# GSM: Inductive Learning on Dynamic Graph Embeddings

Marina Ananyeva[1], Ilya Makarov[1,2⋆], Mikhail Pendiukhov[3]

[1] National Research University Higher School of Economics, 3 Kochnovskiy Proezd, 107207 Moscow, Russia
[2] University of Ljubljana, Faculty of Computer and Information Science, Vecna pot 113, SI-1000 Ljubljana, Slovenia
[3] Analytical Software Solutions LLC, 3 proezd Marinoy Roshchi 40, bld. 1, 127018 Moscow, Russia

ananyeva.me@gmail.com, iamakarov@hse.ru, dx@apsolutions.ru

**Abstract.** In this paper we study the problem of learning graph embeddings for dynamic networks and ability to generalize to unseen nodes called inductive learning. Firstly, we overview the state-of-the-art methods and techniques of constructing graph embeddings and learning algorithms for both transductive and inductive approaches. Secondly, we propose an improved model GSM based on GraphSAGE algorithm and set up the experiments on datasets CORA, Reddit, and HSEcite, which is collected from Scopus citation database across authors with affiliation to NRU HSE in 2011-2017. The results show that our three-layer model with attention-based aggregation function, added normalization layers, regularization (dropout) outperforms suggested by respective authors GraphSAGE models with mean, LSTM and pool aggregation functions, thus giving more insight on possible ways to improve inducting learning model based on GraphSAGE model.

**Keywords:** Graph Embeddings, Dynamic Graphs, Inductive Learning Approach

## 1 Introduction

Real-world networks are full of useful information that can be extracted to help solve complex problems in various application fields. To make this goal feasible, networks have to be transformed into simplified representations called graphs. Graph-based models are used in a wide range of application areas: social network analysis [1–3], trade and financial transactions [4], co-authorship networks [5–7], neural connections in the human brain [8], biochemical protein-protein interactions [9]. There are several conventional ways to operate with graph models, which are based either on the basic data representation structures of the original graph (e.g. adjacency list, incidence or adjacency matrices) or on the graph representation in vector space over real numbers. In this work we are going to focus on the latter approach.

⋆ Corresponding Author

Graph embedding is an effective method to convert the initial graph into a space of lower dimension. In particular, it learns a mapping from a given graph to a vector space and optimize it by finding the best possible option to preserve the network properties and graph structural information. The problem of graph embeddings lies between machine learning task and representation learning. The problem setting can be broadly divided into four categories: (1) node classification, (2) link prediction, (3) visualization, and (4) clustering [10, 11]. In this paper, we set the node classification task. The second research problem aims to obtain the best possible data representations, and it covers supervised, semi-supervised and unsupervised tasks for learning embeddings. Vector representations are more efficient and simple to work with in data science tasks in comparison to raw graphs, which are limited to a certain subset of machine learning and statistical models [12].

Most of the existing embedding approaches are mainly designed for static graphs. In fact, many real-world networks have dynamic nature, e.g. financial transactions flows, reposts graphs in Facebook, citations in arXiv. In these cases, it may seem more reasonable to model dynamic graphs considering its changes over time. These graphs can be called dynamic in terms of either node or edge temporal data or graph structure. In our case we use the second definition. It means that instead of considering the dynamics of the internal changing information of nodes (or edges), we consider that the graph structure can evolve over time periods by acquiring, preserving or losing its nodes and edges. In order to extend the static embedding algorithms, dynamic graphs are often divided into several snapshots that represent the state of the graph at some time point. Transductive algorithms are applied to each snapshot independently, however, the final results still remains unsatisfactory and unfold a promising direction of further scientific studies. Poor performance is mostly explained by the challenges needed to overcome while working with dynamic graphs: scalability, efficiency, stability and flexibility. The following research questions stay open for dynamic case. What are the effective ways to use only local structure for node embeddings instead of the whole graph? How to chose the node's neighborhood? Which orders of proximity should be considered? What are the best practices for aggregating information from neighbors?

