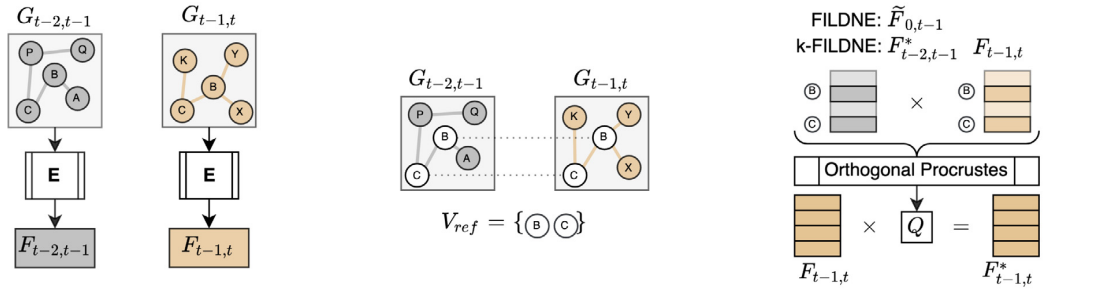
Step 3. Embedding composition. In the FILDNE method (see Algorithm 2 and Fig. 4d), at each iteration, we combine the previously composed embedding $\tilde{F}_{0,t-1}$ with its aligned embedding of the current snapshot $F^*_{t-1,t}$ using following *convex combination*:

$$\tilde{F}_{0,t} = \alpha\tilde{F}_{0,t-1} + (1-\alpha)F^*_{t-1,t} \tag{2}$$

At the beginning ($t = 2$) we use $\tilde{F}_{0,t-1} = F_{0,1}$. The α parameter can be selected using search methods or experts' knowledge. Whenever $\alpha > 0.5$, it means that the past embedding is more important than the recent data increment in the Graph Stream.

Algorithm 2: FILDNE

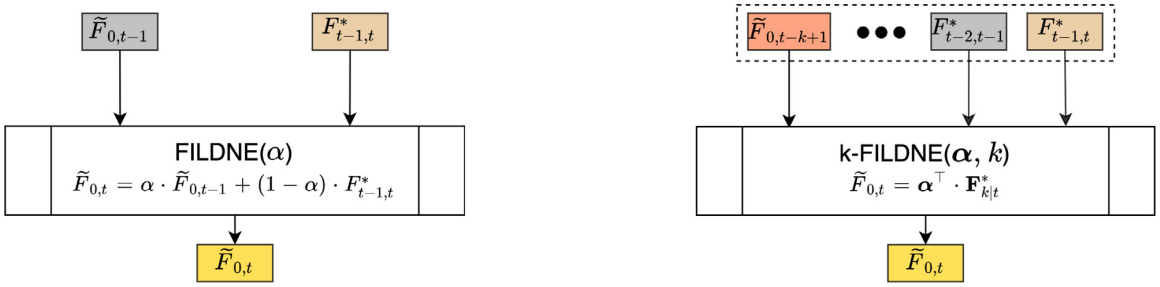
Input: $\tilde{F}_{0,t-1}, F_t, \{(a_{t-1}^{(v)}, a_t^{(v)}) : v \in V_{t-1\cap t}\}$, select$(\cdot,\cdot)$, α
$F_t^* = \text{Embedding Alignment}(\tilde{F}_{0,t-1}, F_t, a_{t-1}, a_t, \text{select})$
$\tilde{F}_{0,t} = \alpha\tilde{F}_{0,t-1} + (1-\alpha)F_t^*$
return $\underline{\tilde{F}_{0,t}}$



(a) Batch embedding. Using Base method **E**, the first step is to compute the embedding $F_{t-1,t}$ for the most recent non-overlapping snapshot.

(b) Reference node selection. Based on node activities $a(\cdot)$ from subsequent snapshots, a scoring function $s(\cdot,\cdot)$ is applied. A given node selection scheme uses those node scores, to obtain the reference nodes set V_{ref}.

(c) Embedding alignment. Solving the Orthogonal Procrustes problem for embeddings of reference nodes allows to obtain a transformation matrix Q, which is multiplied by $F_{t-1,t}$. FILDNE uses matrix $F_{0,t-1}(V_{ref})$ and k-FILDNE $F_{0,t-1}(V_{ref})$



(d) Single composition step of FILDNE. The previously composed embedding matrix $\widetilde{F}_{0,t-1}$ is combined with the aligned embedding matrix $F^*_{t-1,t}$ using parameter α.

(e) Single composition step of k-FILDNE. The algorithm obtains the embedding $\widetilde{F}_{0,t}$ by computing the dot product between the parameter vector $\boldsymbol{\alpha}$ and the vector of k aligned embeddings up to time t $\boldsymbol{F}^*_{k|t}$.

Fig. 4. Overview of the proposed FILDNE algorithm in both variants.

The k-FILDNE version (see Algorithm 3 and Fig. 4e) builds upon two parameters, that is $k \in \mathbb{N}$, the number of last embeddings combined by the algorithm, and $\boldsymbol{\alpha} \in \mathbb{R}^k$, a k-dimensional real-valued vector:

$$\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_k]^\intercal \tag{3}$$

which is constrained to $\{\boldsymbol{\alpha} : 0 \leq \alpha_i \leq 1, \sum_{i=1}^{k} \alpha_i = 1\}$ namely the $k-1$-dimensional simplex. We propose a method to estimate $\boldsymbol{\alpha}$ what is described in Section 4.2. Such construction enables us to combine more than two embeddings at once, in opposite to FILDNE scenario where we always combine 2 embeddings.

Let $\boldsymbol{F}^*_{k|t}$ denote the vector of k embeddings up to time t:

$$\boldsymbol{F}^*_{k|t} = [\tilde{F}_{0,t-k+1}, F^*_{t-k+2}, \ldots, F^*_t]^\intercal. \tag{4}$$

The embedding rule is defined as follows:

$$\tilde{F}_{0,t} = \boldsymbol{\alpha}^\top \cdot \boldsymbol{F}^*_{k|t} \tag{5}$$

which is the dot product of $\boldsymbol{\alpha}$ and the sequence of aligned embeddings $\boldsymbol{F}^*_{k|t}$.

4.1. Reference nodes selection

To select appropriate reference nodes (see Fig. 4b) we first introduce an *activity function*

$$a : V \to \mathbb{R} \tag{6}$$

Algorithm 3: k-FILDNE

Input: $\boldsymbol{F}^*_{k-1|t-1}$, F_t, $\{(a^{(v)}_{t-1}, a^{(v)}_t) : v \in V_{t-1 \cap t}\}$, select$(\cdot,\cdot)$, G_t, prior

F^*_t = Embedding Alignment($F^*_{t-1}, F_t, a_{t-1}, a_t$, select)

$\boldsymbol{F}^*_{k|t} = [\boldsymbol{F}^*_{k-1|t-1}; F^*_t]$

$\boldsymbol{\alpha}$ = Alpha Estimation($\boldsymbol{F}^*_{k|t}, G_t$, prior)

$\tilde{F}_{0,t} = \boldsymbol{\alpha}^\top \cdot \boldsymbol{F}^*_{k|t}$

return $\tilde{F}_{0,t}$

that for each node in the graph G assigns a scalar describing its behavior. In our experiments we use *multi-degree* as the activity function. Activity is measured for common vertices $V_{t-1 \cap t}$ between two neighboring snapshots.

The next step is to obtain a ranking of nodes best suited as a reference. To do so we apply a *scoring function* — in our case:

$$s(a^{(v)}_{t-1}, a^{(v)}_t) = |a^{(v)}_{t-1} - a^{(v)}_t| \left(\frac{\pi}{2} - \arctan(\max\{a^{(v)}_{t-1}, a^{(v)}_t\})\right), \tag{7}$$

where $a^{(v)}_{t-1}$ and $a^{(v)}_t$ are v's activities from neighboring snapshots. The resulting scores are **sorted in ascending order**. Finally, we are able to select a number of reference nodes based on the ranking. We propose the following schemes:

- **Percent** — based on the lowest score the top **p** percent of nodes is selected:

  $$select(S, V) = V_{ref} \subseteq \text{sort}_S(V), \text{ s.t. } |V_{ref}| = \mathbf{p}|V|$$

- **Multiplier** – based on the lowest score the number of nodes is determined as **Md** (multiplier times the embedding size), but no more than $\mathbf{p_{max}}$ (maximum percent) of common vertices:

$$\text{select}(S, V) = V_{ref} \subseteq \text{sort}_S(V), \text{ s.t. } |V_{ref}| = \min(\mathbf{Md}, \mathbf{p_{max}}|V|)$$

- **Threshold** – all nodes with score lower than a given threshold **th** are selected as reference:

$$\text{select}(S, V) = V_{ref} \subseteq V, \text{ s.t. } \forall_{v \in V_{ref}} S^{(v)} \leq \mathbf{th}$$

The methodology of determining reference nodes presented in this section is a generic solution. Other possible approaches include using nodes' attributes or experts' knowledge.

4.2. Alpha estimation

A significant problem arises while using the k-FILDNE model with k parameters. Assuming that each parameter has the same number of considered values $|\Lambda|$, the model has a search space of size $O(|\Lambda|^k)$, i.e., it grows exponentially with each new dimension. Hence, we need to find a way to estimate the model parameters using an algorithm that is cheaper than a full search over the entire parameter space. We propose an algorithm that uses Bayesian inference with assumption of Dirichlet-Multinomial distribution (see Algorithm 4 and Fig. 5). We want to estimate the parameters $\hat{\boldsymbol{\alpha}} = \{\hat{\alpha}_1, \ldots, \hat{\alpha}_k\}$. For the prior we use the Dirichlet distribution:

$$Dir(\boldsymbol{\alpha}|\boldsymbol{\beta}) = \frac{1}{B(\boldsymbol{\beta})} \prod_{i=1}^{k} \alpha_i^{\beta_i - 1} \mathbb{1}_{\{S_{k-1}\}}(\boldsymbol{\alpha}), \tag{8}$$

where $B(\cdot)$ is the beta function used for normalization purposes. $\boldsymbol{\beta}$ reflects the prior knowledge about the distribution and S_{k-1} denotes the $k-1$-dimensional simplex. We define two settings – *uniform*: $\beta_1 = \beta_2 = \cdots = \beta_k = 1$, where representations are equally important, and *increasing*: $\beta_1 < \beta_2 < \cdots < \beta_k$, where more recent embeddings are assumed to be more significant.

The likelihood function has the form:

$$p(\boldsymbol{D}|\boldsymbol{\alpha}) = \prod_{i=1}^{k} \alpha_i^{N_i} \tag{9}$$

where $\boldsymbol{D} = [N_1, N_2, \ldots, N_k]$ is the vector of class occurrences from the link prediction experiment described below. Note that $N = \sum_{i=1}^{k} N_i$ is the total size of the sample.

Each of the embeddings $\tilde{F}_{0,t-k+1}, F^*_{t-k+2}, \ldots, F^*_t$ (see Eq. (4)) is a separate class in the Multinomial distribution. We take edges from the most recent snapshot G_t and split them into the train and test sets. Negative edges are sampled in both groups in numbers enabling class balance. We fit a set of Logistic Regression classifiers with input vectors for each edge, built as Hadamard product of node embeddings, and the outputs denoting edge existence. If a link was correctly predicted with several embeddings, we randomly choose only a single representation. If none of them can provide the correct classification of the link, such an edge is removed from the sample. The class counts (correct predictions) $[N_1, N_2, \ldots, N_k]$ are measured on the test set.

Using the above assumptions we estimate the parameters $\hat{\boldsymbol{\alpha}}$ as the maximum a posteriori probability of the Dirichlet-Multinomial model [46]:

$$\hat{\alpha}_j = \frac{N_j + \beta_j - 1}{N + \sum_{i=1}^{k} \beta_i - k} \tag{10}$$

Algorithm 4: Alpha Estimation

Input: $\boldsymbol{F}^*_{k|t}$, G_t, prior

1 Generate link prediction dataset over graph G_t
2 Fit Logistic Regression classifiers on train set
3 Evaluate link prediction on test set and report correct predictions $[N_1, N_2, \ldots, N_k]$
4 Set $\boldsymbol{\beta}$ according to *prior* distribution
5 $\hat{\boldsymbol{\alpha}} = \left[\frac{N_1 + \beta_1 - 1}{N + \sum_{i=1}^{k} \beta_i - k}, \ldots, \frac{N_k + \beta_k - 1}{N + \sum_{i=1}^{k} \beta_i - k} \right]$
6 **return** $\hat{\boldsymbol{\alpha}}$

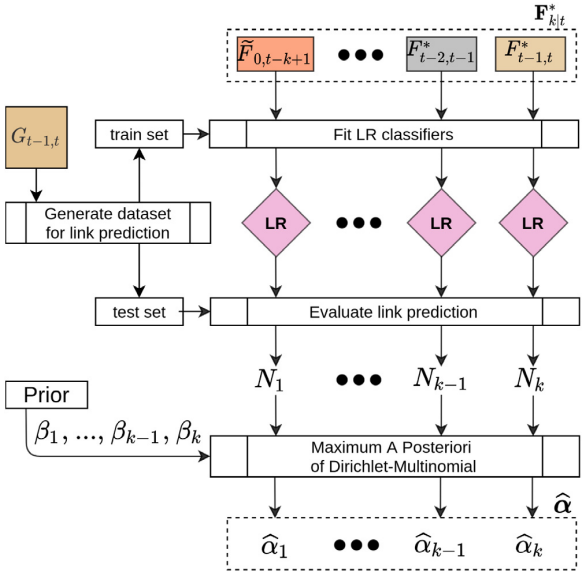


Fig. 5. Alpha estimation. The newly arrived snapshot $G_{t-1,t}$ is prepared as train and test sets for link prediction. For each embedding in the vector $\boldsymbol{F}^*_{k|t}$ a logistic regression (LR) classifier is fitted on the train set and further evaluated on the test set. A poorly performing embedding should yield $\hat{\alpha}_i$ value that barely influence the final representation. This is achieved by Maximum A Posteriori estimation for the Dirichlet-Multinomial model fed with the classification results as described in Section 4.2.

4.3. Missing embeddings: new and inactive nodes

In the FILDNE algorithm, the way we establish the embedding for new or disappearing nodes is trivial. In such a case at time t, we only have one out of two embeddings, and Eq. (2) cannot be applied. For new (previously unseen) nodes v, we use the its embedding from the most recent snapshot as the estimated embedding, i.e. $\tilde{F}_{0,t}(v) = F^*_{t-1,t}(v)$. Contrary, if a node does not appear in most recent graph snapshot, we use its previously estimated embedding, i.e. $\tilde{F}_{0,t}(v) = F^*_{0,t-1}(v)$.

For the k-FILDNE method, the problem is more complex. The number of input embeddings k' varies between 1 and k. If there are all embeddings available, we simply apply Eq. (5), and if $k' = 1$, then we proceed with the same idea as for the FILDNE approach. Otherwise, we take the estimated $\hat{\boldsymbol{\alpha}}$ coefficients (see Section 4.2) corresponding to all k' available embeddings for a given node. Next, we normalize those values, so that they sum up to 1. Then, we apply Eq. (5) assuming that we only have k' components.

4.4. Complexity analysis

To estimate the time and space complexity of our algorithm, we consider the scenario of a single incremental step, i.e. the moment when we have $k-1$ past embeddings $[\tilde{F}_{0,t-k+1}, F^*_{t-k+2}, \ldots, F^*_{t-1}]^\intercal$ and a new graph snapshot G_t appears. We use a random walk based embedding algorithm (Node2vec) and the k-FILDNE version. We denote $O_T(\cdot)$ and $O_S(\cdot)$ as the time and space complexities, respectively. Considering each algorithm step separately, we get:

(1) First, we run Node2vec:

$$O_T(\text{Node2vec}) = O(d|V_{t-1,t}|)$$

$$O_S(\text{Node2vec}) = O(\gamma\omega|E_{t-1,t}|),$$

where γ, ω are the number and length of random walks respectively.

(2) We obtain reference nodes by sorting all vertices $V_{t-1\cap t}$ by their activity score; it takes:

$$O_T(\text{sort}) = O(|V_{t-1\cap t}|log|V_{t-1\cap t}|)$$

$$O_S(\text{sort}) = O(|V_{t-1\cap t}|)$$

Next, we solve the Orthogonal Procrustes (OP) problem during embedding alignment:

$$O_T(\text{OP}) = O(d^3),$$

$$O_S(\text{OP}) = O(d^2).$$

(3a) Link prediction requires sampling of non-existing edges (in the same number as existing edges; Negative Sampling (NS)) takes:

$$O_T(\text{NS}) = O_S(\text{NS}) = O(|E_{t-1,t}|)$$

We train k Logistic Regressions using all embeddings:

$$O_T(\text{LR}) = O(kd^2|E_{t-1,t}|)$$

$$O_S(\text{LR}) = O(kd|E_{t-1,t}|)$$

Finally, we apply LR and aggregate the Class Counts (CC), which takes:

$$O_T(\text{CC}) = O(k * |E_{t-1,t}|)$$

$$O_S(\text{CC}) = O(k)$$

(3b) The estimation of $\boldsymbol{\alpha}$ using Dirichlet-Multinomial model is done by simple division of k numbers, so it takes:

$$O_T(\text{dirichlet}) = O_S(\text{dirichlet}) = O(k)$$

(4) for the composition of all embeddings, we perform a Weighted Matrix Addition (WMA), which takes:

$$\begin{aligned} O_T(\text{WMA}) &= O_S(\text{WMA}) \\ &= O(d|V_{0,t-k+1}| + \cdots + d|V_{t-1,t}|) = O(kd|V|) \end{aligned}$$

where $|V|$ is the number of vertices in the whole graph.

Therefore, the time and memory complexities of FILDNE equals the sum of all above steps' complexities. The size of the last snapshot $V_{t-1,t}$, $E_{t-1,t}$ is smaller than of the whole graph $V_{0,t}$, $E_{0,t}$: $|V_{t-1,t}| < |V_{0,t}| = |V|$, $|E_{t-1,t}| < |E_{0,t}| = |E|$. The total complexity can be approximated as: $O_T(\text{k-FILDNE}) \approx O(|V|(\log|V|+k)+|E|k)$ and $O_S(\text{k-FILDNE}) \approx O(k(|V|+|E|))$. Let us note that the d and k hyper-parameters do not scale with the size of the network. Hence, we can simplify FILDNE's complexity to: $O_T(\text{k-FILDNE}) \approx O(|V|\log|V|+|E|)$ and $O_S(\text{k-FILDNE}) \approx O(|V|+|E|)$.

When considering an iterative application of FILDNE on k snapshots, we can obtain an upper bound by multiplying the estimated complexities by k, i.e., $O_T(k \times \text{k-FILDNE}) \approx O(k^2(|V|+|E|)+k|V|log|V|)$ and $O_S(k \times \text{k-FILDNE}) \approx O(k^2(|V|+|E|))$.

4.5. FILDNE versus other dynamic graph embedding methods

Let us now consider the differences between different approaches of dynamic embedding presented in the literature. First off, we aim at highlighting the differences in data requirements of each approach while computing embedding for new snapshot. The most unfavorable group of methods requires to store whole graph information from the very beginning to time t, i.e. $G_{0,t}$. These are **CTDNE**, **Online-Node2vec**, **tNodeEmbed**, **dyngraph2vec**, **Dynnode2vec**. Additionally, tNodeEmbed method requires all intermediate embeddings $F_{0,1}, \ldots, F_{0,t-1}$. Another group of methods necessitate only the most recent graph snapshots, i.e. G_{t-1}, G_t (**GloDyne**). Our approach, in opposite to all previous, requires only one embedding $\tilde{F}_{0,t-1}$ (**FILDNE**) or k embeddings $\tilde{F}_{0,t-k+1}, \ldots, F^*_{t-2}, F^*_{t-1,t}$ (**k-FILDNE**) and the activity of nodes from G_{t-1}, i.e. $a^{(v)}_{t-1} \forall v \in G_{t-1}$.

Second off, we want to emphasize the difference in the easiness of methods' hyperparameters tuning. Such dynamic network embedding methods as **dyngraph2vec**, **tNodeEmbed** strongly rely on deep network architecture and all related optimization problems. **Online-Node2vec** requires to specify time-dependent hyperparameters that are not very intuitive: time-decay and half-life, which must be set by an expert or through extensive searching. Our approach limits the number of hyperparameters to only two simple and intuitive: the number of reference nodes, combination weight α (FILDNE), or *prior* (k-FILDNE).

Most importantly, third off, our approach allows utilizing embeddings of choice (methods or already calculated vectors) in contrast to all other methods, which rely on specific embedding objectives. It enables our method to use additional network-related data, e.g., nodes' attributes (using, e.g., **DGI**).

The **LCF** method proposed in [38] does not fit to any of the above-mentioned criteria. The authors propose a similar framework to our FILDNE method. However, there are some non-negligible differences. In contrast to them, we employ a convex combination, a special case of the linear combination. Such an approach allows us to keep a nearly constant order of magnitude of embeddings vectors, whereas, for a linear combination with all coefficients equal to one (as proposed in [38]), the magnitude depends on the number of combined embeddings. Further, they introduce an exponential decay based approach where the most recent representation prevails the final embedding. Such an assumption is not flexible, whereas, in our proposed FILDNE methods, the combination parameters α are dynamically estimated from the data. Moreover, our experiments show that the assumption of exponential decay is not satisfied in the real world temporal networks (See Fig. 10). The authors do not neglect that proximity-based embeddings are not comparable across timestamps and therefore apply embedding alignment. However, they do it naively by taking all the available reference nodes between snapshots. Our approach measures nodes' activities' stability to select the most appropriate ones. Finally, the reported results are not compared to the other state-of-the-art Dynamic Graph Embedding methods. Furthermore, not commonly used accuracy metrics makes it impossible to compare with them directly.

5. Experiments

We evaluate the performance of our proposed algorithm in several experiments and compare the results against commonly used baseline methods as well as the Base methods in naive Dynamic Graph Embedding setting. Firstly, we train all methods and use the computed embeddings in a *link prediction* experiment, which should check whether these vector representations encode structural graph information, which can be used to distinguish connected nodes from non-connected ones. Next, for

Table 3
Statistics of graph datasets. |**V**| — no. of nodes, |**E**| — no. of temporal edges (events), **D** — density of the graph, **Directed** — is the graph directed or not. **LP, EC, GR** — dataset was used in **L**ink **P**rediction, **E**dge **C**lassification, **G**raph **R**econstruction tasks, respectively. **Avg. Jaccard Index** of nodes and edges computed as the mean of respective Jaccard Indexes across all consecutive snapshot pairs.