In this research we propose an improved model ***GSM*** *(GraphSage attention-Modified)*, one of the most effective models with inductive extension, and test it on the open-source datasets (CORA, Reddit), including our own data collected from Scopus that we called HSEcite. We demonstrate its performance on multilabel classification task. The results of applying our model reveal the significance of deliberate choice of aggregation functions, normalization and regularization for GraphSage-based algorithms. We put the experiments for different types of aggregation functions (mean, max pooling, LSTM, attention-based), apply normalization and dropout (regularization) for layers.

In **Section 2** we give the basic definitions and formalize the problem. **Section 3** provides an overview of transductive graph embedding methods. The CNN, GAE and most notable algorithms for inductive learning embeddings on

dynamic graphs are listed in **Section 4**. Finally, **Section 5** contains the set up of experiments and discussion of the results.

## 2 Notation and basic definitions

First, we denote the basic definitions. Let us assume the input of representation learning algorithm being an undirected graph denoted as follows:

**Definition 1.** *Graph is an unordered pair $G = (V, E)$, where $V$ is a set of vertices $v \in V$, and $E$ is a set of edges $e \in E$.*

Matrix $A$ will be associated with binary adjacency matrix of size $n$ x $n$. We also make an assumption that the algorithms can take an input of a real-valued matrix of vertex attributes which represent its metadata $X \in \mathbb{R}^{k \cdot |V|}$, where $k$ is a number of attributes. Hence, the goal of embedding is to leverage the information from matrices A and X to map each vertex into a vector $\boldsymbol{v} \in \mathbb{R}^d$, $d << |V|$. We denote the first-order and second-order proximities the same way as Cai et al. [12]:

**Definition 2.** *The first-order proximity $s_{ij}^{(1)}$ between vertices $i$ and $j$ is called the weight of the edge $e_{ij}$, which in simple case equals $A_{i,j}$.*

Hence, let $s_i^{(1)} = [s_{i,1}^{(1)}, s_{i,2}^{(1)}, \ldots, s_{i,|V|}^{(1)}]$ define the first-order proximity between $v_i$ and other vertices. The more two connected vertices are similar, the greater is the weight of their common edge.

**Definition 3.** *The second-order proximity $s_{ij}^{(2)}$ between vertices $i$ and $j$ is a similarity between $s_i^{(1)}$ and $s_j^{(1)}$, $is$ and $js$ neighbourhoods.*

For comparison of nodes' neighbourhoods $s_i^{(1)}$ and $s_j^{(1)}$ cosine similarity, Jaccard index or any other applicable measure can be considered. The proximity between two nodes equals to zero in case they do not share common neighbours. The above-mentioned definitions of proximities will help us to express the meaning of embedding.

**Definition 4.** *Graph embedding learns a mapping $f : i \rightarrow \boldsymbol{i} \in \mathbb{R}^d$, where $d << |V|$. The function's objective is to produce the similarity between $\boldsymbol{i}$ and $\boldsymbol{j}$ preserving the first- and second-order proximities of nodes $i$ and $j$ as much as possible.*

Formally, Leskovec J. and Zitnik M. [13] defined the following steps:

1. Denote an encoder (such function $\phi(u)$ that maps a node $u$ into a d-dimen–sional vector $\boldsymbol{u}$ ).
2. Define a similarity function for nodes of initial graph.
3. Optimize parameters of the encoder: $similarity(u, v) \approx \phi_u^T \phi_v$.

We expect the vertices which are close to each other in the original graph get a close representation in the vector space. Generally, embeddings methods use various notions of closeness between nodes: connectivity, common neighbours, similarity of local structure. For instance, if we assume the weight of an edge as a good measure of proximity, it approximately equals to the scalar product of nodes' embeddings. Hence, the angle between vectors of close nodes should be minimal. The loss function will take the form:

$$L = \sum_{(u,v) \in VxV} ||\phi_u^T \phi_v - A_{u,v}||^2.$$

Alternatively, we can define the node's higher-order neighbourhoods (k-hop neighbours). We discuss these differences in detail in subsection dedicated to random walk-based methods. We also have to provide the essential definitions for generalization to the dynamic case.