Dataset	$\|V\|$	$\|E\|$	D	Timespan (days)	Directed	Tasks	Embedding size	Avg. Jaccard Index	
								Nodes	Edges
Enron-employees	151	50 572	4.466	1138	×	LP, GR	32	0.864	0.361
Hypertext09	113	20 818	3.290	2	×	LP, GR	32	0.816	0.142
Radoslaw-email	167	82 927	2.991	271	√	LP, GR	32	0.903	0.430
fb-forum	899	33 720	0.084	164	×	LP, GR	128	0.692	0.253
fb-messages	1899	61 734	0.034	216	×	LP, GR	128	0.516	0.093
Bitcoin-alpha	3783	24 186	0.002	1901	√	EC, LP, GR	32	0.217	0
Bitcoin-otc	5881	35 592	0.001	1903	√	EC, LP, GR	128	0.213	0

two datasets with edge labels, we use the same embeddings to perform *edge classification* — we measure the classification quality. We would like to preserve the distances between nodes in the resulting embedding space when learning representations on graphs. Hence, we compute metrics used in *graph reconstruction* (distortion, mAP). Finally, we perform a hyperparameter sensitivity study to show how they influence the model's performance.

5.1. Datasets

We conduct experiments on seven popular dynamic graph datasets downloaded from the Network Repository [47] — we summarize their statistics in Table 3. Selected datasets vary in the total time span between the first and last event (edges): from 2 days (*hypertext09*) up to about 5 years (*bitcoin-otc*). Three of them contain directed edges. Each dataset is used in link prediction and graph reconstruction tasks. Availability of edge labels in *bitcoin-alpha* and *bitcoin-otc* datasets allows us to perform edge classification. In preliminary experiments, we have checked several sizes of node embeddings for all networks and selected the best ones for each one (see Table 3). We note the differences in the characteristics of selected datasets. The two bitcoin networks exhibit a meager amount of intersecting nodes and edges across pairs of consecutive snapshots when compared to the others. The fb-forum, fb-messages, and hypertext09 graphs stay at a moderate level of variability, while enron-employees and radoslaw-email seem stable.

5.2. Experimental setup

In the following paragraphs, we explain how we have configured the experimental environment.

Base methods. We evaluate several state-of-the-art and representative node embedding algorithms from different method families, i.e. based on: random walks (**Node2vec** [2], **CTDNE** [29]), matrix factorization (**HOPE** [25], **LLE** [23], **LE** [24]), graph neural networks (**DGI** [28]) and general function optimization (**LINE** [1]).

These methods do not provide the ability to perform training incrementally. We use these methods to compute two kinds of embeddings:

- baseline embeddings to compare our proposed FILDNE method and other streaming/incremental ones against, i.e., the node representations are computed on the cumulative snapshots $G_{0,t}$ at a given time t;
- embeddings for non-overlapping graphs $G_{t,t+1}$, which are further consumed by our proposed FILDNE method.

Streaming methods. In terms of other incremental methods designed for graph streams embedding, we compare our method to: **tNodeEmbed** [34], two variants of **Online-Node2vec** [30] (**StreamWalk, SecondOrder**) and **dyngraph2vec** [35] (in the **AE-RNN** version).

Convergence issues. During the experiments, we found out that the LLE [23] method did not converge on three datasets (fb-messages, bitcoin-alpha, bitcoin-otc). We checked several hyperparameter settings of the underlying optimizer, but none of them fixed this issue. We decided not to report the results of this method for these datasets.

Data preprocessing. For each dataset, we apply the following preprocessing steps: (1) we take all edges and sort them by time in an ascending manner, (2) we split these edges into ten equally sized parts according to time, (3) we convert each edges chunk into a graph snapshot and call it $G_{t,t+1}$, where $t \in \{0, \ldots, 9\}$. In total, we obtain *10 non-overlapping graph snapshots*. We also save cumulative graphs which accumulate all edges from the beginning, i.e. $G_{0,t}$, which contain all edges from snapshots $G_{0,1}, G_{1,2}, \ldots, G_{t-1,t}$.

Note that tNodeEmbed updates its internal model for every single timestamp. For the method to be comparable in our experimental scenario, we assume that each of the ten snapshots, as mentioned earlier, is equivalent to a single timestamp that is processed by tNodeEmbed. For this method, we also reuse already calculated Node2vec embeddings.

Embedding calculation. We train all of the above-mentioned methods and obtain two groups of embeddings: (1) *cumulative* $F_{0,t}$ for $t \in \{0, \ldots, 9\}$, which are computed in all $G_{0,t}$ using the *Base* and *Streaming methods*; (2) *non-overlapping* $F_{t,t+1}$ using the Base methods and further combined by our proposed FILDNE methods to obtain *cumulative* embeddings $\tilde{F}_{0,t}$. Unsupervised embeddings (all but tNodeEmbed) are trained once and used in all downstream tasks. tNodeEmbed is a supervised method, and the embeddings are trained for link prediction and edge classification individually. We repeat the training and evaluation procedure 30 times, reporting averaged statistics, to address the random initialization and stochastic optimization used in the methods (e.g., random walk generation in Node2vec and CTDNE).

Mean ranks. For each snapshot and dataset, we establish a ranking of methods based on the average of 30 runs. We summarize it as *mean rank*, which is the average ranking of methods. We report this score in each results table in a separate column. Based on this score, we mark the three best methods in bold.

Fine-tuning. To provide a fair comparison of all methods, we decided to perform a hyperparameter search, with an equal number of 100 iterations, overall methods. We use Tree of Parzen Estimators (TPE) [48] (implemented in the HyperOPT [49] package) choosing most appropriate hyperparameters.

Reproducibility. To allow other researchers and developers to try out our proposed FILDNE model, we make our code available at https://gitlab.com/fildne/fildne. We also publish all experiments in the form of a Data Version Control (DVC) pipeline [50], so they can be easily reproduced.

Table 4
Comparison of *link prediction (AUC [%])* on cumulative graph embeddings $F_{0,t}$, $t \in \{1, \ldots, 9\}$ across all Base methods. The presented values are the mean AUC over 8 snapshots ($G_{2,3}, \ldots, G_{9,10}$) and 30 methods' retrains. We select the 3 best methods (LINE, Node2vec, CTDNE) based on the mean ranks and use those methods in further experiments. Underlined values show the highest AUC score for each dataset individually. Methods that did not converged are marked as "×".

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
LINE	66.08	54.69	79.03	70.75	90.59	72.71	86.74	**1.89**
n2v	65.60	52.82	75.84	65.43	92.28	73.76	90.15	**1.91**
CTDNE	55.58	51.23	65.24	54.35	87.30	67.31	83.32	**3.46**
HOPE	59.16	49.71	53.72	51.16	58.79	53.89	88.71	4.64
DGI	55.53	50.26	54.34	53.09	59.76	57.23	70.88	4.91
LE	47.46	49.73	54.17	51.81	58.23	55.01	77.99	5.07
LLE	×	×	55.69	×	59.96	58.13	54.87	5.44

Table 5
Comparison of FILDNE and other methods in *link prediction task (AUC [%])*. The presented values are the mean AUC scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest AUC score for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
n2v	65.60	52.82	75.84	65.43	92.28	73.76	90.15	**5.43**
LINE	66.08	54.69	79.03	70.75	90.59	72.71	86.74	5.77
CTDNE	55.58	51.23	65.24	54.35	87.30	67.31	83.32	9.61
FILDNE(n2v)	59.25	53.36	73.94	64.27	92.43	71.35	91.67	5.75
FILDNE(LINE)	58.28	62.39	74.11	76.11	92.00	73.87	93.42	**4.21**
FILDNE(CTDNE)	55.39	55.97	66.10	55.86	88.33	65.51	85.64	9.30
k-FILDNE(n2v)	62.22	53.93	75.95	65.09	91.43	71.71	91.66	5.55
k-FILDNE(LINE)	59.41	59.66	75.06	74.83	91.92	73.27	93.62	**4.48**
k-FILDNE(CTDNE)	53.52	53.38	61.24	57.01	87.85	64.44	85.54	10.00
DynGraph2vec(AERNN)	67.88	58.67	70.50	66.93	74.66	66.40	85.72	8.38
TNODEEMBED	68.68	48.40	55.96	60.97	83.57	55.79	76.87	9.61
Online-n2v(streamwalk)	69.59	59.66	73.21	69.55	84.84	68.36	91.13	6.68
Online-n2v(secondorder)	68.70	58.33	69.34	75.31	87.59	72.44	89.41	6.23

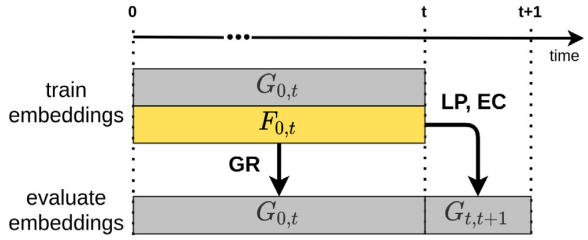


Fig. 6. Evaluation protocol. Link prediction (LP) and edge classification (EC) tasks are evaluated on the **next** snapshot $G_{t,t+1}$, whereas graph reconstruction (GR) task is evaluated on the **corresponding** graph snapshot $G_{0,t}$.

5.3. Link prediction

Setup. For each graph snapshot $G_{t,t+1}$, we generate a link prediction dataset. We split existing edges into a train (75%) and test (25%) dataset and mark as class 1. Then we sample the same number of non-existing edges from the graph (negative samples — class 0).

We implement **next snapshot prediction** evaluation scheme. The embedding methods are trained on cumulative snapshots, i.e. $G_{0,1}, G_{0,2}, \ldots, G_{0,t}$ and on non-overlapping ones, i.e. $G_{0,1}, G_{1,2}, \ldots, G_{t-1,t}$. The latter embeddings are further combined by our proposed FILDNE method to obtain $\tilde{F}_{0,t}$). For each embedding $F_{0,t}$, we evaluate it on the **next** snapshot $G_{t,t+1}$ (see Fig. 6). We follow well established protocol to provide link-prediction by means of classification if an edge exists [2]. Thus, we combine pair of node embeddings into edge features using **Hadamard** operator and feed **logistic regression** classifier.

Note that the Base methods do not provide any mechanism for obtaining embeddings for previously unseen nodes. We modify the sampling procedures mentioned above to consider edges (and negative edges) with nodes present in the appropriate embedding matrix.

Results. Experiments on the cumulative snapshots with Base methods are presented in Table 4. We see that LINE and Node2vec achieve the best results on all datasets. Based on the mean rank, we observe that the **3 best methods (LINE, Node2vec, CTDNE)** outperform other methods. Hence, in the other part of this article, we will focus only on these — using them as Base methods for our proposed FILDNE algorithm. Although, if the reader is interested in full results, we provide those in Appendix B.

Based on the mean rank, we see in Table 5 that among the three best methods, there are two embeddings composed with FILDNE. The results clearly show that our method beats other streaming approaches in all cases besides the bitcoin-alpha dataset. Considering the three best methods, we observe that for fb-messages, enron-employees, and hypertext09, there are two FILDNE-based representations, whereas for bitcoin-otc and radoslaw-email, our proposed method is placed in all three of them.

We notice that in the LINE Base method on bitcoin-otc and radoslaw-email, both our FILDNE methods result in high-performance gain compared to the vanilla Base algorithm. For other methods and datasets, we see comparable results. In general, we see that the gap between k-FILDNE and FILDNE is small.

5.4. Edge classification

Setup. This downstream task is defined similarly to the link prediction setup (see Fig. 6); however, the dataset is built differently. We do not need to sample negative instances. We use the bitcoin-alpha and bitcoin-otc graphs, where each edge has a label assigned (besides the timestamp) — it represents the trust

Table 6
Comparison of FILDNE and other methods in *edge classification task (AUC [%])*. The presented values are the mean AUC scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest AUC score for each dataset individually.

	Bitcoin alpha	Bitcoin otc	Mean rank
n2v	65.11	60.92	7.25
LINE	63.09	64.39	6.44
CTDNE	63.81	55.92	8.56
FILDNE(n2v)	61.57	65.47	6.56
FILDNE(LINE)	63.91	66.56	**5.00**
FILDNE(CTDNE)	59.28	57.85	8.56
k-FILDNE(n2v)	64.10	66.23	5.25
k-FILDNE(LINE)	63.92	66.57	**4.94**
k-FILDNE(CTDNE)	58.25	59.84	8.06
DynGraph2vec(AERNN)	56.40	56.14	10.12
tNodeEmbed	71.88	69.32	**3.12**
Online-n2v(streamwalk)	56.73	56.70	9.25
Online-n2v(secondorder)	63.39	53.93	7.88

value of a transaction (values range from −10 to 10). We choose a threshold of 0 to define two classes (values below 0 — class 0, negative trust, untrusted; values equal or higher than 0 — class 1, positive trust). We use the same edge embeddings as in the link prediction task (all methods but tNodeEmbed, which is retrained in a supervised way for this problem explicitly) to train a logistic regression classifier. As the resulting datasets are mostly imbalanced, we set the "class_weight" argument to "balanced" to let the algorithm automatically determine appropriate class weights (we use the Scikit-learn implementation).

Results. We report results of edge classification in Table 6. The tNodeEmbed method outperforms others, but it is re-fitted in the same task, while the others reuse previously trained embeddings. Nevertheless, among the three best results based on the mean rank, two are obtained by FILDNE. For the bitcoin-otc dataset, we see that FILDNE improves the AUC of edge classification for all Base methods. Analogously to link prediction (see Section 5.3), we see that FILDNE and k-FILDNE perform similarly as well as our method beats other streaming approaches (except tNodeEmbed).

5.5. Graph reconstruction

Setup. Contrary to link prediction and edge classification tasks, we *do not use* here the next snapshot prediction (see Fig. 6). The main goal in graph reconstruction problems is to tell how well the embedding represents the graph it was trained on, i.e. a given embedding $F_{0,t}$ is evaluated on its corresponding graph snapshot $G_{0,t}$ (in case of FILDNE we check how well the composed Base method embeddings reflect the original cumulative graph). Graphs are transformed to static ones in order to fulfill the graph reconstruction framework. We use the two metrics [51]:

- a local one — **mAP (mean Average Precision)**, which captures local graph properties; for any node and its embedding vector it checks how many of the nearest vectors (in the sense of euclidean norm) in the embedding space are actually first-order neighbors of this node:

$$mAP = \frac{1}{|V|}\sum_{v\in V}\frac{1}{\deg(v)}\sum_{i=1}^{|N_v|}\frac{|N_v \cap R_{v,w_i}|}{|R_{v,w_i}|}, \tag{11}$$

  where $\deg(v)$ denotes the degree of v, $N_v = \{w_1, w_2, \ldots, w_{\deg(v)}\}$ — neighborhood of v, R_{v,w_i} — is the smallest set of such points that contains w_i (that is, R_{v,w_i} is the smallest set of nearest points in the embedding space required to retrieve the ith neighbor of v).

- a global one — **distortion**, which compares the distances in the embedding space (euclidean norm) with the distances in the graph (shortest path lengths), the embedding distances are normalized to be within the range $[1; n]$, where n is the longest shortest path length (also called *diameter* of the graph).

$$D = \frac{1}{\binom{n}{2}}\left(\sum_{u,v\in U: u\neq v}\frac{|d_E(u,v) - d_G(u,v)|}{d_G(u,v)}\right), \tag{12}$$

  where n is the number of nodes, d_G are the graph distances and d_E are the embedding distances.

Note that we are interested in higher mAP values and lower distortion values (with 100% and 0 as the ideal values, respectively). We also do not need an auxiliary classifier, like logistic regression for link prediction and edge classification. Using a given graph and its respective embedding, we compute the metrics.

Results. Table 7 summarizes the graph reconstruction task results as the mean Average Precision. Considering the mean ranks, we see both FILDNE and k-FILDNE models in the top three methods. One of them (k-FILDNE with Node2vec embeddings) achieves the best results for fb-forum, improving the pure Node2vec by approximately 10% percentage points. Overall, Node2vec significantly outperforms other methods on all the remaining datasets. We observe that the other streaming approaches perform poorly — they allocate themselves at the three last positions in the mean ranks.

We also examine distortion as a graph reconstruction measure (see Table 8) for which our proposed method improves the results of Base methods in most cases, or it stays at a comparable level. Contrary to mAP, the competitive streaming methods occupy two of the top three positions alongside FILDNE. Moreover, our method achieves the best performance on fb-forum and hypertext09 datasets.

5.6. Time and memory costs

Setup. In this experiment, we measured the time of computing embeddings using all of the considered methods, i.e.:

- for Base and streaming methods it is only the time needed to compute the embedding of the graph $G_{0,t}$ (either in batched or streaming manner);
- for both FILDNE and k-FILDNE we sum up the time of computing: node activities in graph $G_{t-1,t}$, embedding $F_{t-1,t}$ using a given Base method, the calibration procedure of this new embedding to previous ones, alpha estimation (for the k-FILDNE) and the time of applying the FILDNE composition equation.

We also measure the peak (highest) memory consumption of the procedures mentioned above. Note that this also includes the reading of the graphs into memory, as well as previously saved models (for streaming methods) or calibrated embeddings and node activities (for both FILDNE methods).

The time measurements are performed while computing embedding in the main pipeline, so we have 30 measurements for each scenario. Meanwhile, the memory measurements are done in a separate branch of the pipeline, as it requires probing the current memory usage with a relatively high frequency to obtain accurate results. It leads to a massive slowdown of the embedding algorithms, so eventually, we decided to perform five repetitions of these measurements.

For the tNodeEmbed method, we report calculation time and memory utilization measured on the link prediction task.

Table 7
Comparison of FILDNE and other methods in *graph reconstruction task (mAP [%])*. The presented values are the mean mAP scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest mAP score for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	hypertext09	Radoslaw email	Mean rank
n2v	65.99	70.18	23.16	67.44	70.46	55.95	42.12	**1.36**
LINE	10.83	10.81	6.42	9.90	45.35	41.90	39.75	7.52
CTDNE	35.65	47.24	7.85	22.84	55.15	54.80	31.42	4.61
FILDNE(n2v)	28.73	26.40	30.98	30.00	63.90	49.45	40.55	**3.55**
FILDNE(LINE)	14.70	18.02	12.16	7.12	41.71	38.24	38.38	7.61
FILDNE(CTDNE)	18.20	24.27	9.53	11.59	49.39	45.92	36.84	6.30
k-FILDNE(n2v)	33.46	29.30	33.17	34.66	64.21	53.05	40.92	**2.41**
k-FILDNE(LINE)	15.39	25.02	15.58	8.92	44.43	38.82	38.08	6.66
k-FILDNE(CTDNE)	16.85	27.46	13.93	20.39	55.10	48.48	37.26	4.98
DynGraph2vec(AERNN)	0.60	0.66	2.22	1.84	20.46	27.75	26.51	11.32
Online-n2v(streamwalk)	1.08	1.83	2.22	1.25	25.85	28.03	29.60	10.70
Online-n2v(secondorder)	1.38	1.54	1.46	1.28	29.09	27.88	27.11	10.98

Table 8
Comparison of FILDNE and other methods in *graph reconstruction task (distortion)*. The presented values are the mean distortion scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest (best) distortion score for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
n2v	0.81	1.29	0.61	0.93	0.66	0.38	0.57	8.41
LINE	0.87	0.94	0.50	0.59	0.65	0.33	0.80	7.52
CTDNE	0.93	1.70	0.76	0.89	0.81	0.36	0.65	10.12
FILDNE(n2v)	0.70	1.05	0.47	0.74	0.69	0.30	0.54	6.21
FILDNE(LINE)	0.65	0.65	0.41	0.54	0.67	0.31	0.70	5.43
FILDNE(CTDNE)	0.85	1.70	0.65	0.93	0.84	0.32	0.66	9.45
k-FILDNE(n2v)	0.64	1.04	0.44	0.72	0.63	0.30	0.67	5.59
k-FILDNE(LINE)	0.65	0.73	0.33	0.50	0.66	0.31	0.74	**5.05**
k-FILDNE(CTDNE)	0.83	1.65	0.46	0.75	0.68	0.30	0.66	6.84
DynGraph2vec(AERNN)	0.52	0.53	0.44	0.50	0.62	0.40	0.45	**4.20**
Online-n2v(streamwalk)	0.64	0.80	0.52	0.53	0.47	0.39	0.56	5.39
Online − n2v(secondorder)	0.54	0.57	0.48	0.49	0.49	0.40	0.45	**3.79**

Results. From the time measurements reported in Table 9, it is visible that our proposed FILDNE method is the fastest for all datasets. The speedup ratio between the fastest FILDNE configuration and the next fastest competitive method ranges from approximately two to three times. Moreover, k-FILDNE is slower than the Basic version, due to the alpha estimation procedure, described in Section 4.2. However, the actual time difference is not significant — in most cases, k-FILDNE is about 1 second slower, what corresponds to a slowdown of 1% − 10%. In the case of memory utilization (see Table 10), two FILDNE configurations are on average in the top three best-performing methods. It does not mean that our method always improves the Base method. Especially on LINE, we can see no progress nor degradation. The rise of memory consumption is present for the hypertext09 dataset due to the small size of the network. We see that the Online-Node2vec methods are well optimized for memory, contrary to other streaming methods.

5.7. Gain results

5.7.1. Comparison to base methods

For all of the experiments, we compare FILDNE results to the corresponding Base method, computing *gain scores* interpreted as a gain whenever above 1, and as a loss whenever below. To calculate these scores first, we average all 30 retrains of each method for each snapshot. Next, we divide the results of FILDNE by the ones obtained with the corresponding Base method. We compute the mean of those ratios and report it as the gain score (see Fig. 7). Note that we are interested in lower values for distortion, time, and memory consumption, so we take the fraction's reciprocal. Each pair (dataset, Base method) can be viewed from the perspective of link prediction performance, edge classification performance, graph reconstruction quality, memory consumption, and time required for computation. Such a perspective allows for a thorough inspection of our method's properties evaluated on real-world datasets.

In the case of link prediction, edge classification, and distortion, we either achieve comparable results (least gain score of 0.85) or improve the performance by a margin of 75%. We observe the most notable improvements for the calculation time, where we can speed up the computations by a factor of 8 for FILDNE (node2vec) and 7 for k-FILDNE (node2vec). Only for one of the considered cases, the speedup is below 1, i.e., 0.98 for k-FILDNE(ctdne) on radoslaw-email dataset. Memory measurements exhibit similar characteristics to other indicators — FILDNE performs in most cases no worse than Base methods. In the best case, FILDNE allows reducing memory consumption up to 4 times. The maximum loss of about 30%, compared to vanilla ctdne, is encountered with enron-employees, radoslaw-email, and fb-messages, which are small to medium-sized networks.

Results in mean Average Precision exhibit a slightly different nature of the FILDNE method. We see a drop in the performance for some datasets (bitcoin-otc, fb-messages). At the same time, distortion in these cases remains at a decent level. We associate such behavior with applying the embedding alignment step, precisely how we choose reference nodes. As explored by us, the multi-degree activity function promotes choosing nodes with a similar degree but does not consider their neighborhood. Implementing dataset-specific activity and scoring functions could improve the results. However, one has to take into account the complexity of the function (see Fig. 8).

5.7.2. Comparison to streaming methods

We selected Node2vec embeddings as the ones composed by our FILDNE method because three out of four competitive

Table 9
Comparison of FILDNE and other methods in *embeddings calculation time [s]*. The presented values are the mean calculation time over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest calculation time for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
n2v	46.71	133.83	24.94	75.72	4.40	9.52	11.90	7.20
LINE	70.78	63.10	103.97	114.16	114.84	105.17	58.56	11.79
CTDNE	16.25	61.41	3.98	77.44	16.82	4.39	2.65	5.27
FILDNE(n2v)	7.87	22.26	8.44	27.62	2.35	4.79	4.65	**2.75**
FILDNE(LINE)	42.34	41.66	80.53	82.18	84.43	83.29	43.50	9.07
FILDNE(CTDNE)	7.84	18.11	1.91	33.67	8.69	2.63	0.68	**1.98**
k-FILDNE(n2v)	8.61	24.04	9.90	30.69	3.71	5.31	6.40	3.93
k-FILDNE(LINE)	43.10	43.20	81.74	85.02	85.67	83.81	45.09	10.11
k-FILDNE(CTDNE)	8.59	19.88	3.33	36.37	9.96	3.15	2.71	**3.20**
DynGraph2vec(AERNN)	144.71	341.14	25.54	84.10	15.89	13.35	15.47	7.82
tNodeEmbed	52.46	143.07	31.32	84.88	24.96	16.90	24.85	9.48
Online-n2v(streamwalk)	67.76	112.97	27.44	52.09	130.27	12.50	212.59	9.43
Online-n2v(secondorder)	20.78	72.98	70.51	74.15	76.83	20.92	86.89	8.98

Table 10
Comparison of FILDNE and other methods in *max. memory utilization [MB]*. The presented values are the mean max. memory utilization during embeddings calculation over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest memory utilization for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
n2v	580.09	317.12	552.29	360.35	212.27	218.45	227.75	3.71
LINE	917.17	947.08	1291.09	1309.96	1286.30	1277.53	920.00	12.16
CTDNE	1751.63	1262.22	1327.77	1113.66	344.43	320.13	361.23	9.07
FILDNE(n2v)	233.96	344.62	207.13	385.37	206.60	241.49	203.49	**2.23**
FILDNE(LINE)	888.59	899.76	1275.47	1279.96	1271.36	1269.56	889.90	10.45
FILDNE(CTDNE)	436.55	849.80	338.78	1270.43	514.53	384.26	300.85	6.80
k-FILDNE(n2v)	224.82	344.28	225.41	387.52	213.02	241.50	218.23	**2.62**
k-FILDNE(LINE)	888.82	901.10	1276.48	1281.39	1271.71	1269.49	889.21	10.66
k-FILDNE(CTDNE)	437.24	848.13	340.99	1272.02	514.99	385.14	300.77	7.09
DynGraph2vec(AERNN)	1373.12	1934.64	899.86	1023.55	793.38	774.93	816.70	10.20
tNodeEmbed	643.19	1225.00	774.82	827.67	624.12	598.00	618.93	8.54
Online-n2v(streamwalk)	321.29	381.68	269.47	307.71	347.81	249.34	390.14	4.70
Online − n2v(secondorder)	260.86	284.38	258.63	265.96	253.60	246.26	259.93	**2.77**

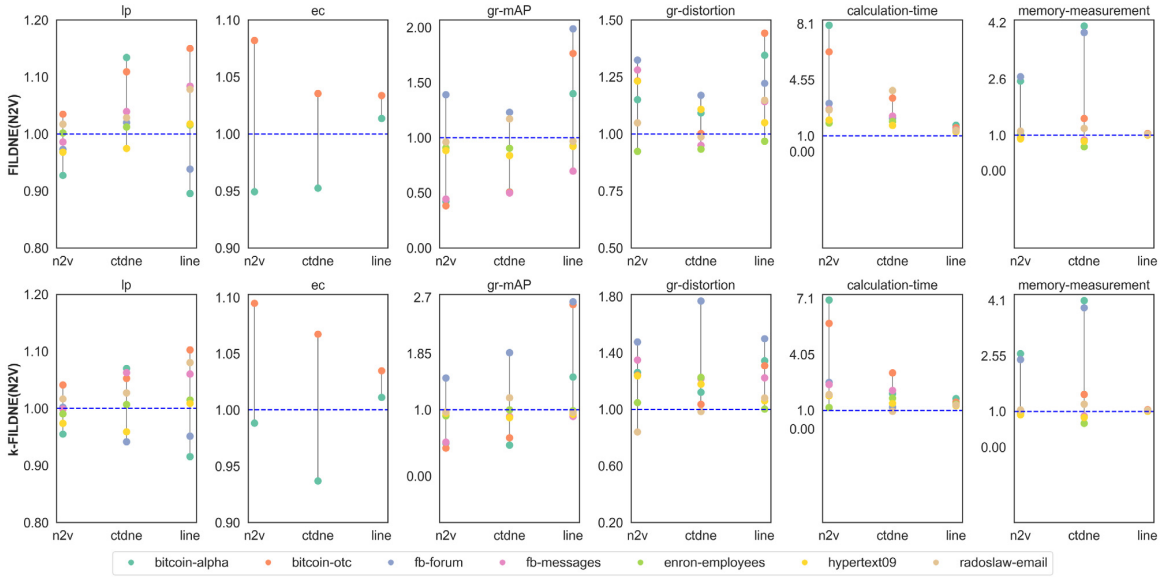


Fig. 7. Mean gain scores for FILDNE and k-FILDNE compared to corresponding Base methods. Points above the dashed line (score equal to 1) indicate a performance gain, whereas points below indicate loss.

streaming approaches are built upon random walks. We compute the gain scores in an analogous way to Section 5.7.1. In terms of link prediction, our proposed FILDNE models achieve comparable results as the other streaming approaches. For edge classification, the tNodeEmbed presents slightly better results than other methods as it is a fine-tuned end-to-end edge classifier. Graph

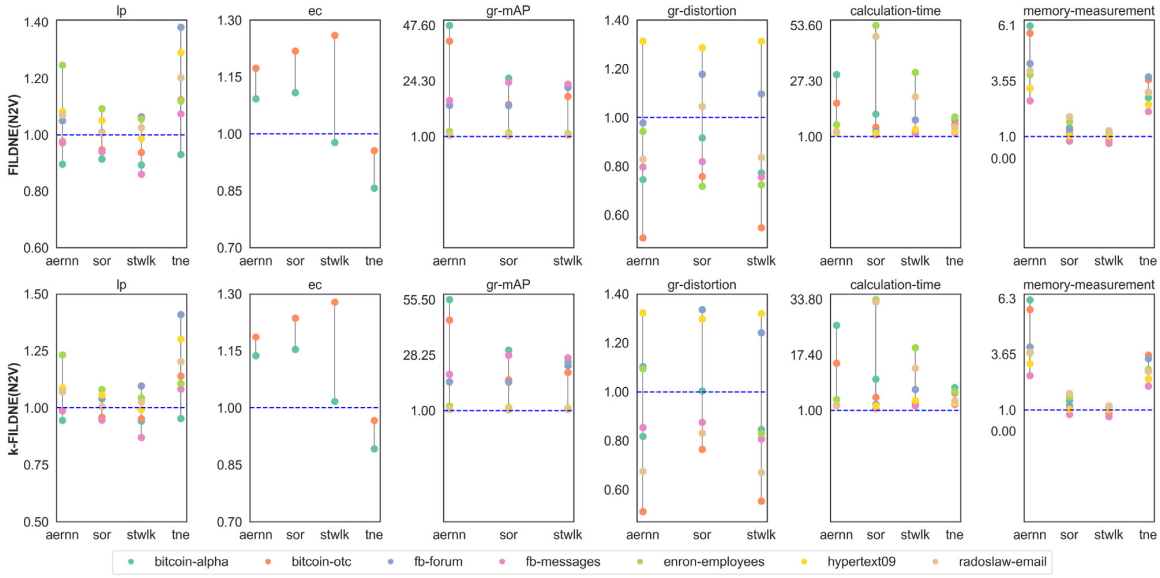


Fig. 8. Mean gain scores for FILDNE and k-FILDNE (with node2vec as the Base method) compared to other streaming methods. Points above the dashed line (score equal to 1) indicate a performance gain, whereas points below indicate loss. Graph reconstruction results are not reported for tNodeEmbed, because the method performs an end-to-end task. We denote tNodeEmbed as `tne`, DynGraph2vec(AERNN) as `aernn` and both Online-Node2vec models as `stwlk` and `sor`.

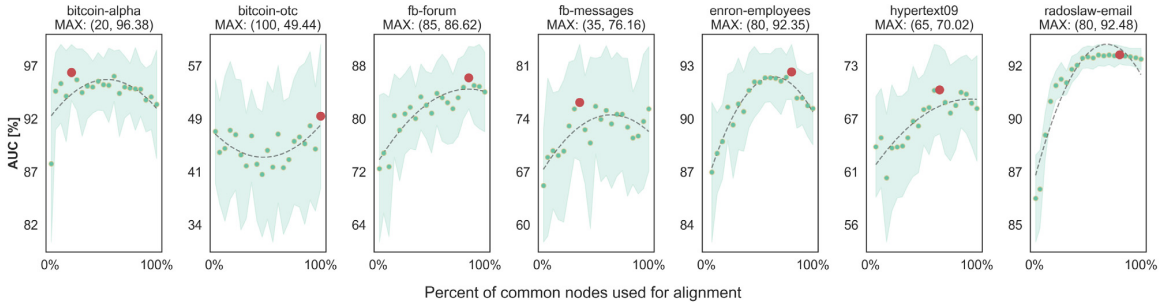


Fig. 9. Link prediction task results (AUC [%]) for different amounts of reference nodes (given as the mean fraction of common nodes between snapshots) The dashed line indicates the trend of these values (fitted using a 2nd order polynomial) and the red dot marks the highest AUC value. Each subtitle presents the name of the dataset and the coordinates of the highest value (percent, AUC).

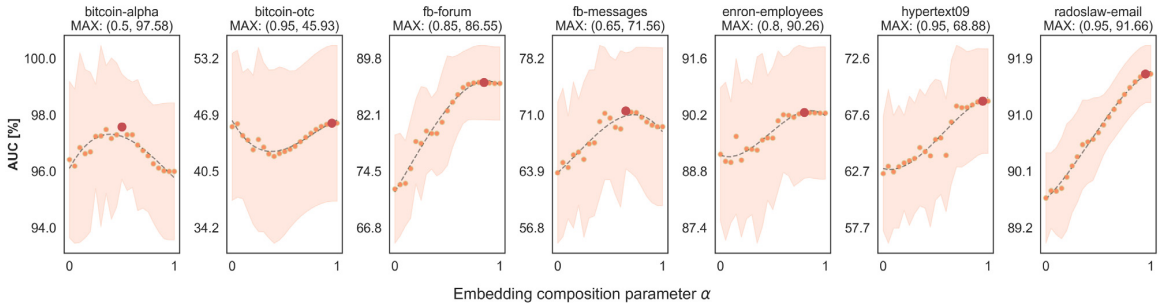


Fig. 10. Link prediction task results (AUC [%]) for different values of α. The dashed line indicates the trend of these values (fitted using a 2nd order polynomial) and the red dot marks the highest AUC value. Each subtitle presents the name of the dataset and the coordinates of the highest value (α, AUC).

reconstruction measured by mean Average Precision shows the superiority of FILDNE by a factor of up to 50. Distortion gain scores vary between 0.55 and 1.3, what confirms FILDNE performance acceptable. We observe that other streaming methods tend to be slow compared to FILDNE method, which is reflected on the time gain scores — embeddings can be computed up to 30 (k-FILDNE) or 50 (FILDNE) times faster. The memory measurements are not surprising — we provide a comparable utilization to both Online-Node2vec models (StreamWalk and SecondOrder). Note that these were explicitly designed to reduce the memory footprint. At the same time, FILDNE significantly outperforms other streaming methods.

Table 11
Comparison of FILDNE and other methods in **link prediction task (AUC)**. The presented values are the mean AUC scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest AUC score for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
n2v	65.60	52.82	75.84	65.43	92.28	73.76	90.15	**6.29**
LINE	66.08	54.69	79.03	70.75	90.59	72.71	86.74	6.48
CTDNE	55.58	51.23	65.24	54.35	87.30	67.31	83.32	11.79
HOPE	59.16	49.71	53.72	51.16	58.79	53.89	88.71	16.32
DGI	55.53	50.26	54.34	53.09	59.76	57.23	70.88	17.36
LE	47.46	49.73	54.17	51.81	58.23	55.01	77.99	18.02
LLE	×	×	55.69	×	59.96	58.13	54.87	19.69
FILDNE(n2v)	59.25	53.36	73.94	64.27	92.43	71.35	91.67	6.98
FILDNE(LINE)	58.28	62.39	74.11	76.11	92.00	73.87	93.42	**4.79**
FILDNE(CTDNE)	55.39	55.97	66.10	55.86	88.33	65.51	85.64	11.00
FILDNE(HOPE)	49.83	50.21	53.57	55.20	59.08	59.37	88.16	15.57
FILDNE(DGI)	53.98	51.48	55.57	53.32	63.56	59.89	68.78	15.77
FILDNE(LE)	49.39	50.59	55.27	49.15	62.12	58.03	80.72	16.79
FILDNE(LLE)	×	×	56.19	×	62.74	58.99	55.53	18.38
k-FILDNE(n2v)	62.22	53.93	75.95	65.09	91.43	71.71	91.66	6.52
k-FILDNE(LINE)	59.41	59.66	75.06	74.83	91.92	73.27	93.62	**5.21**
k-FILDNE(CTDNE)	53.52	53.38	61.24	57.01	87.85	64.44	85.54	12.25
k-FILDNE(HOPE)	53.60	49.50	53.35	54.08	53.30	58.36	78.47	17.39
k-FILDNE(DGI)	51.32	47.97	55.71	53.86	61.75	58.83	66.27	17.23
k-FILDNE(LE)	51.60	50.59	54.12	49.41	61.40	57.98	60.27	17.93
k-FILDNE(LLE)	×	×	55.84	×	61.33	57.73	56.53	19.44
DynGraph2vec(AERNN)	67.88	58.67	70.50	66.93	74.66	66.40	85.72	9.41
tNodeEmbed	68.68	48.40	55.96	60.97	83.57	55.79	76.87	13.93
Online-n2v(streamwalk)	69.59	59.66	73.21	69.55	84.84	68.36	91.13	7.50
Online-n2v(secondorder)	68.70	58.33	69.34	75.31	87.59	72.44	89.41	6.77

Table 12
Comparison of FILDNE and other methods in **edge classification task (AUC)**. The presented values are the mean AUC scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest AUC score for each dataset individually.

	Bitcoin alpha	Bitcoin otc	Mean rank
n2v	65.11	60.92	8.69
LINE	63.09	64.39	7.81
CTDNE	63.81	55.92	10.56
HOPE	57.76	43.04	17.25
DGI	57.58	59.39	12.00
LE	53.60	44.60	16.41
LLE	×	×	×
FILDNE(n2v)	61.57	65.47	7.75
FILDNE(LINE)	63.91	66.56	**6.19**
FILDNE(CTDNE)	59.28	57.85	10.94
FILDNE(HOPE)	58.89	50.77	15.03
FILDNE(DGI)	58.26	52.81	14.38
FILDNE(LE)	52.31	54.63	16.25
FILDNE(LLE)	×	×	×
k-FILDNE(n2v)	64.10	66.23	6.25
k-FILDNE(LINE)	63.92	66.57	**6.12**
k-FILDNE(CTDNE)	58.25	59.84	10.19
k-FILDNE(HOPE)	57.87	49.84	15.00
k-FILDNE(DGI)	58.29	54.61	12.94
k-FILDNE(LE)	47.80	49.87	18.50
k-FILDNE(LLE)	×	×	×
DynGraph2vec(AERNN)	56.40	56.14	13.62
tNodeEmbed	71.88	69.32	**3.62**
Online-n2v(streamwalk)	56.73	56.70	12.81
Online-n2v(secondorder)	63.39	53.93	10.69

5.8. Hyperparameter sensitivity

In this experiment, we will demonstrate the influence of FILDNE 's hyperparameters on downstream link prediction tasks in the last snapshot G_{10}. We use Node2vec as the Base method in all of the following experiments.

Reference node percentage. The first hyperparameter we evaluate is the percentage of reference nodes used in the alignment step. We proposed in Section 4.1 three selection schemes. Here we focus on the percentage scenario (with values: 1%, 5%, 10%, . . . , 95%, 100%). We examine if each dataset has an individual percentage that yields the best results in downstream tasks. We plot the mean AUC values and standard deviations in link prediction tasks for each dataset (see Fig. 9) with k-FILDNE (uniform prior) applied. A single AUC score for a given percentage value is computed as the mean over 30 experiment realizations. Finally, we fit a second-order polynomial (least sq. error) for visualizing the trend on a given dataset and mark in red the point with the highest AUC value.

There is an optimal reference nodes proportion within the exclusive range (0%, 100%) if we observe a concave polynomial approximation. For more complex datasets, e.g., bitcoin-otc, that have a low node and edge Jaccard Index (see Table 3) – representing high dynamics of the network and few common nodes in consecutive windows – it turns out that the best results are for 100% ratio.

FILDNE alpha parameter. Next, we examine the impact of α parameter in FILDNE on AUC, exploring a range from 0 to 1 with a step size of 0.05. The results presented in Fig. 10 for obtained for 30 experiment repetitions. We utilize the best ratio of reference nodes found in the previous experiment.

We hypothesized that we should consider both the historical and recent embedding information while constructing node vector representation. Our intuition is confirmed as we can observe different α for distinct datasets. In fb-forum, fb-messages, enron-employees, hypertext09, radoslaw-email, and bitcoin-otc, the past events are critical for performance. We see that greater α values indicate an increase in the embedding quality to a certain level, above which the performance decreases again. For bitcoin-alpha, we have to keep a subtle balance between the present and the past.

Table 13
Comparison of FILDNE and other methods in **graph reconstruction task (mAP)**. The presented values are the mean mAP scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest mAP score for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
n2v	65.99	70.18	23.16	67.44	70.46	55.95	42.12	**1.84**
LINE	10.83	10.81	6.42	9.90	45.35	41.90	39.75	8.05
CTDNE	35.65	47.24	7.85	22.84	55.15	54.80	31.42	5.25
HOPE	0.51	0.64	2.43	1.39	13.20	28.10	27.06	18.32
DGI	0.66	0.66	2.55	1.45	12.44	27.11	26.91	18.48
LE	0.81	1.12	2.71	1.22	11.89	28.04	76.53	14.00
LLE	×	×	2.65	×	11.98	27.16	48.62	15.38
FILDNE(n2v)	28.73	26.40	30.98	30.00	63.90	49.45	40.55	**4.04**
FILDNE(LINE)	14.70	18.02	12.16	7.12	41.71	38.24	38.38	8.18
FILDNE(CTDNE)	18.20	24.27	9.53	11.59	49.39	45.92	36.84	6.86
FILDNE(HOPE)	0.59	0.67	2.50	1.46	12.90	27.42	27.55	18.27
FILDNE(DGI)	0.77	0.80	2.88	1.68	12.69	27.20	29.66	15.50
FILDNE(LE)	0.76	0.94	2.71	1.29	13.08	27.64	60.00	14.09
FILDNE(LLE)	×	×	2.58	×	11.21	28.12	26.34	19.34
k-FILDNE(n2v)	33.46	29.30	33.17	34.66	64.21	53.05	40.92	**2.89**
k-FILDNE(LINE)	15.39	25.02	15.58	8.92	44.43	38.82	38.08	7.23
k-FILDNE(CTDNE)	16.85	27.46	13.93	20.39	55.10	48.48	37.26	5.52
k-FILDNE(HOPE)	0.59	0.70	2.42	1.48	13.28	29.80	23.92	17.43
k-FILDNE(DGI)	0.93	0.94	3.24	2.06	12.72	27.76	29.04	14.04
k-FILDNE(LE)	0.73	0.90	2.81	1.40	12.84	29.05	41.80	13.79
k-FILDNE(LLE)	×	×	2.98	×	12.58	28.63	31.78	14.91
DynGraph2vec(AERNN)	0.60	0.66	2.22	1.84	20.46	27.75	26.51	17.07
Online-n2v(streamwalk)	1.08	1.83	2.22	1.25	25.85	28.03	29.60	15.20
Online-n2v(secondorder)	1.38	1.54	1.46	1.28	29.09	27.88	27.11	16.04

Table 14
Comparison of FILDNE and other methods in graph reconstruction task (distortion). The presented values are the mean distortion scores over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the highest distortion score for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
n2v	0.81	1.29	0.61	0.93	0.66	0.38	0.57	17.66
LINE	0.87	0.94	0.50	0.59	0.65	0.33	0.80	15.93
CTDNE	0.93	1.70	0.76	0.89	0.81	0.36	0.65	19.41
HOPE	0.58	0.60	0.36	0.48	0.48	0.34	0.44	8.93
DGI	0.51	0.55	0.36	0.61	0.49	0.34	0.38	7.38
LE	0.58	0.58	0.33	0.45	0.47	0.33	0.48	8.43
LLE	×	×	0.54	×	0.77	0.35	0.77	18.91
FILDNE(n2v)	0.70	1.05	0.47	0.74	0.69	0.30	0.54	14.29
FILDNE(LINE)	0.65	0.65	0.41	0.54	0.67	0.31	0.70	12.70
FILDNE(CTDNE)	0.85	1.70	0.65	0.93	0.84	0.32	0.66	18.54
FILDNE(HOPE)	0.56	0.60	0.40	0.48	0.48	0.34	0.42	8.89
FILDNE(DGI)	0.53	0.72	0.30	0.55	0.47	0.33	0.38	7.68
FILDNE(LE)	0.55	0.65	0.28	0.38	0.41	0.34	0.47	**6.96**
FILDNE(LLE)	×	×	0.46	×	0.88	0.35	0.81	19.00
k-FILDNE(n2v)	0.64	1.04	0.44	0.72	0.63	0.30	0.67	12.91
k-FILDNE(LINE)	0.65	0.73	0.33	0.50	0.66	0.31	0.74	11.61
k-FILDNE(CTDNE)	0.83	1.65	0.46	0.75	0.68	0.30	0.66	14.73
k-FILDNE(HOPE)	0.57	0.60	0.43	0.48	0.48	0.32	0.40	8.50
k-FILDNE(DGI)	0.52	0.72	0.30	0.51	0.49	0.33	0.42	**7.30**
k-FILDNE(LE)	0.55	0.61	0.28	0.38	0.43	0.32	0.43	**5.52**
k-FILDNE(LLE)	×	×	0.40	×	0.60	0.33	0.60	11.78
DynGraph2vec(AERNN)	0.52	0.53	0.44	0.50	0.62	0.40	0.45	10.54
Online-n2v(streamwalk)	0.64	0.80	0.52	0.53	0.47	0.39	0.56	13.95
Online-n2v(secondorder)	0.54	0.57	0.48	0.49	0.49	0.40	0.45	10.20

k-FILDNE Prior. For k-FILDNE, we change the *prior* parameter and evaluate both possible values (uniform, increase). We noticed that the downstream task performance's change is not significant – both settings yield similar results. This shows that, after a certain number of observations (internal link prediction), the prior values become irrelevant – the likelihood computed from the actual data becomes more critical (the property that Maximum Likelihood Estimate is equal to Maximum A Posteriori in the limit).