**Definition 5.** *Dynamic graph is a sequence of static graphs over discrete time steps called snapshots, i.e.* $G = \{G_1, \ldots, G_T\}$*, where* $G_t = (V_t, E_t)$*,* $T$ *– the number of snapshots.*

We assume the set up with expanding graphs, allowing new vertices to join the dynamic graph $V_t \subseteq V_{t+1}$ and to add edges to existing vertices $E_t \subseteq E_{t+1}$. The disappeared vertices and edges can remain as part of the graph with zero weight to others, but we use the datasets with evolving structures with all previous connections included. By considering embeddings on dynamic graph, we extend the notion for dynamic graph embedding.

**Definition 6.** *Dynamic graph embedding is a time-series of mapping functions* $F = \{f_1, , f_T\}$ *on dynamic graph* $G = \{G_1, \ldots, G_T\}$*, such that* $f_t$ *corresponds to a graph embedding for* $G_t$ *and all mappings preserve the proximity measure respectively to their graphs.*

Most of the graph-based methods optimize the mappings as *unsupervised* task, using only the information from matrices A and X and generating scores based on sampled paths or node neighbours. Other models use node labels for optimization of embeddings during *supervised* representation learning. In *semi-supervised* tasks the data contains labeled and mostly unlabeled instances. Let $L$ and $L^*$ be the numbers of labeled and unlabeled objects. Then, we define $[v_1 : v_L] = [v_1, v_2, \ldots, v_L]$ and $[v_{L+1} : v_{L+L^*}]$ as feature vectors of labeled and unlabeled objects. The labels are $[y_1 : y_L]$. Based on given sample, the goal is to learn a classifier $f : v \rightarrow \boldsymbol{v}$ and to use the labeled objects to improve the overall performance based only on small labeled set. The hypothesis in graph-based semi-supervised learning implies that nearby nodes tend to have the same labels. Transductive learning applies this classifier $f$ on unlabeled instances observed at training time. Inductive learning try to learn a parameterized classifier which can be generalized on unseen instances. In this research, we are interested in more complicated inductive case.

Considering evolving network, a sample above includes only observed objects called out-of-sample nodes, for which we want to infer the learned embeddings. For instance, Hamilton et al. [14] suggested inductive node classification for texts and protein-protein interactions.

The above basic concepts and definitions will be useful for a better understanding of the descriptions of the algorithms in the following sections.

## 3 Related research

Transductive methods imply that nodes or edges can be predicted only for ones observed during training time and do not naturally generalize to unseen instances. The disadvantages of such kind of task generalization are computational inefficiency, especially for large graphs. A brief review of existing methods divided to matrix factorization-based methods, random-walk based algorithms, and deep learning architectures for graphs is presented based on [10–12].

### 3.1 Matrix Factorization-based methods

Embedding based on the matrix factorization approach uses the properties of the graph represented in a matrix form, e.g. node pairwise similarity, and is aimed at decomposition of this matrix into the product of others to get node embedding. In most cases, the algorithm's input has to be a graph from non-relational data features in high dimensional space. One can simply use a column vector or row vector of adjacency matrix as the vector representation of nodes, but the representation space will be $N$-dimensional, where $N$ is the number of nodes in a graph. Therefore, the goal is to form and learn low-rank vector space for the initial matrix preserving the network's properties. The most common matrix factorization-based methods for embeddings are ***Singular Value Decomposition*** [15, 16], ***Non-negative matrix factorization*** [17], ***Locally Linear Embedding*** [18]. ***Laplacian Eigenmaps*** approach try to preserve local distances and learn manifold structure.The disadvatange of such methods is that it cannot be applied to large graphs, because they operate on dense matrices. The generated network representations are obtained through factorizing the Laplacian matrix of the adjacency matrix, therefore it exploits only the first-order proximity and demonstrates the importance of second-order proximity, which help preserving network structure. **HOPE** algorithm preserves high-order proximity and enables to transform the original SVD problem to a generalized one, but it also requires the whole graph matrix [16]. **GraRep** is another factorization method taking into account local and global structural information [19]. It uses SVD and transformed transition probability matrix of DeepWalk, concatenating the final representations. However, it is not scalable and also requires the whole graph matrix. **LINE** [20] preserves the first- and second-order proximities separately and trains the embeddings by negative sampling, concatenating the obtained representations, which is a sub-optimal solution. Nevertheless, LINE adopts shallow structure, for which it is difficult to capture the highly non-linear graph structure in the network.