5.9. Summary of the results

We provided an extensive experimental protocol that incorporated various network-related learning tasks, namely link prediction, edge classification, and graph reconstruction. We measured quality in each of those tasks and time/memory consumption on seven benchmark datasets. We can conclude that the experiments' results proved FILDNE superiority by reducing computation time (up to 50x) and memory consumption (up to 6x), achieving the same quality.

Table 15
Comparison of FILDNE and other methods in embeddings calculation time (in seconds). The presented values are the mean calculation time over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest calculation time for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
n2v	46.71	133.83	24.94	75.72	4.40	9.52	11.90	17.91
LINE	70.78	63.10	103.97	114.16	114.84	105.17	58.56	22.50
CTDNE	16.25	61.41	3.98	77.44	16.82	4.39	2.65	15.86
HOPE	3.12	4.93	1.51	2.29	0.81	0.48	1.39	6.09
DGI	3.57	15.83	2.95	5.10	0.43	0.58	0.53	8.46
LE	5.34	13.81	1.71	2.64	0.29	0.25	0.38	5.52
LLE	×	×	4.22	×	0.28	0.25	0.66	7.09
FILDNE(n2v)	7.87	22.26	8.44	27.62	2.35	4.79	4.65	13.29
FILDNE(LINE)	42.34	41.66	80.53	82.18	84.43	83.29	43.50	19.79
FILDNE(CTDNE)	7.84	18.11	1.91	33.67	8.69	2.63	0.68	10.84
FILDNE(HOPE)	1.09	1.52	1.02	1.35	0.34	0.25	0.46	**2.71**
FILDNE(DGI)	0.86	4.59	2.10	2.91	0.34	0.51	0.38	5.09
FILDNE(LE)	1.24	2.01	1.45	1.68	0.25	0.19	0.28	**2.38**
FILDNE(LLE)	×	×	1.81	×	0.16	0.11	0.33	**2.34**
k-FILDNE(n2v)	8.61	24.04	9.90	30.69	3.71	5.31	6.40	14.50
k-FILDNE(LINE)	43.10	43.20	81.74	85.02	85.67	83.81	45.09	20.82
k-FILDNE(CTDNE)	8.59	19.88	3.33	36.37	9.96	3.15	2.71	13.61
k-FILDNE(HOPE)	1.78	3.50	2.24	4.14	1.42	0.75	1.91	8.21
k-FILDNE(DGI)	1.56	5.99	3.15	5.18	1.44	0.98	1.80	9.57
k-FILDNE(LE)	1.94	3.42	2.48	3.96	1.31	0.65	1.67	7.70
k-FILDNE(LLE)	×	×	2.80	×	1.29	0.63	1.74	9.34
DynGraph2vec(AERNN)	144.71	341.14	25.54	84.10	15.89	13.35	15.47	18.54
tNodeEmbed	52.46	143.07	31.32	84.88	24.96	16.90	24.85	20.20
Online-n2v(streamwalk)	67.76	112.97	27.44	52.09	130.27	12.50	212.59	20.14
Online-n2v(secondorder)	20.78	72.98	70.51	74.15	76.83	20.92	86.89	19.70

Table 16
Comparison of FILDNE and other methods in max. memory utilization (in MB). The presented values are the mean max. memory utilization during embeddings calculation over 8 graph snapshots and 30 methods' retrains. We also report the mean ranks of all methods in this experiment. Methods marked in bold are the 3 best methods based on the mean rank. Underlined values show the lowest memory utilization for each dataset individually.

	Bitcoin alpha	Bitcoin otc	fb forum	fb messages	Enron employees	Hypertext09	Radoslaw email	Mean rank
n2v	580.09	317.12	552.29	360.35	212.27	218.45	227.75	8.70
LINE	917.17	947.08	1291.09	1309.96	1286.30	1277.53	920.00	22.75
CTDNE	1751.63	1262.22	1327.77	1113.66	344.43	320.13	361.23	19.18
HOPE	669.68	938.47	300.07	384.13	258.63	253.09	257.59	13.05
DGI	501.36	776.38	418.67	504.65	344.67	338.79	352.69	16.16
LE	538.78	728.63	283.31	329.16	260.49	254.95	273.09	11.89
LLE	×	×	259.21	×	235.45	217.13	258.49	7.56
FILDNE(n2v)	233.96	344.62	207.13	385.37	206.60	241.49	203.49	4.75
FILDNE(LINE)	888.59	899.76	1275.47	1279.96	1271.36	1269.56	889.90	20.96
FILDNE(CTDNE)	436.55	849.80	338.78	1270.43	514.53	384.26	300.85	16.14
FILDNE(HOPE)	268.60	298.96	247.20	268.57	218.52	219.81	220.97	**4.73**
FILDNE(DGI)	342.13	437.32	381.80	401.09	323.55	323.18	328.49	13.36
FILDNE(LE)	247.31	266.87	242.43	247.96	220.95	219.35	220.60	**3.43**
FILDNE(LLE)	×	×	228.67	×	215.67	213.14	216.45	**2.62**
k-FILDNE(n2v)	224.82	344.28	225.41	387.52	213.02	241.50	218.23	5.38
k-FILDNE(LINE)	888.82	901.10	1276.48	1281.39	1271.71	1269.49	889.21	21.18
k-FILDNE(CTDNE)	437.24	848.13	340.99	1272.02	514.99	385.14	300.77	16.43
k-FILDNE(HOPE)	265.25	300.65	260.65	290.92	232.45	228.96	241.79	6.79
k-FILDNE(DGI)	344.94	442.24	392.31	405.24	333.50	329.50	343.56	14.32
k-FILDNE(LE)	252.32	267.62	261.33	265.87	233.62	224.25	238.77	5.61
k-FILDNE(LLE)	×	×	255.57	×	237.24	225.54	239.13	7.47
DynGraph2vec(AERNN)	1373.12	1934.64	899.86	1023.55	793.38	774.93	816.70	20.91
tNodeEmbed	643.19	1225.00	774.82	827.67	624.12	598.00	618.93	19.14
Online-n2v(streamwalk)	321.29	381.68	269.47	307.71	347.81	249.34	390.14	11.59
Online-n2v(secondorder)	260.86	284.38	258.63	265.96	253.60	246.26	259.93	7.61

6. Conclusion and future work

In this paper, we proposed a Framework for Incremental Learning of Dynamic Networks Embedding (FILDNE). It utilizes timestep vectors obtained from any existing node embedding method and produces dynamic representation reducing the computational costs by working on batched data (non-overlapping graph snapshots). We showed experimentally in link prediction, edge classification, and graph reconstruction tasks on seven real-world datasets that FILNDE compared to static, dynamic, and temporal node embedding approaches reduces memory and computational time costs while providing competitive accuracy gains. Moreover, we conducted a hyperparameter sensitivity study and provided insights into how FILDNE's hyperparameters influence the vector representation quality. In terms of future work, we plan to address FILDNE's limitations, namely: (1) it does not provide a mechanism for online learning (update after the single

event) as our method requires batches (graph snapshots); (2) the calibration step requires an overlapping between consecutive snapshots in terms of nodes (there must exist common nodes) — one can expect a new way to calibrate such snapshots using nodes' structural similarity only; (3) we did not explore how our method works in an attributed environment (i.e., using both attributed graphs and embedding method suitable for such data); (4) in our experimental setup we decided to use snapshots of equal size (in terms of time intervals), but it might be required to extend that scenario.

CRediT authorship contribution statement

Piotr Bielak: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Kamil Tagowski:** Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Maciej Falkiewicz:** Conceptualization, Methodology, Software, Validation, Formal analysis, Writing – original draft, Writing – review & editing, Visualization. **Tomasz Kajdanowicz:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Supervision, Funding acquisition. **Nitesh V. Chawla:** Conceptualization, Formal analysis, Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The project was partially supported by The National Science Centre, Poland the research projects no. 2016/21/D/ST6/02948 and 2016/23/B/ST6/01735, by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 691152 (RENOIR); the Polish Ministry of Science and Higher Education fund for supporting internationally co-financed projects in 2016–2019 (agreement no. 3628/H2020/2016/2) and statutory funds of Department of Computational Intelligence.

Appendix A. Source code of used methods

We base our experiments on following methods implementations:

DynGraph2vec — https://github.com/palash1992/DynamicGEM

DGI — https://github.com/PetarV-/DGI

Node2vec — https://github.com/eliorc/node2vec/

HOPE — https://github.com/palash1992/GEM

LLE — https://github.com/palash1992/GEM

LE — https://scikit-learn.org/

Online-Node2vec — https://github.com/ferencberes/online-node2vec

tNodeEmbed — https://github.com/urielsinger/tNodeEmbed

FILDNE — https://gitlab.com/fildne/fildne

Appendix B. All methods results (mean)

We provide comprehensive experimental results for all examined streaming approaches, Base methods and both FILDNE variants applied on these Base methods (see Tables 11–16).

References

[1] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 1067–1077.

[2] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864.

[3] F. Huang, X. Zhang, J. Xu, C. Li, Z. Li, Network embedding by fusing multimodal contents and links, Knowl.-Based Syst. 171 (2019) 44–55.

[4] S. Xu, S. Liu, L. Feng, Manifold graph embedding with structure information propagation for community discovery, Knowl.-Based Syst. 208 (2020) 106448.

[5] W. Nelson, M. Zitnik, B. Wang, J. Leskovec, A. Goldenberg, R. Sharan, To embed or not: network embedding as a paradigm in computational biology, Front. Genet. 10 (2019) 381.

[6] X. Yue, Z. Wang, J. Huang, S. Parthasarathy, S. Moosavinasab, Y. Huang, S.M. Lin, W. Zhang, P. Zhang, H. Sun, Graph embedding on biomedical networks: methods, applications and evaluations, Bioinformatics 36 (4) (2020) 1241–1251.

[7] J. You, B. Liu, Z. Ying, V. Pande, J. Leskovec, Graph convolutional policy network for goal-directed molecular graph generation, in: Advances in Neural Information Processing Systems, 2018, pp. 6410–6421.

[8] C. Zang, F. Wang, MoFlow: an invertible flow model for generating molecular graphs, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 617–626.

[9] D. Xu, C. Wei, P. Peng, Q. Xuan, H. Guo, GE-GAN: A novel deep learning framework for road traffic state estimation, Transp. Res. C 117 (2020) 102635.

[10] Z. Wu, S. Pan, G. Long, J. Jiang, C. Zhang, Graph wavenet for deep spatial-temporal graph modeling, 2019, arXiv preprint arXiv:1906.00121.

[11] Y. Dong, N.V. Chawla, A. Swami, metapath2vec: Scalable representation learning for heterogeneous networks, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 135–144.

[12] Y. Gao, M. Gong, Y. Xie, H. Zhong, Community-oriented attributed network embedding, Knowl.-Based Syst. 193 (2020) 105418.

[13] X. Geng, Y. Li, L. Wang, L. Zhang, Q. Yang, J. Ye, Y. Liu, Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 3656–3663.

[14] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, K. Murphy, Machine learning on graphs: A model and comprehensive taxonomy, 2020, arXiv preprint arXiv:2005.03675.

[15] H. Cai, V.W. Zheng, K.C.C. Chang, A comprehensive survey of graph embedding: Problems, techniques, and applications, IEEE Trans. Knowl. Data Eng. 30 (9) (2018) 1616–1637, http://dx.doi.org/10.1109/TKDE.2018.2807452.

[16] F. Chen, Y.-C. Wang, B. Wang, C.-C.J. Kuo, Graph representation learning: a survey, APSIPA Trans. Signal Inf. Process. 9 (2020) e15, http://dx.doi.org/10.1017/ATSIP.2020.13.

[17] P. Holme, J. Saramäki, Temporal networks, Phys. Rep. 519 (3) (2012) 97–125.

[18] A. Zaki, M. Attia, D. Hegazy, S. Amin, Comprehensive survey on dynamic graph models, Int. J. Adv. Comput. Sci. Appl. 7 (2) (2016) 573–582.

[19] S.M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, P. Poupart, Representation learning for dynamic graphs: A survey, J. Mach. Learn. Res. 21 (70) (2020) 1–73.

[20] S. Mahdavi, S. Khoshraftar, A. An, Dynnode2vec: Scalable dynamic network embedding, in: 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018, pp. 3762–3765.

[21] A. Tsitsulin, M. Munkhoeva, D. Mottin, P. Karras, I. Oseledets, E. Müller, Frede: Linear-space anytime graph embeddings, 2020, arXiv preprint arXiv:2006.04746.

[22] P. Cui, X. Wang, J. Pei, W. Zhu, A survey on network embedding, IEEE Trans. Knowl. Data Eng. 31 (5) (2019) 833–852, http://dx.doi.org/10.1109/TKDE.2018.2849727.

[23] S.T. Roweis, L.K. Saul, Nonlinear dimensionality reduction by locally linear embedding, Science 290 (5500) (2000) 2323–2326.

[24] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Comput. 15 (6) (2003) 1373–1396.

[25] M. Ou, P. Cui, J. Pei, Z. Zhang, W. Zhu, Asymmetric transitivity preserving graph embedding, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Vol. 13-17-August-2016, Association for Computing Machinery, 2016, pp. 1105–1114, http://dx.doi.org/10.1145/2939672.2939751.

[26] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 701–710.

[27] T. Mikolov, K. Chen, G.S. Corrado, J. Dean, Efficient estimation of word representations in vector space, 2013, CoRR abs/1301.3781.

[28] P. Velickovic, W. Fedus, W.L. Hamilton, P. Liò, Y. Bengio, R.D. Hjelm, Deep graph infomax, in: ICLR (Poster), 2019.
[29] G.H. Nguyen, J.B. Lee, R.A. Rossi, N.K. Ahmed, E. Koh, S. Kim, Continuous-Time Dynamic Network Embeddings, Association for Computing Machinery (ACM), 2018, pp. 969–976, http://dx.doi.org/10.1145/3184558.3191526.
[30] J.B. Lee, G. Nguyen, R.A. Rossi, N.K. Ahmed, E. Koh, S. Kim, Dynamic node embeddings from edge streams, 2019.
[31] C. Hou, H. Zhang, S. He, K. Tang, GloDyNE: Global topology preserving dynamic network embedding, 2020, arXiv preprint arXiv:2008.01935.
[32] F. Béres, D.M. Kelen, R. Pálovics, A.A. Benczúr, Node embeddings in dynamic graphs, Appl. Netw. Sci. 4 (1) (2019) http://dx.doi.org/10.1007/s41109-019-0169-5.
[33] M. Torricelli, M. Karsai, L. Gauvin, Weg2vec: Event embedding for temporal networks, Sci. Rep. 10 (1) (2020) 1–11.
[34] U. Singer, I. Guy, K. Radinsky, Node embedding over temporal graphs, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence, AAAI Press, 2019, pp. 4605–4612.
[35] P. Goyal, S.R. Chhetri, A. Canedo, Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning, Knowl.-Based Syst. 187 (2020) 104816.
[36] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T.B. Schardl, C.E. Leiserson, EvolveGCN: Evolving graph convolutional networks for dynamic graphs, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, the Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, the Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 5363–5370, https://aaai.org/ojs/index.php/AAAI/article/view/5984.
[37] Y. Ma, Z. Guo, Z. Ren, J. Tang, D. Yin, Streaming graph neural networks, in: J. Huang, Y. Chang, X. Cheng, J. Kamps, V. Murdock, J. Wen, Y. Liu (Eds.), Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020, ACM, 2020, pp. 719–728, http://dx.doi.org/10.1145/3397271.3401092.
[38] P. Trivedi, A. Büyükçakır, Y. Lin, Y. Qian, D. Jin, D. Koutra, On structural vs. Proximity-based temporal node embeddings, 2020.
[39] E. Grave, A. Joulin, Q. Berthet, Unsupervised alignment of embeddings with wasserstein procrustes, in: The 22nd International Conference on Artificial Intelligence and Statistics, 2019, pp. 1880–1890.
[40] X. Chen, M. Heimann, F. Vahedian, D. Koutra, Consistent network alignment with node embedding, 2020, arXiv preprint arXiv:2005.04725.
[41] L. Liu, W.K. Cheung, X. Li, L. Liao, Aligning users across social networks using network embedding, in: Ijcai, 2016, pp. 1774–1780.
[42] T. Derr, H. Karimi, X. Liu, J. Xu, J. Tang, Deep adversarial network alignment, 2019, arXiv preprint arXiv:1902.10307.
[43] Y. Zhang, Y. Li, Y. Zhu, X. Hu, Wasserstein GAN based on autoencoder with back-translation for cross-lingual embedding mappings, Pattern Recognit. Lett. 129 (2020) 311–316.
[44] W.L. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: I. Guyon, U. von Luxburg, S. Bengio, H.M. Wallach, R. Fergus, S.V.N. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, pp. 1024–1034, https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.
[45] D.J. Marceau, L. Guindon, M. Bruel, C. Marois, Building temporal topology in a GIS database to study the land-use changes in a rural-urban environment, Prof. Geogr. 53 (4) (2001) 546–558, http://dx.doi.org/10.1111/0033-0124.00304.
[46] K.K. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.
[47] R. Rossi, N. Ahmed, The network data repository with interactive graph analytics and visualization, in: Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
[48] J.S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: Advances in Neural Information Processing Systems, 2011, pp. 2546–2554.
[49] J. Bergstra, D. Yamins, D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: International Conference on Machine Learning, 2013, pp. 115–123.
[50] R. Kuprieiev, D. Petrov, R. Valles, P. Redzyński, C. da Costa-Luis, A. Schepanovski, S. Pachhai, I. Shcheklein, J. Orpinel, F. Santos, P. Rowlands, A. Sharma, Zhanibek, D. Hodovic, A. Grigorev, Earl, karajan1001, N. Dash, G. Vyshnya, nik123, maykulkarni, xliiv, M. Hora, Vera, S. Mangal, W. Baranowski, C. Wolff, A. Maslakov, A. Khamutov, K. Benoy, DVC: Data version control - git for data & models, 2020, http://dx.doi.org/10.5281/zenodo.3998731.
[51] C. De Sa, A. Gu, C. Ré, F. Sala, Representation tradeoffs for hyperbolic embeddings, Proc. Mach. Learn. Res. 80 (2018) 4460.

4.4 Retrofitting Structural Graph Embeddings with Node Attribute Information

Authors:	Piotr Bielak, Daria Puchalska, Tomasz Kajdanowicz
Published at:	International Conference on Computational Science 2022 (CORE A)
MEIN points:	140
Citations:	Web of Science: 0
(as of 05.06.2023)	Scopus: 1
	Google Scholar: 1
Credit:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization

Retrofitting structural graph embeddings with node attribute information

Piotr Bielak[1][0000−0002−1487−2569], Daria Puchalska[1], and Tomasz Kajdanowicz[1][0000−0002−8417−1012]

Department of Artificial Intelligence,
Wrocław University of Science and Technology,
Wrocław, Poland
piotr.bielak@pwr.edu.pl

Abstract. Representation learning for graphs has attracted increasing attention in recent years. In this paper, we define and study a new problem of learning attributed graph embeddings. Our setting considers how to update existing node representations from structural graph embedding methods when some additional node attributes are given. To this end, we propose Graph Embedding RetroFitting (GERF), a method that delivers a compound node embedding that follows both the graph structure and attribute space similarity. Unlike other attributed graph embedding methods, GERF is a novel representation learning method that does not require recalculation of the embedding from scratch but rather uses existing ones and retrofits the embedding according to neighborhoods defined by the graph structure and the node attributes space. Moreover, our approach keeps the same embedding space all the time and allows comparing the positions of embedding vectors and quantifying the impact of attributes on the representation update. Our GERF method updates embedding vectors by optimizing the invariance loss, graph neighbor loss, and attribute the neighbor loss to obtain high-quality embeddings. Experiments on WikiCS, Amazon-CS, Amazon-Photo, and Coauthor-CS datasets demonstrate that our proposed algorithm receives similar results compared to other state-of-the-art attributed graph embedding models despite working in retrofitting manner.

Keywords: graph embedding · attributed graphs · graph embedding retrofitting.

1 Introduction

Machine learning methods have been studied in a variety of applications and data types, including images and video (computer vision), text (natural language processing), audio or time-series data, among many others. Since most downstream ML models expect a vector from a continuous space as input, representation learning methods have been developed to create those representation vectors (embeddings) automatically. While there are many embedding methods traditional data types, such as word2vec [11] and FastText [4] for text, or ResNet

[7] and EfficientNet [14] for images, this task is much more difficult for graph-structured data. A simple concatenation of unimodal representations (graph structure and node attributes) is often not sufficient, as it does not consider the mutual relationships between modalities. Therefore, the main challenge for such methods is discovering the interrelationship between multiple modalities to create a coherent representation that will integrate the multimodal information.

Problem statement Consider a situation in which data changing over time is analyzed on an ongoing basis. In the first case, the structure of the network remains unchanged, but the attributes of the nodes are constantly changing – an example may be a network of connected weather sensors. Conversely, the values of the node attributes can be constant, but the structure of the graph changes, e.g., in a telephone network, where the edge denotes the currently ongoing call. In both situations, graph embedding models that consider both the network structure and node attributes can be used, however, if one of these modalities does not change over time, this may not be the best solution. Especially, in the first of the above-mentioned situations, it may be more advantageous to generate the structural graph embeddings once, and then use a method that would modify them depending on the current values of the attributes, somehow incorporating information from the attribute space into the structural embedding space. The simplest solution would be to simply concatenate both vectors, but the resulting representation would be neither consistent nor low-dimensional.

Goal The aim of this work is to develop an algorithm that will enhance (retrofit) existing structural node embeddings by incorporating information from the attribute space. That is, based on the node attributes, it will appropriately modify the embedding vectors derived from the structural graph representation learning methods. The new embedding vectors returned by such method should provide better performance in downstream tasks than by using naive approaches (like concatenation of structural embeddings with node attribute vectors).

Contributions We summarize our contributions as follows: (1) we introduce a new problem in the area of graph representation learning, in which a structural network embedding is updated (retrofitted) according to node attributes, (2) we propose a novel method (GERF) for unsupervised representation learning on graphs which combines information from the space of structural embeddings and node attributes while maintaining low dimensionality of the representation vectors, (3) we perform experiments demonstrating competitive quality of the proposed GERF method compared to other approaches, (4) we make our code and experimental pipeline publicly available to ensure reproducibility: `https://github.com/pbielak/gerf/`.

2 Related Work

The problem of graph representation learning (GRL) has received a lot of attention in recent years in the machine learning community. The main goal is to learn

low-dimensional continuous vector representations (embeddings), which can be later used for specific downstream prediction tasks such as node classification or link prediction. We can distinguish two groups of embeddings in GRL methods: (i) structural embeddings that take into account information extracted from the network structure only, such as the neighborhood of nodes proximities, and (ii) attributed embeddings which, apart from the relationships in the network structure, also reflect the similarity in the node feature space.

2.1 Structural representation learning methods

Early GRL methods built low-dimensional node embeddings that reflect the structure of the network. Among them, the most frequently referenced and used are: DeepWalk [12], Node2vec [6], LINE [15] and SDNE [16]. **DeepWalk** [12] samples node sequences using random walks and passes them into the Skip-gram model [11] (a word embedding method). **Node2vec** [6] extends DeepWalk by developing a biased random walk procedure to explore diverse neighborhoods by interpolating between a breadth-first (BFS) and depth-first (DFS) graph search algorithms. **LINE** [15] is a scalable method that learns node representations by preserving the first-order (similar embeddings of neighbor nodes) and second-order graph proximities (similar embeddings of nodes sharing the same neighborhood). **SDNE** [16] also focuses on preserving the first-order and second-order proximities. However, it uses an autoencoder approach to map the highly non-linear underlying network structure to latent space.

2.2 Attributed graph embedding methods

The structure of the network is given by connections between objects. However, there are many other possible sources of information. Additional node attributes can be given in the form of a vector representation of their content, which in the case of classic methods such as *bag-of-words* model or *tf-idf* is an additional challenge because these vectors are usually sparse. Methods designed to learn representations in attributed networks include TADW [17], FSCNMF [3], DANE [5] and ANRL [18]. **TADW** [17] (**T**ext-**A**ssociated **D**eep**W**alk) shows that DeepWalk is equivalent to matrix factorization and proposes its text-associated version. **FSCNMF** [3] is based on non-negative matrix factorization and produces node embeddings that are consistent with the graph structure and nodes' attributes. The structure-based embedding matrix serves as a regularizer when optimizing the attribute-based embedding matrix and vice-versa.

3 Notations and problem definition

Graph A graph G is a pair $G = (V, E)$, where $V = \{v_1, \ldots, v_{|V|}\}$ is a set of nodes and $E \subseteq V \times V$ is the set of edges that connect node pairs, i.e., each edge e_{ij} is a pair (v_i, v_j). The graph connectivity can be represented as an adjacency matrix $\mathbf{A} \in \{0, 1\}^{|V| \times |V|}$ with element A_{ij} indicating the existence of an edge (v_i, v_j).

Attributed graph Apart from the structure of connections between objects, this kind of graph has additional information about each of the nodes, i.e., each node has an assigned feature vector (also called the attribute vector). An attributed graph is a 3-tuple $G = (V, E, \mathbf{X})$, where V and E follow the previous definition. $\mathbf{X} \in \mathbb{R}^{|V| \times d_X}$ is a matrix that encodes all node attributes information, and $\mathbf{x}_i$ describes the attributes associated with node v_i.

Attribute proximity We can analyze the similarity between nodes not only based on the network structure but also in the attribute space. Given a network $G = \{V, E, \mathbf{X}\}$, the attribute proximity of two nodes v_i and v_j is determined by the similarity of $\mathbf{x}_i$ and $\mathbf{x}_j$. Note that these are two separate spaces to analyze. The similarity of two nodes in the graph structure does not imply their similarity in the attribute space and vice versa. Thus, the representation learning methods for attributed graphs should take into account dependencies in both spaces and coherently combine them.

Node representation learning Given a network $G = (V, E)$ (or $G = (V, E, \mathbf{X})$ in the case of an attributed network), the goal is to represent every graph node $v_i \in V$ as a low-dimensional vector $\mathbf{z}_i$ (called node embedding) by learning a mapping function $f : v_i \rightarrow \mathbf{z}_i \in \mathbb{R}^{d_Z}$, where $d_Z << |V|$, such that important network properties are preserved in the embedding space (e.g. structural and semantic graph information). Overall, the node embeddings are stored as a node embedding matrix $\mathbf{Z} \in \mathbb{R}^{V \times d_Z}$. If two nodes are similar in the graph structure (they are connected or share neighbors), or have similar attribute values, their learned embeddings should also be similar.

Attribute-based neighborhood We can easily define the neighborhood of a node in the network as the set of other nodes that are connected to it by an edge. However, there are no clear relationships between objects within the attribute space itself. To combine information from the attribute space (which objects are closer to each other in this space and which are further) with structural relationships, it is necessary to first define the so-called *attribute-based neighborhood*. Based on the attribute matrix $\mathbf{X}$, for each node in the network G, its k nearest neighbors in this space were found based on the Euclidean distance metric, with k being equal to the number of neighbors of this node in the network G. The neighborhood of the node v_i in the attribute space, defined in this way, will be denoted by $\mathcal{N}_{\mathbf{X}}(v_i)$. Therefore, $\forall_i \; |\mathcal{N}(v_i)| = |\mathcal{N}_{\mathbf{X}}(v_i)|$.

4 Graph Embedding RetroFitting (GERF)

In this section, we describe our proposed Graph Embedding RetroFitting model that allows to update existing structural node embeddings $\mathbf{Z}$ with the node attribute information $\mathbf{X}$, resulting in the retrofitted node embeddings $\mathbf{Z}^*$. Figure 1 shows the overall processing pipeline of our method.

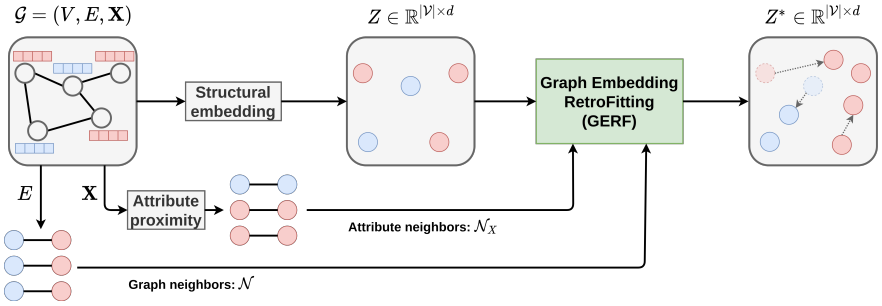


Fig. 1. Our proposed graph embedding retrofitting (GERF) method uses information about the structure of the graph (graph neighbors) and the node attribute space (attribute neighbors) to retrofit a structural embedding Z into one that incorporates node attribute information Z^*.

4.1 Objective function

Our model is based on the optimization of the $\mathbf{Z}^* \in \mathbb{R}^{V \times d}$ matrix. The objective function of our proposed GERF model takes the following form:

$$\begin{aligned}\mathcal{L}(\mathbf{Z}^*) = (1 - \lambda_G - \lambda_X) \sum_{i=1}^{n} \|\mathbf{z}_i^* - \mathbf{z}_i\|^2 \\ + \lambda_G \sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}(v_i)} \frac{\|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2}{|\mathcal{N}(v_i)|} + \lambda_X \sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}_{\mathbf{X}}(v_i)} \frac{\|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2}{|\mathcal{N}_{\mathbf{X}}(v_i)|},\end{aligned} \tag{1}$$

where $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_n)$ are the pre-trained structural embeddings for each node, $\mathbf{Z}^* = (\mathbf{z}_1^*, \ldots, \mathbf{z}_n^*)$ are the new embeddings combining multimodal information (from both spaces), and $\lambda_G > 0$ and $\lambda_X > 0$ are non-negative method hyperparameters that control the importance of the structural and attribute similarity, respectively.

With the purpose of the work in mind, one can easily explain the intuition behind each component in the objective function and why it should be included there. Since the method is intended to enhance the space of structural embeddings by incorporating information from the attribute space, it is necessary to include an a component in the objective function that will "keep the embeddings in place". That is, make sure that the new embeddings do not deviate significantly from their original values because this would lead to a complete loss of information from this space. Hence, what is needed is a component which is later referred to as **invariance loss**:

$$\mathcal{L}_I(\mathbf{Z}^*) = \sum_{i=1}^{n} \|\mathbf{z}_i^* - \mathbf{z}_i\|^2. \tag{2}$$

Further, in order to combine information from the network structure and node attributes, for each node its neighborhood in both of these spaces is considered, as defined earlier. In order for the representation of each node to be

similar to the representation of objects close to it in both spaces, it is necessary to define the loss related to the distance between the embeddings in the network, called **graph neighbor loss**:

$$\mathcal{L}_G(\mathbf{Z}^*) = \sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}(v_i)} \frac{1}{|\mathcal{N}(v_i)|} \|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2, \tag{3}$$

as well as an analogous component that controls the distances between node embeddings based on their attributes, called **attribute neighbor loss**:

$$\mathcal{L}_X(\mathbf{Z}^*) = \sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}_{\mathbf{X}}(v_i)} \frac{1}{|\mathcal{N}(v_i)|} \|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2. \tag{4}$$

By combining equations 2-4, it is possible to write the formula in Equation 1 in a different, simpler form:

$$\mathcal{L}(\mathbf{Z}^*) = (1 - \lambda_G - \lambda_X)\, \mathcal{L}_I(\mathbf{Z}^*) + \lambda_G\, \mathcal{L}_G(\mathbf{Z}^*) + \lambda_X\, \mathcal{L}_X(\mathbf{Z}^*). \tag{5}$$

4.2 Optimization

The Adam optimizer [8] was used to minimize the objective function from Equation 1. One can easily derive the formula for the first derivative of the function $\mathcal{L}$ with respect to one vector $\mathbf{z}_i^*$ as follows:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{z}_i^*} &= 2\,(1 - \lambda_G - \lambda_X)\,(\mathbf{z}_i^* - \mathbf{z}_i) \\
&+ 2\,\lambda_G \sum_{j:\, v_j \in \mathcal{N}(v_i)} \frac{\mathbf{z}_i^* - \mathbf{z}_j^*}{|\mathcal{N}(v_i)|} - 2\,\lambda_G \sum_{j:\, v_i \in \mathcal{N}(v_j)} \frac{\mathbf{z}_j^* - \mathbf{z}_i^*}{|\mathcal{N}(v_j)|} \\
&+ 2\,\lambda_X \sum_{j:\, v_j \in \mathcal{N}_{\mathbf{X}}(v_i)} \frac{\mathbf{z}_i^* - \mathbf{z}_j^*}{|\mathcal{N}_{\mathbf{X}}(v_i)|} - 2\,\lambda_X \sum_{j:\, v_i \in \mathcal{N}_{\mathbf{X}}(v_j)} \frac{\mathbf{z}_j^* - \mathbf{z}_i^*}{|\mathcal{N}_{\mathbf{X}}(v_j)|}.
\end{aligned}$$

The matrix $\mathbf{Z}^*$ is initialized with the values of $\mathbf{Z}$. Currently, the values of λ_G and λ_X hyperparameters are determined based on grid search and simultaneous analysis of the model results in downstream tasks. However, more advanced techniques could be proposed for this purpose, e.g. based on the properties of the network structure and attribute space, which is planned for future work.

4.3 Summary

The proposed method allows the creation of a coherent representation for nodes in the network based on their attribute values and pre-trained structural embeddings. It assumes that there are dependencies between objects in the attribute space. Objects are considered adjacent if the distance between their attribute vectors is sufficiently small compared to others. The main advantages of this method are intuitive assumptions and simplicity of operation, as it allows for easy integration of multimodal information from the network structure and node attributes in the form of a low-dimensional representation vector.

5 Experimental setup

We perform an analysis of selected graph representation learning methods in the node classification downstream task. We compare attributed graph embedding models (TADW, FSCNMF, DGI), structural embeddings (node2vec, LINE, SDNE), and the ones modified by the proposed GERF method and a few other baselines with each other. Additionally, a simple approach is tested that completely ignores the network structure and uses only node attributes for the prediction. Four real-world benchmark datasets are employed.

5.1 Datasets

We employ four real-wold benchmark datasets from the *PyTorch-Geometric* [2] library. The statistics are provided in Table 1.

- **WikiCS** [10] is a network of Computer Science-related Wikipedia articles with edges denoting references between those articles. Each article belongs to one of 10 subfields (classes) and has features computed as averaged GloVe embeddings of the article content. We use the first provided train/val/test data splits without any modifications (we recompute the embeddings 10 times).
- **Amazon Computers** (Amazon-CS), **Amazon Photos** [9] are two networks extracted from Amazon's co-purchase data. Nodes are products and edges denote that these products were often bought together. Based on the reviews, each product is described using a Bag-of-Words representation (node features). There are 10 and 8 product categories (node classes), respectively. There are no data splits available for those datasets, so we generate a random train/val/test split (10%/10%/80%) for each one.
- **Coauthor-CS** is a network extracted from the Microsoft Academic Graph [13]. Nodes are authors, and edges denote a collaboration of two authors. Each author is described by the keywords used in their articles (Bag-of-Words representation; node features). There are 15 author research fields (node classes). Similar to the Amazon datasets, there is no data split provided, so we generate a random train/val/test split (10%/10%/80%).

Table 1. Datasets statistics.

Name	Nodes	Edges	Features	Classes
WikiCS	11,701	216,123	300	10
Amazon Computers	13,752	245,861	767	10
Amazon Photos	7,650	119,081	745	8
Coauthor-CS	18,333	81,894	6,805	15

5.2 Embedding methods

To be able to make a qualitative comparison of embedding methods and their ability to compress highly non-linear dependencies in a low-dimensional space, it was concluded that the same size of embedding would be assumed for each of the methods. Thus, each of the algorithms produces 128-dimensional representation vectors. The exact settings for the structural embedding methods are listed below:

- **node2vec** – the same default parameters were adopted for each of the datasets because the quality of the representation vectors obtained in this way was satisfactory (note that the priority of the work is not to achieve the highest possible results of the compared algorithms, but to show that the method proposed in Section 4 is able to improve them, therefore little importance was given to the search for the best set of hyperparameters). We use the following settings: batch size – 128, learning rate – 0.01, walk length – 20, number of walks per node – 10, context size – 10, and number of negative samples – 1. Besides, the algorithm parameters p and q have been set to 1, which means that it is effectively DeepWalk.
- **LINE** – the number of epochs was set to 10 for all datasets, and the mini-batch size was set to 128 (WikiCS, Coauthor-CS), 4096 (Amazon-CS), 256 (Amazon-Photos). Such settings were required as otherwise, the training resulted in collapsed embeddings or NaN values in the embedding vectors. The number of samples in the negative sampling procedure was set to 1 for all datasets.
- **SDNE** – these embeddings showed the worst quality in the downstream task, therefore a hyperparameter grid search was performed, searching for the optimal values of the α, β, ν parameters, as well as the number of epochs and the size of the autoencoder hidden layer. Based on preliminary experiments, the number of epochs was set to 50 and the hidden layer size to 256 for all datasets. The values of method parameters α was chosen to be 10^{-4}, β was left at default 5 and ν_1, ν_2 were set to 10^{-5}, 10^{-4}, respectively.

As the proposed GERF method combines information from the network structure and the attribute space, it can be compared with attributed representation learning methods for graphs. We choose the following:

- **TADW** – the learning rate was set to 0.01 and the number of epochs to 20, the other parameters were taken as defaults from the *Karate Club* implementation [1].
- **FSCNMF** – the number of epochs was set to 500, other settings are also default from the *Karate Club* implementation.
- **DGI** – we use a single layer Graph Convolutional (GCN) encoder network with PReLU activation and train it using the Adam optimizer with a learning rate of 0.001.

In addition to the above-mentioned representation learning methods, an approach in which the graph structure is completely ignored and only node attributes are used for prediction is additionally tested. Such a representation of

nodes is high-dimensional and extremely sparse, and in experiments, it will be referred to as **features**.

5.3 Baselines

To check the quality of the proposed method, which allows for modifying the existing structural embeddings based on node attributes, it is compared with several baseline methods:

- **Concat** – concatenation of the structural embedding and the attribute vector for each node (note the large dimension size of such a representation),
- **ConcatPCA** – in this method, the dimensionality of the concatenated structural embedding and the feature vector is reduced to the size of the embedding only, using *Principal Component Analysis*,
- **MLP** – a simple autoencoder architecture, with an encoder consisting of three linear layers: the first one with the size of $d_X + d_Z$ neurons with the ReLU activation function, another with the size of $(d_X + d_Z)//2$ also with the ReLU activation function, and the last one with the size of d_Z with the Tanh activation. The decoder is a single linear layer that takes a vector from a hidden space with dimension d_Z and returns a vector of size $d_X + d_Z$, which should be the best reconstruction of the input vector to the encoder. The autoencoder was trained for 20 epochs with the mini-batch size of 128, and Adam with learning rate of 0.001 was used as the optimizer.

5.4 GERF hyperparameters

The hyperparameter values (λ_G, λ_X) of the proposed GERF method were determined by performing a grid search (see Table 2). For each of the hyperparameters, we checked the following values: $0, 0.1, \ldots, 1.0$, while preserving the overall hyperparameter constraints ($\lambda_G + \lambda_X \leq 1$). Moreover, the learning rate was set to 10^{-1}, while the number of epochs was set to 100.

Table 2. Best found GERF hyperparameter values (λ_G, λ_X) for all datasets.

Dataset	node2vec (λ_G, λ_X)	LINE (λ_G, λ_X)	SDNE (λ_G, λ_X)
WikiCS	(0.1, 0.4)	(0.4, 0.4)	(0.3, 0.4)
Amazon-CS	(0.2, 0.3)	(0.3, 0.2)	(0.8, 0.0)
Amazon-Photo	(0.2, 0.4)	(0.3, 0.3)	(0.5, 0.2)
Coauthor-CS	(0.2, 0.5)	(0.6, 0.3)	(0.9, 0.0)

While performing the grid search, we collected the node classification performance for each hyperparameter setting (not only the best one). We present

the influence of each hyperparameter on the overall node classification performance in Figure 2 (for the WikiCS dataset). We notice that the results for both hyperparameters form a convex function with a single value that maximizes the downstream task performance. We also note that the proposed GERF method is robust in terms of hyperparameters. However, it is worth optimizing them as we observed up to 5 pp dispersion in the hyperparameter combinations impact on AUC (depending on the structural embeddings – LINE, node2vec, SDNE).

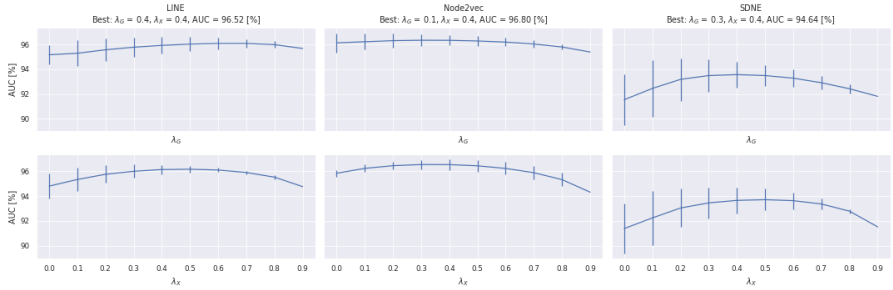


Fig. 2. Evaluation of different hyperparameter (λ_G, λ_X) values of the proposed GERF method in the node classification task on the WikiCS dataset. We present the mean and standard deviation of AUC (validation split) for each of the hyperparameter values. Note that while keeping one parameter fixed, we compute the AUC statistics over all possible values of the other hyperparameter.

6 Node classification

Setup The embeddings returned by the attributed representation learning methods, structural embeddings (themselves and enhanced by the proposed GERF method and baselines), as well as node attribute vectors, were compared in the node classification task. We compute the embeddings of both datasets 10 times to mitigate the random nature of the methods and their optimization procedure for both the structural and attributed graph representation learning methods. Each of those 10 embeddings is processed by the baselines and the proposed GERF model. We use a L_2 regularized logistic regression (from the *scikit-learn* package) trained on the embedding vectors (input) and the class information (output). The maximum number of iterations was set to 250, other parameters were left with their default values.

The classification results in terms of the AUC metric are shown in Table 3 For each of the three structural node embedding methods – node2vec, LINE and SDNE – as well as the embeddings updated by the baselines and our proposed GERF method – the best result is marked in bold. We report both the mean and standard deviation over 10 embedding recalculations.

Table 3. Node classification results in terms of the mean and standard deviation of the AUC metric over 10 recomputations of embeddings. For each structural embedding method (node2vec, LINE and SDNE) and their updated versions (by baselines and our proposed GERF method) we mark the best result in bold.

Method	WikiCS	Amazon-CS	Amazon-Photo	Coauthor-CS
features	94.79 ± 0.00	90.10 ± 0.00	94.58 ± 0.00	98.16 ± 0.00
node2vec	93.97 ± 0.15	98.22 ± 0.04	98.64 ± 0.04	98.33 ± 0.04
Concat (node2vec)	96.25 ± 0.10	98.23 ± 0.04	98.65 ± 0.04	98.38 ± 0.04
ConcatPCA (node2vec)	96.01 ± 0.11	98.22 ± 0.04	98.64 ± 0.04	98.34 ± 0.04
MLP (node2vec)	96.03 ± 0.08	98.29 ± 0.04	98.66 ± 0.04	98.41 ± 0.04
GERF (node2vec)	**96.28 ± 0.09**	**98.65 ± 0.04**	**99.18 ± 0.03**	**99.23 ± 0.02**
LINE	91.74 ± 0.20	97.63 ± 0.06	98.44 ± 0.08	93.13 ± 0.28
Concat (LINE)	95.02 ± 0.15	97.65 ± 0.06	98.45 ± 0.08	93.69 ± 0.26
ConcatPCA (LINE)	94.68 ± 0.14	97.64 ± 0.06	98.44 ± 0.08	93.16 ± 0.28
MLP (LINE)	94.90 ± 0.12	97.55 ± 0.07	98.45 ± 0.06	93.39 ± 0.24
GERF (LINE)	**96.18 ± 0.05**	**98.28 ± 0.05**	**99.06 ± 0.03**	**98.39 ± 0.05**
SDNE	74.94 ± 0.71	88.24 ± 0.41	90.89 ± 0.34	67.05 ± 1.05
Concat (SDNE)	**94.14 ± 0.27**	88.81 ± 0.41	91.34 ± 0.33	**93.86 ± 0.68**
ConcatPCA (SDNE)	93.63 ± 0.31	88.46 ± 0.43	91.16 ± 0.33	92.44 ± 0.73
MLP (SDNE)	93.75 ± 0.27	87.84 ± 0.42	90.55 ± 0.30	68.13 ± 0.86
GERF (SDNE)	92.97 ± 0.74	**97.49 ± 0.07**	**98.43 ± 0.08**	87.37 ± 2.86
TADW	90.65 ± 0.00	58.71 ± 0.00	55.91 ± 0.00	81.33 ± 0.00
FSCNMF	84.24 ± 0.00	49.93 ± 0.00	49.56 ± 0.00	50.14 ± 0.00
DGI	93.54 ± 0.17	78.20 ± 0.55	90.02 ± 0.54	98.48 ± 0.06

Discussion The first thing to note is the high score of the model that only uses node attributes to predict the class label and completely ignores information from the network structure despite the extremely large dimensionality of such a representation. In all cases, using only node attributes (features) for their classification gave better or similar results than the attributed representation learning methods. However, it should be noted that due to the long time of operation of these algorithms, the hyperparameter grid search was not performed, and their default values were adopted, which could have an impact on the results obtained.

Structural embedding vectors for WikiCS performed even worse than the attributed ones, particularly those learned using SDNE. However, the quality of the predictions increased significantly (in the case of SDNE, one could even say drastically) as they were processed by the baselines or the proposed GERF method, which can be seen in the increase of the AUC measure even by almost 18 percentage points (comparing SDNE and GERF (SDNE) embeddings for WikiCS)!

In general, in each case, the proposed GERF method allowed the incorporation of information from the attribute space into the structural embedding

space, improving the quality of prediction in this downstream task. Surprisingly good results were achieved with the use of this method on the node2vec and LINE embeddings, where it turned out to be not only better than the dedicated attributed methods, but also than all the proposed baselines, and in a significant way.

In the case of the SDNE embeddings group and the WikiCS and Coauthor-CS datasets, the baselines (*Concat*, *ConcatPCA* and *MLP*) methods turned out to be better than the proposed GERF method, but it is worth noting that both have some disadvantages. The concatenation of the attribute vector and structural embedding, which has proven to be best is highly dimensional and inconsistent with the structural embedding part. On the other hand, the *MLP* refiner, based on a simple autoencoder architecture, can exhibit problems when reconstructing sparse attribute vectors. All things considered, the results obtained by the proposed GERF method are satisfactory. While maintaining a low-dimensional representation, which allows saving memory, it achieves results similar or better to other methods, which depends on the quality of the underlying structural embeddings.

7 Conclusions

In this paper, we introduced a new graph representation learning problem setting, where given already precomputed structural node embeddings, we want to update them accordingly to node attributes, in such a way that the resulting embedding will preserve the information from both the structure and attributes. We proposed a novel graph embedding retrofitting model (GERF), which solves this problem by optimizing a compound loss function, which includes an invariance loss (keeping the new embedding close to the structural one), a graph neighborhood loss (which pushes embedding of neighboring nodes closer together) and a attribute neighborhood loss (which decreases the distance of embeddings of nodes with similar attributes). We evaluate this method on four real-world benchmark datasets (WikiCS, Amazon-CS, Amazon-Photo and Coauthor-CS), comparing it to attributed graph representation learning methods and other baselines and find that our method allows to enhance structural embeddings and results in better downstream node classification performance. In all cases, our method achieves the best results compared to other attribute aware embedding methods as well as for all datasets. In future, we want to find a way to automatically determine the hyperparameters (λ_G and λ_X) of our method based on the available graph data.

References

1. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. https://github.com/benedekrozemberczki/karateclub
2. PyTorch geometric main page. https://pytorch-geometric.readthedocs.io/en/latest/index.html

3. Bandyopadhyay, S., Kara, H., Kannan, A., Murty, M.: FSCNMF: Fusing Structure and Content via Non-negative Matrix Factorization for Embedding Information Networks (04 2018)
4. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics **5**, 135–146 (2017)
5. Gao, H., Huang, H.: Deep Attributed Network Embedding. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18. pp. 3364–3370 (2018). https://doi.org/10.24963/ijcai.2018/467
6. Grover, A., Leskovec, J.: Node2vec: Scalable Feature Learning for Networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 855–864. KDD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939754
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
9. McAuley, J., Targett, C., Shi, Q., van den Hengel, A.: Image-based recommendations on styles and substitutes. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. p. 43–52. Association for Computing Machinery (2015). https://doi.org/10.1145/2766462.2767755
10. Mernyei, P., Cangea, C.: Wiki-cs: A wikipedia-based benchmark for graph neural networks. arXiv preprint arXiv:2007.02901 (2020)
11. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient Estimation of Word Representations in Vector Space. In: Bengio, Y., LeCun, Y. (eds.) 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings (2013)
12. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 701–710. KDD '14, ACM (2014)
13. Sinha, A., Shen, Z., Song, Y., Ma, H., Eide, D., Hsu, B.J.P., Wang, K.: An overview of microsoft academic service (mas) and applications. In: Proceedings of the 24th International Conference on World Wide Web. p. 243–246. WWW '15 Companion, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2740908.2742839, https://doi.org/10.1145/2740908.2742839
14. Tan, M., Le, Q.: EfficientNet: Rethinking model scaling for convolutional neural networks. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 6105–6114. PMLR (09–15 Jun 2019)
15. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: LINE: Large-scale Information Network Embedding. Proceedings of the 24th International Conference on World Wide Web (May 2015). https://doi.org/10.1145/2736277.2741093, http://dx.doi.org/10.1145/2736277.2741093
16. Wang, D., Cui, P., Zhu, W.: Structural Deep Network Embedding. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and

Data Mining. p. 1225–1234. KDD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939753
17. Yang, C., Liu, Z., Zhao, D., Sun, M., Chang, E.Y.: Network representation learning with rich text information. In: Proceedings of the 24th International Conference on Artificial Intelligence. p. 2111–2117. IJCAI'15, AAAI Press (2015)
18. Zhang, Z., Yang, H., Bu, J., Zhou, S., Yu, P., Zhang, J., Ester, M., Wang, C.: ANRL: Attributed Network Representation Learning via Deep Neural Networks. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18. pp. 3155–3161. International Joint Conferences on Artificial Intelligence Organization (7 2018). https://doi.org/10.24963/ijcai.2018/438

4.5 A deeper look at Graph Embedding RetroFitting

Authors:	Piotr Bielak, Jakub Binkowski, Albert Sawczyn, Katsiaryna Viarenich, Daria Puchalska, Tomasz Kajdanowicz
Published at:	Journal of Computational Science (IF 3.817)
MEIN points:	100
Citations: (as of 05.06.2023)	Web of Science: 0 Scopus: 0 Google Scholar: 0
Credit:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization

Journal of Computational Science 68 (2023) 101979

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

A deeper look at Graph Embedding RetroFitting

Piotr Bielak, Jakub Binkowski, Albert Sawczyn, Katsiaryna Viarenich, Daria Puchalska, Tomasz Kajdanowicz *

Wroclaw University of Science and Technology, Wroclaw, Poland

ARTICLE INFO

Keywords:
Graph embedding
Attributed graphs
Retrofitting

ABSTRACT

Representation learning for graphs has attracted increasing attention in recent years. In particular, this work is focused on a new problem in this realm which is learning attributed graph embeddings. The setting considers how to update existing node representations from structural graph embedding methods when some additional node attributes are given. Recently, Graph Embedding RetroFitting (GERF) Bielak et al. was proposed to this end – a method that delivers a compound node embedding that follows both the graph structure and attribute space similarity. It uses existing structural node embeddings and retrofits them according to the neighborhood defined by the node attributes space (by optimizing the invariance loss and the attribute neighbor loss). In order to refine GERF method, we aim to include the simplification of the objective function and provide an algorithm for automatic hyperparameter estimation, whereas the experimental scenario is extended by a more robust hyperparameter search for all considered methods and a link prediction problem for evaluation of node embeddings.

1. Introduction

Machine learning methods have been studied in a variety of applications and data types, including images and video (computer vision), text (natural language processing), audio or time-series data, among many others. Since most downstream ML models expect a vector from a continuous space as input, representation learning methods have been developed to create those representation vectors (embeddings) automatically. While there are many embedding methods traditional data types, such as word2vec [1] and FastText [2] for text, or ResNet [3] and EfficientNet [4] for images, this task is much more difficult for graph-structured data. A simple concatenation of unimodal representations (graph structure and node attributes) is often not sufficient, as it does not consider the mutual relationships between modalities. Therefore, the main challenge for such methods is discovering the interrelationship between multiple modalities to create a coherent representation that will integrate the multimodal information.

Problem statement. Consider a situation in which data changing over time is analyzed on an ongoing basis. In the first case, the structure of the network remains unchanged, but the attributes of the nodes are constantly changing — an example may be a network of connected weather sensors. Conversely, the values of the node attributes can be constant, but the structure of the graph changes, e.g., in a telephone network, where the edge denotes the currently ongoing call. In both situations, graph embedding models that consider both the network structure and node attributes can be used, however, if one of these modalities does not change over time, this may not be the best solution. Especially, in the first of the above-mentioned situations, it may be more advantageous to generate the structural graph embeddings once, and then use a method that would modify them depending on the current values of the attributes, somehow incorporating information from the attribute space into the structural embedding space. The simplest solution would be to simply concatenate both vectors, but the resulting representation would be neither consistent nor low-dimensional.

GERF. The previously proposed Graph Embedding Retrofitting [5] algorithm enhances (retrofit) existing structural node embeddings by incorporating information from the attribute space. That is, based on the node attributes, it will appropriately modify the embedding vectors derived from the structural graph representation learning methods. It uses a threefold loss function, consisting of an invariance loss, a graph-neighbor loss and an attribute-neighbor loss, to update the structural node embedding vectors. The new vectors returned by such method showed better performance in downstream tasks than by using naive approaches (like concatenation of structural embeddings with node attribute vectors).

Goal. GERF showed promising performance in the problem of retrofitting graph embeddings we consider. However, we identified several areas in the original method which could be further refined. First,

* Corresponding author.
E-mail address: tomasz.kajdanowicz@pwr.edu.pl (T. Kajdanowicz).

https://doi.org/10.1016/j.jocs.2023.101979
Received 15 October 2022; Received in revised form 10 February 2023; Accepted 23 February 2023
Available online 27 February 2023

we propose to automate hyperparameter estimation with Dirichlet-Multinomial model, which helps to avoid expensive grid search. Second, we simplify the loss function by removing graph neighbor loss term, having no effect on training. Furthermore, to cover a broader range of tasks in the experimental study, we extended it with an evaluation of GERF on link prediction.

Contributions. We summarize our contributions as follows:

- we improve the previously proposed GERF [5] (Graph Embedding Retrofitting) model by simplifying the objective function,
- we propose a method for automatic estimation of GERF's hyperparameters,
- we perform additional experiments (in form of a link prediction study) demonstrating competitive quality of the proposed GERF method compared to other approaches,
- we make our code and experimental pipeline publicly available to ensure reproducibility: https://github.com/graphml-lab-pwr/gerf/.

2. Related work

The problem of graph representation learning (GRL) has received a lot of attention in recent years in the machine learning community. The main goal is to learn low-dimensional continuous vector representations (embeddings), which can be later used for specific downstream prediction tasks such as node classification or link prediction. We can distinguish two groups of embeddings in GRL methods: (i) structural embeddings that take into account information extracted from the network structure only, such as the neighborhood of nodes proximities, and (ii) attributed embeddings which, apart from the relationships in the network structure, also reflect the similarity in the node feature space.

2.1. Structural representation learning methods

Early GRL methods built low-dimensional node embeddings that reflect the structure of the network. Among them, the most frequently referenced and used are: DeepWalk [6], Node2vec [7], LINE [8] and SDNE [9]. **DeepWalk** [6] samples node sequences using random walks and passes them into the Skip-gram model [1] (a word embedding method). **Node2vec** [7] extends DeepWalk by developing a biased random walk procedure to explore diverse neighborhoods by interpolating between a breadth-first (BFS) and depth-first (DFS) graph search algorithms. **LINE** [8] is a scalable method that learns node representations by preserving the first-order (similar embeddings of neighbor nodes) and second-order graph proximities (similar embeddings of nodes sharing the same neighborhood). **SDNE** [9] also focuses on preserving the first-order and second-order proximities. However, it uses an autoencoder approach to map the highly non-linear underlying network structure to latent space.

2.2. Attributed graph embedding methods

The structure of the network is given by connections between objects. However, there are many other possible sources of information. Additional node attributes can be given in the form of a vector representation of their content, which in the case of classic methods such as *bag-of-words* model or *tf-idf* is an additional challenge because these vectors are usually sparse. Methods designed to learn representations in attributed networks include TADW [10], FSCNMF [11], DANE [12], ANRL [13], CNR [14] and Attrib2vec [15]. **TADW** [10] (**T**ext-**A**ssociated **D**eep**W**alk) shows that DeepWalk is equivalent to matrix factorization and proposes its text-associated version. **FSCNMF** [11] is based on non-negative matrix factorization and produces node embeddings that are consistent with the graph structure and nodes' attributes. The structure-based embedding matrix serves as a regularizer when optimizing the attribute-based embedding matrix and vice-versa.

Moreover, there exists another class of attributed graph embedding methods — Graph Neural Networks. These are mainly based on the so-called message-passing mechanism, where nodes generate messages (based on their attributes) and pass them, along the graph edges, to their neighboring nodes. Next, each node aggregates all received messages into a single summary vector, which is eventually used as the new node representation vector. Popular GNN architectures include: Graph Convolutional Networks (GCNs) [16], GraphSAGE [17], GAT [18], GIN [19]. However, these models are defined only as neural network layers and are mostly trained in a supervised setting, which we do not consider in our paper.

3. Notations and problem definition

Graph. A graph G is a pair $G = (V, E)$, where $V = \{v_1, \ldots, v_{|V|}\}$ is a set of nodes and $E \subseteq V \times V$ is the set of edges that connect node pairs, i.e., each edge e_{ij} is a pair (v_i, v_j) where $v_i \in V$ and $v_j \in V$. The graph connectivity can be represented as an adjacency matrix $\mathbf{A} \in \{0, 1\}^{|V|\times|V|}$ with element A_{ij} indicating the existence of an edge (v_i, v_j).

Attributed graph. Apart from the structure of connections between objects, this kind of graph has additional information about each of the nodes, i.e., each node has an assigned feature vector (also called the attribute vector). An attributed graph is a 3-tuple $G = (V, E, \mathbf{X})$, where V and E follow the previous definition. $\mathbf{X} \in \mathbb{R}^{|V|\times d_X}$ is a matrix that encodes all node attributes information where d_X is the dimensionality of the node attributes, and $\mathbf{x}_i$ describes the attributes associated with node v_i.

Attribute proximity. We can analyze the similarity between nodes not only based on the network structure but also in the attribute space. Given a network $G = \{V, E, \mathbf{X}\}$, the attribute proximity of two nodes v_i and v_j is determined by the similarity of $\mathbf{x}_i$ and $\mathbf{x}_j$. Note that these are two separate spaces to analyze. The similarity of two nodes in the graph structure does not imply their similarity in the attribute space and vice versa. Thus, the representation learning methods for attributed graphs should take into account dependencies in both spaces and coherently combine them.

Node representation learning. Given a network $G = (V, E)$ (or $G = (V, E, \mathbf{X})$ in the case of an attributed network), the goal is to represent every graph node $v_i \in V$ as a low-dimensional vector $\mathbf{z}_i$ (called node embedding) by learning a mapping function $f : v_i \to \mathbf{z}_i \in \mathbb{R}^{d_Z}$, where $d_Z \ll |V|$, such that important network properties are preserved in the embedding space (e.g. structural and semantic graph information). Overall, the node embeddings are stored as a node embedding matrix $\mathbf{Z} \in \mathbb{R}^{V\times d_Z}$. If two nodes are similar in the graph structure (they are connected or share neighbors), or have similar attribute values, their learned embeddings should also be similar.

Attribute-based neighborhood. We can easily define the neighborhood of a node i in the network $\mathcal{N}(i) : \{v \in V : (v, i) \in E \text{ or } (i, v) \in E\}$ as the set of other nodes that are connected to it by an edge – this is known as a direct or 1-hop neighborhood. However, there are no clear relationships between objects within the attribute space itself. To combine information from the attribute space (which objects are closer to each other in this space and which are further) with structural relationships, it is necessary to first define the so-called *attribute-based neighborhood*. Based on the attribute matrix $\mathbf{X}$, for each node in the network we find k nearest neighbors in this space using the Euclidean distance metric, with k being equal to the number of neighbors of this node in the network G. The neighborhood of the node v_i in the attribute space, defined in this way, will be denoted by $\mathcal{N}_{\mathbf{X}}(v_i)$. Therefore, $\forall_i\ |\mathcal{N}(v_i)| = |\mathcal{N}_{\mathbf{X}}(v_i)|$.

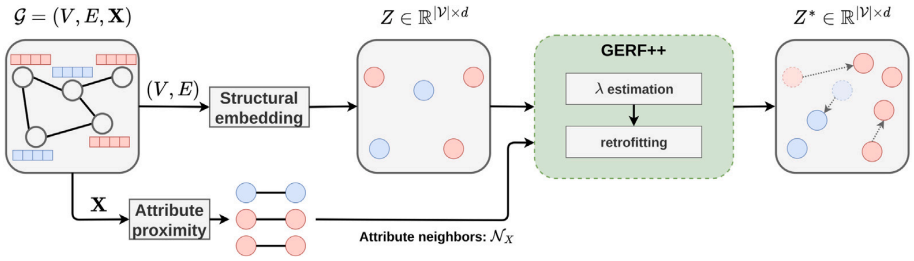


Fig. 1. The improved graph embedding retrofitting (GERF) pipeline removes the graph neighbor terms and proposes a method for automatic estimation of λ hyperparameters. Overall, the model uses the node attribute space to retrofit a structural embedding Z into one that incorporates both information sources Z^*.

4. Graph Embedding RetroFitting (GERF)

In this section, we introduce our modifications to the Graph Embedding RetroFitting model. Let us recall that GERF allows to update existing structural node embeddings $\mathbf{Z}$ with the node attribute information $\mathbf{X}$, resulting in the retrofitted node embeddings $\mathbf{Z}^*$. Fig. 1 shows the updated processing pipeline of our method.

4.1. Objective function

The original GERF model uses a threefold loss function, i.e. invariance loss, graph neighbor loss and attribute neighbor loss. Due to a more detailed experimental evaluation, we concluded that the graph neighbor loss can be omitted as the information is already present in the structural node embedding. Hence, the new objective function is defined as follows:

$$\mathcal{L}(\mathbf{Z}^*) = (1 - \lambda_X) \sum_{i=1}^{n} \|\mathbf{z}_i^* - \mathbf{z}_i\|^2 + \lambda_X \sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}_{\mathbf{X}}(v_i)} \frac{\|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2}{|\mathcal{N}_{\mathbf{X}}(v_i)|}, \tag{1}$$

where $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_n)$ are the pre-trained structural embeddings for each node, $\mathbf{Z}^* = (\mathbf{z}_1^*, \ldots, \mathbf{z}_n^*)$ are the new embeddings combining multimodal information (from both spaces), and $\lambda_X \in [0, 1]$ is a method hyperparameter that control the tradeoff between the structural and attribute similarity. Moreover, we define $\lambda_Z = 1 - \lambda_X$.

With the purpose of the work in mind, one can easily explain the intuition behind each component in the objective function and why it should be included there. Since the method is intended to enhance the space of structural embeddings by incorporating information from the attribute space, it is necessary to include a component in the objective function that will "keep the embeddings in place". That is, make sure that the new embeddings do not deviate significantly from their original values because this would lead to a complete loss of information from this space. Hence, what is needed is a component which is later referred to as **invariance loss**:

$$\mathcal{L}_I(\mathbf{Z}^*) = \sum_{i=1}^{n} \|\mathbf{z}_i^* - \mathbf{z}_i\|^2. \tag{2}$$

Further, in order to incorporate the information from the node attributes, for each node its attribute neighborhood is considered (as defined earlier). The component that controls the distances between node embeddings based on their attributes is called the **attribute neighbor loss**:

$$\mathcal{L}_X(\mathbf{Z}^*) = \sum_{i=1}^{n} \sum_{j:\, v_j \in \mathcal{N}_{\mathbf{X}}(v_i)} \frac{1}{|\mathcal{N}(v_i)|} \|\mathbf{z}_i^* - \mathbf{z}_j^*\|^2. \tag{3}$$

By combining Eqs. (2) and (3), it is possible to write the formula in Eq. (1) in a different, simpler form:

$$\mathcal{L}(\mathbf{Z}^*) = \lambda_Z \mathcal{L}_I(\mathbf{Z}^*) + \lambda_X \mathcal{L}_X(\mathbf{Z}^*). \tag{4}$$

4.2. Optimization

We use the Adam optimizer [20] to minimize the objective function from Eq. (1). One can easily derive the formula for the first derivative of the function $\mathcal{L}$ with respect to one vector $\mathbf{z}_i^*$ as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}_i}^* = 2\lambda_Z (\mathbf{z}_i^* - \mathbf{z}_i) + 2\lambda_X \sum_{j:\, v_j \in \mathcal{N}_{\mathbf{X}}(v_i)} \frac{\mathbf{z}_i^* - \mathbf{z}_j^*}{|\mathcal{N}_{\mathbf{X}}(v_i)|} - 2\lambda_X \sum_{j:\, v_i \in \mathcal{N}_{\mathbf{X}}(v_j)} \frac{\mathbf{z}_j^* - \mathbf{z}_i^*}{|\mathcal{N}_{\mathbf{X}}(v_j)|}.$$

The matrix $\mathbf{Z}^*$ is initialized with the values of $\mathbf{Z}$. Such an approach reduces the number of optimization steps till convergence.

4.3. Hyperparameter estimation

To find the retrofitted embedding, we need to specify the λ_X hyperparameter (we defined the other hyperparameter as $\lambda_Z = 1 - \lambda_X$). This can be done in various ways, including the usage of expert knowledge or just a naive grid search (as presented in Section 5). However, a better approach would be to estimate these hyperparameters automatically. We propose to utilize the approach based on the Dirichlet-Multinomial model, as proposed in [21]. The estimation method is depicted in Fig. 2. In particular, we evaluate the structural embedding in a node classification task (or link prediction task) and count the number of successful classification attempts S_Z. Next, we repeat the same task on the node attributes to obtain the number of successes S_X. For evaluation, we leverage logistic regression fitted on the training set and obtain a number of successes from the validation set. Then, we use these numbers as the evidence for the Multinomial distribution. To include prior knowledge about the distribution, we use the conjugate Dirichlet prior distribution with initial parameters denoted by $\boldsymbol{\alpha} = \{\alpha_Z, \alpha_X\}$. We test two strategies for the prior. First, we use a uniform setting: $\alpha_Z = \alpha_X = 1$, where the representations are equally important. Second, we compute α_Z and α_X based on our approximate measure of the network homophily, which is defined as follows, where E_X is the set of edges induced by attribute nearest neighbors (such as defined in the cost function), E_Z is the set of edges induced by structural embedding nearest neighbors, and E is the original set of network's edges:

$$\alpha_X = \frac{|E_X \cap E|}{|E|}; \alpha_Z = \frac{|E_Z \cap E|}{|E|}$$

Since the $\boldsymbol{\alpha}$ parameters for the Dirichlet prior serve as an initial belief about correct classification count, α_X and α_Z are further scaled by a fraction of the number of nodes to obtain the right units. The fraction was set to 0.05 to reflect the scenario of a preliminary trial, before the entire data to compute likelihood is available. It is worth noting that one can use more robust prior probabilities for $\boldsymbol{\alpha}$, e.g., priors based on a different definition of homophily or better scaling to the count units. Using the above assumptions, we estimate the parameters λ_Z and λ_X as the maximum a posteriori probability (MAP) of the Dirichlet-Multinomial model [22] :

$$\hat{\lambda}_Z = \frac{S_Z + \alpha_Z - 1}{S_Z + S_X + \alpha_Z + \alpha_X - 2}; \hat{\lambda}_X = \frac{S_X + \alpha_X - 1}{S_Z + S_X + \alpha_Z + \alpha_X - 2}$$

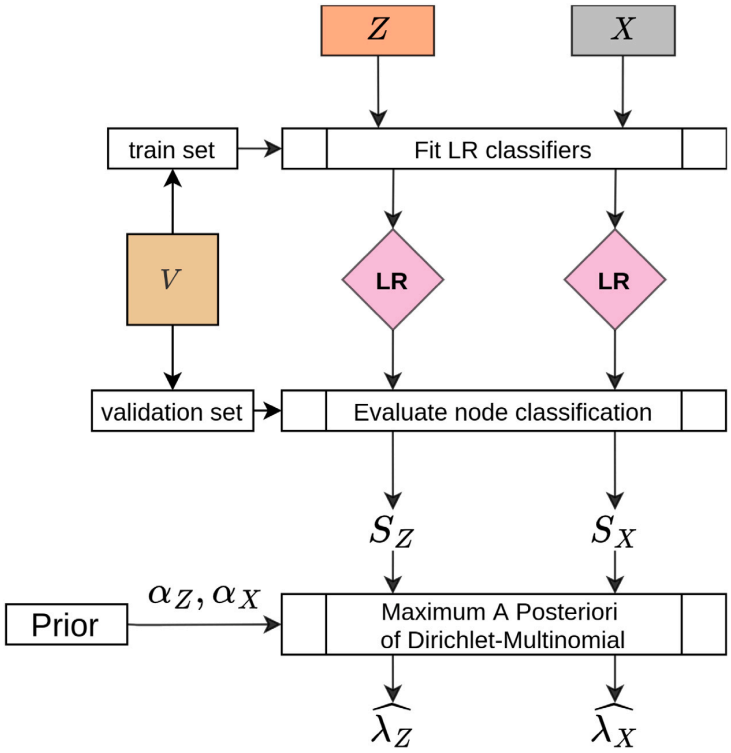


Fig. 2. Hyperparameter estimation algorithm. First, we use the structural node embedding **Z** to train a logistic regression classifier (LR) and count the number of successful predictions S_Z. We do the same for the node attributes and obtain S_X. These values are combined with prior distribution parameters to compute the MAP estimate of the Dirichlet-Multinomial model, which we use as estimates for GERF's hyperparameters, i.e., λ_Z and λ_X.

5. Experimental setup

We perform an analysis of selected graph representation learning methods in the node classification and link prediction downstream tasks. We compare attributed graph embedding models (TADW, FSC-NMF, DGI), structural embeddings (node2vec, LINE, SDNE), and the ones modified by the improved GERF method and a few other baselines with each other. Additionally, a simple approach is tested that completely ignores the network structure and uses only node attributes for the prediction (called *features* from now on). Four real-world benchmark datasets are employed.

5.1. Datasets

We employ four real-wold benchmark datasets from the *PyTorch-Geometric* [23] library. The statistics are provided in Table 1.

- **WikiCS** [24] is a network of Computer Science-related Wikipedia articles with edges denoting references between those articles. Each article belongs to one of 10 subfields (classes) and has features computed as averaged GloVe embeddings of the article content. We use the first provided train/val/test data splits without any modifications (we recompute the embeddings 10 times).
- **Amazon Computers** (Amazon-CS), **Amazon Photos** [25] are two networks extracted from Amazon's co-purchase data. Nodes are products and edges denote that these products were often bought together. Based on the reviews, each product is described using a Bag-of-Words representation (node features). There are 10 and 8 product categories (node classes), respectively.No data splits are available for those datasets, so we generate a random train/val/test split (10%/10%/80%) for each one.

Table 1
The datasets statistics.

Name	Nodes	Edges	Features	Classes
WikiCS	11,701	216,123	300	10
Amazon Computers	13,752	245,861	767	10
Amazon Photos	7,650	119,081	745	8
Coauthor-CS	18,333	81,894	6,805	15

- **Coauthor-CS** is a network extracted from the Microsoft Academic Graph [26]. Nodes are authors, and edges denote a collaboration of two authors. Each author is described by the keywords used in their articles (Bag-of-Words representation; node features). There are 15 author research fields (node classes). Similar to the Amazon datasets, no data split is provided, so we generate a random train/val/test split (10%/10%/80%).

5.2. Embedding methods

To be able to make a qualitative comparison of embedding methods and their ability to compress highly non-linear dependencies in a low-dimensional space, we use the same embedding size (equal to 128) for each of the methods. Moreover, to ensure a fair comparison, we optimize the hyperparameters of all the node representation learning methods using the Optuna library [27]. For each method, we take 100 optimization steps and tune the following parameters. Continuous parameters (marked by [ ]) are sampled from the uniform distribution in the linear domain (*) or in the log-scaled domain (**).

Discrete parameters (marked by { }) are sampled from the categorical distribution.

- **node2vec**:
  - learning rate**: [0.001, 0.1]
  - batch size: {32, 64, 128, 256}
  - walk length*: [10, 30]
  - walks per node*: [2, 20]
  - context size*: [2, 20]
  - num negative samples*: [1, 3]
  - p*: [0.5, 1.5]
  - q*: [0.5, 1.5]
- **LINE**:
  - negative ratio*: {0, 1, 2, 3}
  - order: {'first', 'second', 'all'}
  - batch size: {32, 64, 128, 256} (for Wiki-CS and Coauthor-CS), {1024, 2048, 4096, 8192} (for Amazon-CS), {64, 128, 256, 512} (for Amazon-Photo)
- **SDNE**:
  - hidden size: {128, 256, 512, 1024}
  - nu1*: [1.e-6, 1.e-4]
  - nu2*: [1.e-5, 1.e-3]
  - alpha*: [1.e-5, 1.e-3]
  - beta*: [3, 7]
  - batch size: {32, 64, 128, 256}
- **TADW**:
  - learning rate**: [0.001, 0.1]
  - reduction dimensions: {16, 32, 64, 128}
  - svd iterations*: [10, 50]
  - lambda*: [1, 15]
- **FSCNMF**:
  - alpha1*: [100, 1500]
  - alpha2*: [0.1, 1.5]
  - alpha3*: [0.1, 1.5]
  - beta1*: [100, 1500]
  - beta2*: [0.1, 1.5]
  - beta3*: [0.1, 1.5]
- **DGI** – we use a single layer Graph Convolutional (GCN) encoder network with PReLU activation and train it using the Adam optimizer; we use the default augmentation functions as given in the PyTorch-Geometric library [23]
  - learning rate**: [0.0001, 0.01]

We evaluate each embedding in the node classification task (as described below) and use the AUC on the validation set to select the best hyperparameter configuration. We attach the best hyperparameters for all datasets and methods in the code repository.

5.3. Baselines

To check the quality of the proposed method, which allows for modifying the existing structural embeddings based on node attributes, we compare it with several baseline methods:

- **Concat** – concatenation of the structural embedding and the attribute vector for each node (note the large dimension size of such a representation),
- **ConcatPCA** – in this method, the dimensionality of the concatenated structural embedding and the feature vector is reduced to the size of the embedding only, using *Principal Component Analysis*,
- **MLP** – a simple autoencoder architecture, with an encoder consisting of three linear layers: the first one with the size of $d_X + d_Z$ neurons with the ReLU activation function, another with the size of $(d_X + d_Z)//2$ also with the ReLU activation function, and the last one with the size of d_Z with the Tanh activation. The decoder is a single linear layer that takes a vector from a hidden space with dimension d_Z and returns a vector of size $d_X + d_Z$, which should be the best reconstruction of the input vector to the encoder. We train the autoencoder for 20 epochs with the mini-batch size of 128 using Adam with the learning rate of 0.001 as the optimizer.

5.4. GERF hyperparameters

Three strategies of hyperparameter values (λ_Z, λ_X) were proposed and tested: a grid search and the estimation based on the Multinomial-Dirichlet model, the best found combinations of hyperparameters are presented in Table 2. For each of the hyperparameters, we checked the following values: $0, 0.05, 0.1, \ldots, 1.0$, while preserving the overall hyperparameter constraints ($\lambda_Z + \lambda_X \leq 1$). For the Multinomial-Dirichlet model, the parameters are by default constrained to sum up to 1, and we tested two strategies for the prior parameters of Dirichlet distribution. Moreover, the learning rate was set to 10^{-1}, while the number of epochs was set to 100.

While performing the grid search, we collected the node classification performance for each hyperparameter setting (not only the best one). We present the influence of each hyperparameter on the overall node classification performance in Fig. 3. We notice that the results for both hyperparameters form a convex function with a single value that maximizes the downstream task performance. We also note that the proposed GERF method is robust in terms of hyperparameters (we do not consider the edge cases). However, it is worth optimizing them as we observed up to 5 pp dispersion in the hyperparameter combinations impact on AUC (depending on the structural embeddings — LINE, node2vec, SDNE).

6. Node classification

We evaluate node embedding vectors in a node classification task, following the setup from the original GERF paper [5]. Let us recall the details.

Setup. The embeddings returned by the attributed representation learning methods, the structural embeddings (themselves and enhanced by the GERF method and baselines), as well as node attribute vectors (*features*), were compared in the node classification task. We compute the embeddings of both datasets 10 times to mitigate the random nature of the methods and their optimization procedure for both the structural and attributed graph representation learning methods. Each of those 10 embeddings is processed by the baselines and the proposed GERF model. We use a L_2 regularized logistic regression (from the *scikit-learn* package [28]) trained on the embedding vectors (input) and the class information (output). The maximum number of iterations was set to 250, other parameters were left with their default values.

The classification results in terms of the AUC metric are shown in Table 3. For each of the three structural node embedding methods – node2vec, LINE and SDNE – as well as the embeddings updated by the baselines and the GERF method – the best result is marked in bold. We report both the mean and standard deviation over 10 embedding recalculations.

Discussion. The first thing to note is the high score of the model that uses only node attributes to predict the class label and completely

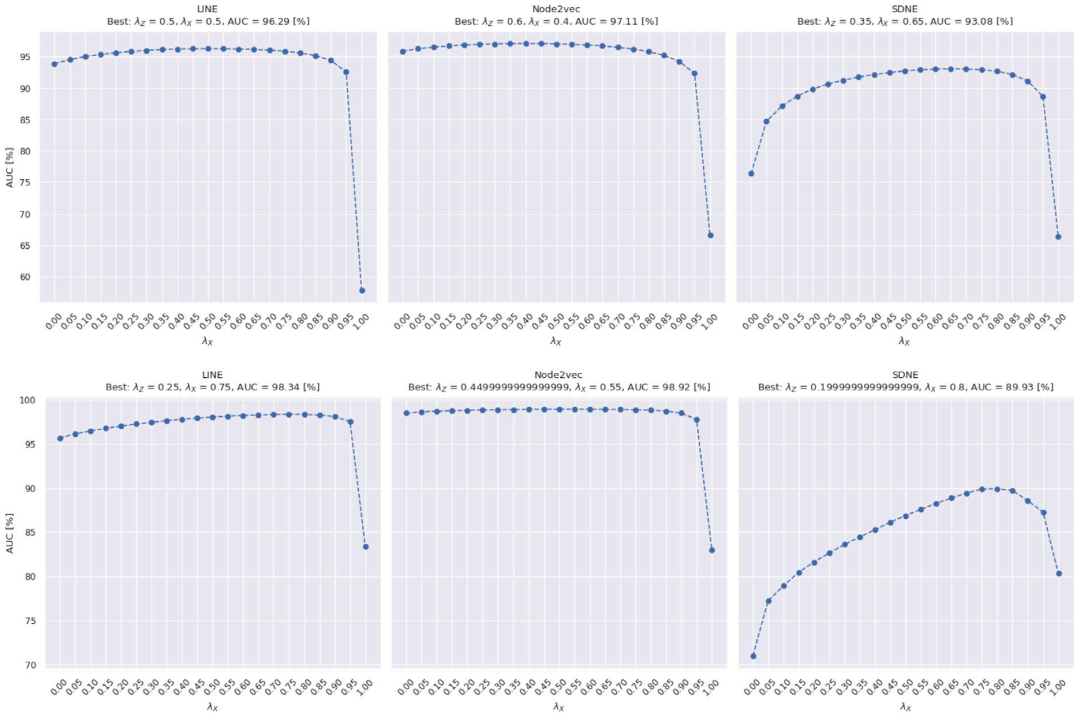


Fig. 3. The evaluation of different hyperparameters (λ_X) values of the proposed GERF method in the node classification task on the WikiCS (top) and Coauthor-CS (bottom) datasets. We present the AUC on the validation split.

Table 2
The best found GERF hyperparameter values. Each entry contains pair of values corresponding to (λ_Z, λ_X) hyperparameters for all datasets and two considered tasks (node classification and link prediction). The table contains values for three methods of hyperparameter estimation: the grid search (grid), the Multinomial-Dirichlet model with uniform prior (uniform), and the Multinomial-Dirichlet model with the homophily-based prior (homophily).

Dataset	Task	node2vec			LINE			SDNE		
		grid	uniform	homophily	grid	uniform	homophily	grid	uniform	homophily
WikiCS	node cls.	(0.6, 0.4)	(0.53, 0.47)	(0.56, 0.44)	(0.5, 0.5)	(0.51, 0.49)	(0.55, 0.45)	(0.35, 0.65)	(0.39, 0.61)	(0.41, 0.59)
	link pred.	(0.95, 0.5)	(0.54, 0.46)	(0.55, 0.45)	(0.95, 0.5)	(0.55, 0.45)	(0.55, 0.45)	(0.6, 0.4)	(0.51, 0.49)	(0.51, 0.49)
Amazon-CS	node cls.	(0.65, 0.35)	(0.69, 0.31)	(0.72, 0.28)	(0.69, 0.31)	(0.7, 0.3)	(0.71, 0.29)	(0.25, 0.75)	(0.61, 0.8)	(0.62, 0.38)
	link pred.	(1.0, 0.0)	(0.60, 0.40)	(0.60, 0.40)	(1.0, 0.0)	(0.59, 0.41)	(0.59, 0.41)	(0.9, 0.1)	(0.55, 0.45)	(0.55 0.45)
Amazon-Photo	node cls.	(0.70, 0.30)	(0.77, 0.23)	(0.78, 0.22)	(0.75, 0.25)	(0.77, 0.23)	(0.79, 0.21)	(0.40, 0.60)	(0.73, 0.27)	(0.73, 0.27)
	link pred.	(1.0, 0.0)	(0.58, 0.42)	(0.58, 0.42)	(1.0, 0.0)	(0.58, 0.42)	(0.58, 0.42)	(0.9, 0.1)	(0.53, 0.47)	(0.53, 0.47)
Coauthor-CS	node cls.	(0.45, 0.55)	(0.78, 0.22)	(0.77, 0.23)	(0.25, 0.75)	(0.76, 0.24)	(0.76, 0.24)	(0.2, 0.8)	(0.57, 0.43)	(0.53, 0.47)
	link pred.	(1.0, 0.0)	(0.57, 0.43)	(0.57, 0.43)	(1.0, 0.0)	(0.57, 0.43)	(0.57, 0.43)	(1.0, 0.0)	(0.46, 0.54)	(0.46, 0.55)

ignores information from the network structure despite the extremely large dimensionality of such a representation. In all cases, using only node attributes (features) for their classification gave better or similar results than the attributed representation learning methods.

The structural embedding vectors for WikiCS performed even worse than the attributed ones, particularly those learned using SDNE. However, the quality of the predictions increased significantly (in the case of SDNE, one could even say drastically) as they were processed by the baselines or the GERF method, which can be seen in the increase of the AUC measure even by almost 16 percentage points (comparing the SDNE and GERF embeddings for WikiCS)!

In general, in each case, the proposed GERF method allows the incorporation of information from the attribute space into the structural embedding space, improving prediction quality in this downstream task.

In the case of the node2vec and SDNE embeddings groups and the WikiCS dataset, the baselines (*Concat*, *ConcatPCA* and *MLP*) methods turn out to be better than the proposed GERF method. Still, it is worth noting that both have some disadvantages. The concatenation of the attribute vector and structural embedding, which has proven to be the best is highly dimensional and inconsistent with the structural embedding part. On the other hand, the *MLP* refiner, based on the simple autoencoder architecture, can exhibit problems when reconstructing sparse attribute vectors. All things considered, the results obtained by the proposed GERF method are satisfactory. While maintaining a low-dimensional representation, which allows saving memory, it achieves results similar or better to other methods, which depends on the quality of the underlying structural embeddings.

When comparing different hyperparameter estimation methods for GERF, i.e., *grid*, *uniform*, *homophily*, we can observe that although the *grid* variant often receives the highest AUC values, it is a much more time-consuming process. We need to examine multiple values to find the best one. Nevertheless, the *uniform* and *homophily* approaches are more suitable for real-world applications as they are (1) only slightly worse than the *grid* approach and (2) they are more time-efficient (only a single evaluation to find the hyperparameters).

Table 3
Node classification results in terms of the mean and standard deviation of the AUC metric over 10 recomputations of embeddings. For each structural embedding method (node2vec, LINE and SDNE) and their updated versions (by baselines and the GERF method) we mark the best result in bold.

Method	WikiCS	Amazon CS	Amazon Photo	Coauthor CS
features	94.79 ± 0.00	89.62 ± 0.00	94.17 ± 0.00	98.17 ± 0.00
node2vec	94.88 ± 0.11	98.15 ± 0.06	98.71 ± 0.04	98.62 ± 0.04
+ Concat	**96.75 ± 0.06**	98.16 ± 0.06	98.71 ± 0.04	98.66 ± 0.04
+ ConcatPCA	96.64 ± 0.07	98.16 ± 0.06	98.71 ± 0.04	98.63 ± 0.04
+ MLP	96.61 ± 0.07	98.23 ± 0.06	98.68 ± 0.04	98.63 ± 0.03
+ $GERF_{(grid)}$	96.40 ± 0.07	**98.56 ± 0.03**	**99.07 ± 0.03**	**99.30 ± 0.02**
+ $GERF_{(uniform)}$	96.41 ± 0.06	98.55 ± 0.03	99.04 ± 0.03	99.09 ± 0.03
+ $GERF_{(homophily)}$	96.41 ± 0.06	98.54 ± 0.03	99.03 ± 0.03	99.09 ± 0.03
LINE	93.02 ± 0.14	97.86 ± 0.10	98.62 ± 0.06	96.43 ± 0.22
+ Concat	95.77 ± 0.11	97.87 ± 0.10	98.63 ± 0.06	96.58 ± 0.21
+ ConcatPCA	95.62 ± 0.10	97.86 ± 0.10	98.62 ± 0.06	96.45 ± 0.22
+ MLP	95.63 ± 0.10	97.80 ± 0.12	98.62 ± 0.06	96.62 ± 0.20
+ $GERF_{(grid)}$	**95.78 ± 0.11**	98.22 ± 0.05	**98.84 ± 0.05**	**98.81 ± 0.04**
+ $GERF_{(uniform)}$	95.77 ± 0.12	**98.23 ± 0.05**	98.83 ± 0.05	97.81 ± 0.14
+ $GERF_{(homophily)}$	95.74 ± 0.12	98.23 ± 0.05	98.82 ± 0.05	97.79 ± 0.14
SDNE	75.99 ± 0.70	89.65 ± 0.32	92.19 ± 0.21	71.46 ± 0.78
+ Concat	**94.92 ± 0.08**	90.16 ± 0.32	92.67 ± 0.21	90.03 ± 0.25
+ ConcatPCA	94.72 ± 0.09	89.93 ± 0.33	92.55 ± 0.20	86.56 ± 0.65
+ MLP	94.50 ± 0.10	89.10 ± 0.26	91.43 ± 0.22	69.60 ± 1.04
+ $GERF_{(grid)}$	92.08 ± 0.21	**95.75 ± 0.13**	**96.94 ± 0.10**	**90.56 ± 0.44**
+ $GERF_{(uniform)}$	92.03 ± 0.21	94.65 ± 0.23	96.18 ± 0.11	86.76 ± 0.68
+ $GERF_{(homophily)}$	91.98 ± 0.21	94.60 ± 0.23	96.19 ± 0.11	87.38 ± 0.66
TADW	91.45 ± 0.00	60.88 ± 0.00	54.88 ± 0.00	83.21 ± 0.00
FSCNMF	86.60 ± 0.00	92.10 ± 0.00	92.45 ± 0.00	98.77 ± 0.00
DGI	93.42 ± 0.18	73.24 ± 0.59	88.21 ± 0.54	93.95 ± 0.50

Table 4
Link prediction results in terms of the mean and standard deviation of the AUC metric over 10 recomputations of embeddings. For each structural embedding method (node2vec, LINE and SDNE) and their updated versions (by the baselines and GERF method), we mark the best result in bold.

Method	WikiCS	Amazon CS	Amazon Photo	Coauthor CS
features	88.68 ± 0.00	71.66 ± 0.00	74.43 ± 0.00	85.47 ± 0.00
node2vec	98.66 ± 0.04	**99.45 ± 0.01**	**99.34 ± 0.01**	**99.83 ± 0.00**
+ Concat	**98.88 ± 0.03**	99.45 ± 0.01	99.34 ± 0.01	99.83 ± 0.00
+ ConcatPCA	98.40 ± 0.02	99.45 ± 0.01	99.35 ± 0.01	99.84 ± 0.00
+ MLP	98.59 ± 0.09	99.45 ± 0.02	99.40 ± 0.02	99.77 ± 0.02
+ $GERF_{(grid)}$	98.71 ± 0.03	99.45 ± 0.01	99.34 ± 0.01	99.83 ± 0.00
+ $GERF_{(uniform)}$	97.82 ± 0.02	97.66 ± 0.01	97.22 ± 0.02	98.89 ± 0.01
+ $GERF_{(homophily)}$	97.82 ± 0.02	97.66 ± 0.01	97.23 ± 0.02	98.90 ± 0.01
LINE	98.50 ± 0.03	**99.05 ± 0.05**	**99.33 ± 0.02**	**99.95 ± 0.00**
+ Concat	**98.65 ± 0.03**	99.05 ± 0.05	99.33 ± 0.02	99.95 ± 0.00
+ ConcatPCA	97.50 ± 0.05	98.22 ± 0.12	98.98 ± 0.06	99.97 ± 0.00
+ MLP	98.64 ± 0.03	98.97 ± 0.06	99.31 ± 0.03	99.97 ± 0.00
+ $GERF_{(grid)}$	98.56 ± 0.03	99.05 ± 0.05	99.33 ± 0.02	99.95 ± 0.00
+ $GERF_{(uniform)}$	97.68 ± 0.07	96.12 ± 0.17	96.41 ± 0.14	99.02 ± 0.03
+ $GERF_{(homophily)}$	97.68 ± 0.07	96.13 ± 0.17	96.41 ± 0.14	99.03 ± 0.03
SDNE	88.65 ± 0.80	86.19 ± 0.36	85.29 ± 0.59	68.50 ± 0.61
+ Concat	**94.59 ± 0.33**	86.19 ± 0.36	85.29 ± 0.59	68.60 ± 0.61
+ ConcatPCA	88.74 ± 0.48	84.06 ± 0.49	84.82 ± 0.53	**72.11 ± 0.35**
+ MLP	91.75 ± 0.47	**87.37 ± 0.37**	85.73 ± 0.47	67.68 ± 1.15
+ $GERF_{(grid)}$	90.90 ± 0.44	86.43 ± 0.35	**85.80 ± 0.71**	68.50 ± 0.61
+ $GERF_{(uniform)}$	90.79 ± 0.43	85.68 ± 0.37	84.57 ± 0.93	63.17 ± 0.89
+ $GERF_{(homophily)}$	90.79 ± 0.43	85.68 ± 0.37	84.58 ± 0.93	63.17 ± 0.89
TADW	80.17 ± 0.00	56.24 ± 0.00	54.79 ± 0.00	67.81 ± 0.00
FSCNMF	72.69 ± 0.00	81.69 ± 0.00	72.29 ± 0.00	92.91 ± 0.00
DGI	94.08 ± 0.15	79.32 ± 3.89	87.88 ± 1.29	88.09 ± 1.02

7. Link prediction

Setup. We employ a similar setup as in the node classification case. We evaluate embedding vectors returned by the attributed representation learning methods, structural embeddings (themselves and enhanced by the GERF method and baselines), as well as the node attribute vectors (*features*). We take all edges in a graph and assign a positive label to them (class 1). Next, we sample a negative edge for each existing edge and assign a negative label to them (class 0). Such a dataset is split into a train, validation and test split using a stratified 10%/10%/80% ratio. To obtain edge-level embedding vectors from the existing node representations, we use the Hadamard operator (as proposed in [7]). We train a L_2 regularized logistic regression (from the *scikit-learn* package [28]) on the edge embedding vectors (input) and the class information (output) – binary classification. The maximum number of iterations was set to 250, other parameters were left with their default values.

The classification results in terms of the AUC metric are shown in Table 4. For each of the three structural node embedding methods – node2vec, LINE and SDNE – as well as the embeddings updated by the baselines and the GERF method – the best result is marked in bold. We report both the mean and standard deviation over the 10 embedding recalculations.

Please note that as the hyperparmeter estimation is a crucial part of the improved GERF method, we perform this estimation on the link prediction task.

Discussion. Similarly to the node classification experiment, the node attributes itself (*features*) already provide satisfactory performance. However, the structural node embeddings (node2vec, LINE, SDNE) are significantly better (or on par — in the case of SDNE). Let us note that the main objective of structural node embedding methods is to reflect the *structure* of the graph, i.e. encode the edges. Hence, the results are not surprising. In each group, the structural embeddings yield the best performance, whereas adding node attributes does not result in any performance boost. From the perspective of the link prediction task, the node attributes do not provide any useful information — which is confirmed by the *Concat* baseline. Consequently, the GERF method has no chance to improve the embedding vectors. These conclusions are true for the node2vec and LINE methods. However, for SDNE, the addition of node attributes exhibits a performance boost. For Amazon-Photo, the GERF method yields the best results. We can conclude that for already good-performing structural embeddings (in the link prediction task), we should not expect any improvement when adding node attributes, but if the embeddings are not performing that well, it might be worth trying to add additional information to the embedding vectors.

8. Conclusions

In this paper, we take a deeper perspective at the Graph Embedding Retrofitting method. Its main objective is to update already precomputed structural node embeddings according to node attributes so that the resulting embedding will preserve the information from both the structure and attributes. We propose two improvements to the GERF method, i.e., (1) we simplify the objective function, and (2) we propose a method for automatic hyperparameter estimation. Moreover, we extend the experimental evaluation and add: (1) a comparison of GERF's hyperparameter estimation methods, (2) a link prediction study. We find that the improved GERF method allows for the enhancement of structural embeddings and results in better downstream node classification performance. In the case of link prediction, the method is limited by the initial performance of the structural embeddings. In the future, we want to extend the GERF method to enable it to handle dynamic graphs as well as provide the notion of density-based attribute neighborhood handling.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was financed by (1) the Polish Ministry of Education and Science, CLARIN-PL, Poland; (2) the European Regional Development Fund as a part of the 2014–2020 Smart Growth Operational Programme, CLARIN — Common Language Resources and Technology Infrastructure, project no. POIR.04.02.00-00C002/19; (3) the statutory funds of the Department of Artificial Intelligence, Wroclaw University of Science and Technology, Poland; (4) European Union under the Horizon Europe grant OMINO (grant number 101086321). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Executive Agency. Neither the European Union nor European Research Executive Agency can be held responsible for them.

References

[1] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: Y. Bengio, Y. LeCun (Eds.), 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings, 2013.

[2] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Trans. Assoc. Comput. Linguist. 5 (2017) 135–146.

[3] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 770–778.

[4] M. Tan, Q. Le, EfficientNet: Rethinking model scaling for convolutional neural networks, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, 97, PMLR, 2019, pp. 6105–6114.

[5] P. Bielak, D. Puchalska, T. Kajdanowicz, Retrofitting structural graph embeddings with node attribute information, in: D. Groen, C. de Mulatier, M. Paszynski, V.V. Krzhizhanovskaya, J.J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2022, Springer International Publishing, Cham, 2022, pp. 178–191.

[6] B. Perozzi, R. Al-Rfou, S. Skiena, DeepWalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, ACM, 2014, pp. 701–710.

[7] A. Grover, J. Leskovec, Node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 855–864, http://dx.doi.org/10.1145/2939672.2939754.

[8] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, LINE: Large-scale information network embedding, Proceedings of the 24th International Conference on World Wide Web (2015) http://dx.doi.org/10.1145/2736277.2741093, http://dx.doi.org/10.1145/2736277.2741093.

[9] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 1225–1234, http://dx.doi.org/10.1145/2939672.2939753.

[10] C. Yang, Z. Liu, D. Zhao, M. Sun, E.Y. Chang, Network representation learning with rich text information, in: Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI '15, AAAI Press, 2015, pp. 2111–2117.

[11] S. Bandyopadhyay, H. Kara, A. Kannan, M. Murty, FSCNMF: Fusing Structure and Content via Non-negative Matrix Factorization for Embedding Information Networks, 2018.

[12] H. Gao, H. Huang, Deep attributed network embedding, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, 2018, pp. 3364–3370, http://dx.doi.org/10.24963/ijcai.2018/467.

[13] Z. Zhang, H. Yang, J. Bu, S. Zhou, P. Yu, J. Zhang, M. Ester, C. Wang, ANRL: Attributed network representation learning via deep neural networks, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 3155–3161, http://dx.doi.org/10.24963/ijcai.2018/438.

[14] I. Oluigbo, H. Seba, M. Haddad, Improving node embedding by a compact neighborhood representation, Neural Comput. Appl. (2022) 1–14, http://dx.doi.org/10.1007/s00521-022-08076-6.

[15] D. Zhang, J. Yin, X. Zhu, C. Zhang, Attributed network embedding via subspace discovery, Data Min. Knowl. Discov. 33 (6) (2019) 1953–1980, http://dx.doi.org/10.1007/s10618-019-00650-2.

[16] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv, 2017, http://dx.doi.org/10.48550/arXiv.1609.02907, http://arxiv.org/abs/1609.02907, arXiv:1609.02907 [cs, stat].

[17] W.L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS '17, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 1025–1035.

[18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph Attention Networks, 2018, arXiv http://arxiv.org/abs/1710.10903 arXiv:1710.10903 [cs, stat].

[19] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, 2019, http://arxiv.org/abs/1810.00826 arXiv:1810.00826 [cs, stat].

[20] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[21] P. Bielak, K. Tagowski, M. Falkiewicz, T. Kajdanowicz, N.V. Chawla, FILDNE: A framework for incremental learning of dynamic networks embeddings, Knowl.-Based Syst. 236 (2022) 107453, http://dx.doi.org/10.1016/j.knosys.2021.107453, https://www.sciencedirect.com/science/article/pii/S0950705121007152.

[22] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.

[23] PyTorch geometric main page, https://pytorch-geometric.readthedocs.io/en/latest/index.html.

[24] P. Mernyei, C. Cangea, Wiki-CS: A wikipedia-based benchmark for graph neural networks, 2020, arXiv preprint arXiv:2007.02901.

[25] J. McAuley, C. Targett, Q. Shi, A. van den Hengel, Image-based recommendations on styles and substitutes, in: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Association for Computing Machinery, 2015, pp. 43–52, http://dx.doi.org/10.1145/2766462.2767755.

[26] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J.P. Hsu, K. Wang, An overview of microsoft academic service (MAS) and applications, in: Proceedings of the 24th International Conference on World Wide Web, in: WWW '15 Companion, Association for Computing Machinery, New York, NY, USA, 2015, pp. 243–246, http://dx.doi.org/10.1145/2740908.2742839.

[27] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

Bibliography

[1] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211 – 230, 2003.

[2] C. Aggarwal, G. He, and P. Zhao. Edge classification in networks. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1038–1049, May 2016.

[3] C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM Comput. Surv.*, 47(1):10:1–10:36, May 2014.

[4] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 37–48, New York, NY, USA, 2013. ACM.

[5] Y. Anava, N. Avigdor-Elgrabli, and I. Gamzu. Improved theoretical and practical guarantees for chromatic correlation clustering. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 55–65, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[6] A. Azran. The rendezvous algorithm: Multiclass semi-supervised learning with markov random walks. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 49–56, New York, NY, USA, 2007. ACM.

[7] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. Video suggestion and discovery for youtube: Taking random walks through the view graph. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 895–904, New York, NY, USA, 2008. ACM.

[8] A. Baptista, R. J. Sánchez-García, A. Baudot, and G. Bianconi. Zoo guide to network embedding, 2023.

[9] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 585–591, Cambridge, MA, USA, 2001. MIT Press.

[10] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, aug 2013.

[11] F. Béres, D. M. Kelen, R. Pálovics, and A. A. Benczúr. Node embeddings in dynamic graphs. *Applied Network Science*, 4(1):64, 2019.

[12] S. Bhagat, G. Cormode, and S. Muthukrishnan. Node classification in social networks. *CoRR*, abs/1101.3291, 2011.

[13] S. Bhagat, I. Rozenbaum, and G. Cormode. Applying link-based classification to label blogs. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, WebKDD/SNA-KDD '07, pages 92–101, New York, NY, USA, 2007. ACM.

[14] F. Bonchi, A. Gionis, F. Gullo, and A. Ukkonen. Chromatic correlation clustering. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1321–1329, New York, NY, USA, 2012. ACM.

[15] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 891–900, New York, NY, USA, 2015. ACM.

[16] S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 1145–1152. AAAI Press, 2016.

[17] I. Chami, R. Ying, C. Ré, and J. Leskovec. Hyperbolic graph convolutional neural networks, 2019.

[18] H. Chen, B. Perozzi, Y. Hu, and S. Skiena. Harp: Hierarchical representation learning for networks. *The 32nd AAAI Conference on Artificial Intelligence*, 06 2017.

[19] X. Chen and K. He. Exploring simple siamese representation learning, 2020.

[20] C. Chevalier and I. Safro. Learning and intelligent optimization. In T. Stützle, editor, *Learning and Intelligent Optimization*, chapter Comparison of Coarsening Schemes for Multilevel Graph Partitioning, pages 191–205. Springer-Verlag, Berlin, Heidelberg, 2009.

[21] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, dec 2014.

[22] A. Clauset, C. Moore, and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, May 2008.

[23] Q. Dai, Q. Li, J. Tang, and D. Wang. Adversarial network embedding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[24] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux. Pick-a-crowd: Tell me what you like, and i'll tell you what to do. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 367–374, New York, NY, USA, 2013. ACM.

[25] C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 107–114, Washington, DC, USA, 2001. IEEE Computer Society.

[26] P. Eades and R. Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical report, Brown University, Providence, RI, USA, 1988.

[27] L. Ericsson, H. Gouk, C. C. Loy, and T. M. Hospedales. Self-supervised representation learning: Introduction, advances, and challenges. *IEEE Signal Processing Magazine*, 39(3):42–62, 2022.

[28] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 123–133, New York, NY, USA, 1991. ACM.

[29] F. Fouss, A. Pirotte, J. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369, March 2007.

[30] L. C. Freeman. Visualizing social networks. *Journal of Social Structure*, 1, 2000.

[31] E. Gansner and S. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience - SPE*, 30, 01 1997.

[32] H. Gao, J. Pei, and H. Huang. Progan: Network embedding via proximity generative adversarial network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 1308–1316, New York, NY, USA, 2019. Association for Computing Machinery.

[33] V. Garcia Satorras, E. Hoogeboom, F. Fuchs, I. Posner, and M. Welling. E(n) equivariant normalizing flows. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4181–4192. Curran Associates, Inc., 2021.

[34] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. *Learning Probabilistic Relational Models*, pages 307–335. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[35] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[36] L. Gong and Q. Cheng. Adaptive edge features guided graph attention networks. *CoRR*, abs/1809.02709, 2018.

[37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[38] P. Goyal, S. R. Chhetri, and A. Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *CoRR*, abs/1809.02657, 2018.

[39] P. Goyal, N. Kamra, X. He, and Y. Liu. Dyngem: Deep embedding method for dynamic graphs. *CoRR*, abs/1805.11273, 2018.

[40] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent - a new approach to self-supervised learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21271–21284. Curran Associates, Inc., 2020.

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

[42] S. Gugger and J. Howard. Adamw and super-convergence is now the fastest way to train neural nets., 2018.

[43] P. Gupta, V. Satuluri, A. Grewal, S. Gurumurthy, V. Zhabiuk, Q. Li, and J. Lin. Real-time twitter recommendation: Online motif detection in large dynamic graphs. *Proc. VLDB Endow.*, 7(13):1379–1380, Aug. 2014.

[44] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[45] M. A. Hasan and M. J. Zaki. *A Survey of Link Prediction in Social Networks*, pages 243–275. Springer US, Boston, MA, 2011.

[46] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133. Curran Associates, Inc., 2020.

[47] Jianchao Yang, Shuicheng Yang, Yun Fu, Xuelong Li, and T. Huang. Non-negative graph embedding. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[48] D. Jiang, Z. Wu, C.-Y. Hsieh, G. Chen, B. Liao, Z. Wang, C. Shen, D. Cao, J. Wu, and T. Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of Cheminformatics*, 13(1):12, Feb 2021.

[49] I. T. Jolliffe. *Principal Component Analysis and Factor Analysis*, pages 115–128. Springer New York, New York, NY, 1986.

[50] W. Ju, Z. Fang, Y. Gu, Z. Liu, Q. Long, Z. Qiao, Y. Qin, J. Shen, F. Sun, Z. Xiao, J. Yang, J. Yuan, Y. Zhao, X. Luo, and M. Zhang. A comprehensive survey on deep graph representation learning, 2023.

[51] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, Mar 1953.

[52] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013. cite arxiv:1312.6114.

[53] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems, 2018.

[54] T. Kipf and M. Welling. Variational graph auto-encoders. *ArXiv*, abs/1611.07308, 2016.

[55] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[56] P. Kolyvakis, A. Kalousis, and D. Kiritsis. Hyperkg: Hyperbolic knowledge graph embeddings for knowledge base completion, 2019.

[57] J. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Publications, 1978.

[58] O. Küçüktunç, E. Saule, K. Kaya, and U. V. Çatalyürek. Diversified recommendation on graphs: Pitfalls, measures, and algorithms. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 715–726, New York, NY, USA, 2013. ACM.

[59] J. B. Lee, G. H. Nguyen, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. Temporal network representation learning. *CoRR*, abs/1904.06449, 2019.

[60] C. Li, J. Ma, X. Guo, and Q. Mei. Deepcas: An end-to-end predictor of information cascades. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 577–586, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[61] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, pages 556–559, New York, NY, USA, 2003. ACM.

[62] C. Liu, W. Fan, Y. Liu, J. Li, H. Li, H. Liu, J. Tang, and Q. Li. Generative diffusion models on graphs: Methods and applications, 2023.

[63] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky. Graph normalizing flows. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[64] Y. Liu, M. Jin, S. Pan, C. Zhou, Y. Zheng, F. Xia, and P. S. Yu. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(6):5879–5900, 2023.

[65] C. F. V. Loan. Generalizing the singular value decomposition. *SIAM Journal on Numerical Analysis*, 13:76–83, 1976.

[66] Q. Lu and L. Getoor. Link-based classification. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 496–503. AAAI Press, 2003.

[67] D. Luo, C. Ding, F. Nie, and H. Huang. Cauchy graph embedding. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 553–560, USA, 2011. Omnipress.

[68] L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150 – 1170, 2011.

[69] T. Man, H. Shen, S. Liu, X. Jin, and X. Cheng. Predict anchor links across social networks via an embedding approach. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 1823–1829. AAAI Press, 2016.

[70] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[71] A. McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20, May 2014.

[72] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[73] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 419–432, New York, NY, USA, 2008. ACM.

[74] J. Neville and D. Jensen. Iterative classification in relational data. In *In Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20. AAAI Press, 2000.

[75] M. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27:39–54, 10 2003.

[76] M. E. Newman. The structure of scientific collaboration networks. *Proceedings of the national academy of sciences*, 98(2):404–409, 2001.

[77] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1105–1114, New York, NY, USA, 2016. ACM.

[78] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710, New York, NY, USA, 2014. ACM.

[79] B. Perozzi, V. Kulkarni, and S. Skiena. Walklets: Multiscale graph embeddings for interpretable network classification. *CoRR*, abs/1605.02115, 2016.

[80] J. Rissanen. Paper: Modeling by shortest data description. *Automatica*, 14(5):465–471, Sept. 1978.

[81] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326, 2000.

[82] C. D. Sa, A. Gu, C. Ré, and F. Sala. Representation tradeoffs for hyperbolic embeddings. *CoRR*, abs/1804.03329, 2018.

[83] B. Shaw and T. Jebara. Structure preserving embedding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 937–944, New York, NY, USA, 2009. ACM.

[84] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, Aug. 2000.

[85] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. *CoRR*, abs/1704.02901, 2017.

[86] U. Singer, I. Guy, and K. Radinsky. Node embedding over temporal graphs. *CoRR*, abs/1903.08889, 2019.

[87] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1067–1077, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[88] J. Tenenbaum, V. Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 01 2000.

[89] S. Thakoor, C. Tallec, M. G. Azar, R. Munos, P. Veličković, and M. Valko. Bootstrapped representation learning on graphs. In *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, 2021.

[90] A. Theocharidis, S. Dongen, A. Enright, and T. Freeman. Network visualisation and analysis of gene expression data using biolayout. *Nature protocols*, 4:1535–50, 10 2009.

[91] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 567–580, New York, NY, USA, 2008. ACM.

[92] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 965–973, New York, NY, USA, 2011. ACM.

[93] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[94] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[95] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep Graph Infomax. In *International Conference on Learning Representations*, 2019.

[96] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2017.

[97] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1225–1234, New York, NY, USA, 2016. ACM.

[98] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo. Graphgan: Graph representation learning with generative adversarial nets. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*

and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[99] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. Community preserving network embedding. In *AAAI*, pages 203–209. AAAI Press, 2017.

[100] Y. Wang, Z. Li, and A. B. Farimani. Graph neural networks for molecules, 2023.

[101] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018.

[102] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998.

[103] H. White, S. Boorman, and R. Breiger. Social structure from multiple networks. i. blockmodels of roles and positions. *American Journal of Sociology*, 81(4):730–780, 1976.

[104] S. D. Winter, T. Decuypere, S. Mitrovic, B. Baesens, and J. D. Weerdt. Combining temporal aspects of dynamic networks with node2vec for a more efficient dynamic link prediction. In *IEEE/ACM 2018 International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2018, Barcelona, Spain, August 28-31, 2018*, pages 1234–1241, 2018.

[105] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[106] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[107] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. Scan: A structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 824–833, New York, NY, USA, 2007. ACM.

[108] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang. Network representation learning with rich text information. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 2111–2117. AAAI Press, 2015.

[109] S.-H. Yang, B. Long, A. Smola, N. Sadagopan, Z. Zheng, and H. Zha. Like like alike: Joint friendship and interest propagation in social networks. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 537–546, New York, NY, USA, 2011. ACM.

[110] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5812–5823. Curran Associates, Inc., 2020.

[111] K. Yu, W. Chu, S. Yu, V. Tresp, and Z. Xu. Stochastic relational models for discriminative link prediction. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 1553–1560, Cambridge, MA, USA, 2006. MIT Press.

[112] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow twins: Self-supervised learning via redundancy reduction, 2021.

[113] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[114] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *Proc. VLDB Endow.*, 2(1):718–729, Aug. 2009.

[115] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang. Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[116] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang. Graph contrastive learning with adaptive augmentation. *CoRR*, abs/2010.14945, 2020.