### 3.2 Random walk-based methods

The idea of random walk based methods is to define a similarity based on stochastically denoted higher-order neighborhoods of nodes. For un–supervised feature learning task we learn node embeddings such that nodes nearby are close to each other, preserving similarity from initial graph in d-dimensional vector space. The advantages of random walk implementation are the efficiency and flexibility, because we take only a part of node pairs with probabilities of co-occurrences for training set and provide stochastic definition of similarity [13]. Given any node $u$, we learn its feature representation $\phi(u)$ predicted by closest nodes from $N(u)$ – target node's k-hop neighborhood. Hence, the goal is to find the embedding of $\phi_u$ such that it predicts close nodes from neighborhood obtained via random walk simulation.

Among the most frequently used algorithms is **DeepWalk** [21], which holds an idea to use unbiased random walks of fixed-length starting from each node and creates a matrix of $d$-dimensional node embeddings using the SkipGram algorithm. SkipGram is applied to the set of random walks maximizing the probability of nodes neighbourhood conditioned by node's embedding. In this way, nodes with similar neighbourhoods (having large second-order proximity values) share similar embedding. For input it takes $G(V, E)$, window size $w$, embedding size $d$, the number of walk for each node $\gamma$ and walk length $t$. As a result we obtain a matrix of vertex representations in $R^{|V|*d}$. Although it shows a good performance on different network datasets, it does not clarify the definition of objective function for preserving graph structural information and is prone to keep only the second-order node proximity. Recently, there were suggested modifications of DeepWalk, e.g. *Max-Margin DeepWalk* [22]– a semi-supervised model that jointly optimizes the max-margin classifier and the social representation learning, also holding discriminative characteristics.Another well-known approach is **Node2Vec** introduced by Grover and Leskovec [23]. The key point was to use flexible and biased random walks, searching for trade-off between exploration of global and local network properties. Thus, authors suggested to define two parameters: p – return back to the previous node, q – moving outwards or inwards, the strategies of DFS and BFS. For the input of learning features algorithm we initialize a graph $G = (V, E, W)$, $d$ dimensions, walk length $l$, $r$ walks per node, context size $k$, and probabilities of return ($p$) and In-out ($q$). As DeepWalk, this model is scalable and local (does not require the entire graph), nevertheless, it is a hyperparameter-supervised approach, extending DeepWalk by introducing two parameters to control random walk sampling, $p$ and $q$. Given $p = 1$, $q = 1$, we get back to DeepWalk. In addition, DeepWalk use hierarchical softmax, while Node2Vec is based on negative sampling.

As alternative approach to random walks the diffusion simulations was suggested for graphs. Random walks tend to suffer from producing extra information by revisiting same node several times, slow spread across the network and inefficient generation of proximity statistics [24]. Instead, we apply diffusion-like process to extract a subgraph of the node's neighbors – diffusion graph. Then, we find Euler tour to use it as sequence with more complete information on local

k-hop neighborhood and all adjacency in the graph in comparison to random walk methods. This approach was named **Diff2Vec** [24]. Fast Sequence Based Embed–ding is a further extension of Diff2Vec – it uses the sequences from Diff2Vec and use it in the input of single-layer neural network with $d$ neurons for learning $d$-dimensional embedding.

### 3.3 Graph convolutional networks and GAE

Convolutional networks and its modifications for the graphs have been widely adopted for learning graph embeddings. In general, the differences in approaches lie in the ways they formulate a similar to image convolution operation for working on graphs: to define the convolution in the spectral domain or treat it as neighborhood matching in the spatial set. Concerning the autoencoders - it is unsupervised models which are composed of two parts, i.e. the encoder and decoder. The autoencoders aim to minimize the loss function as a difference between input and output representation while intermediately reducing the data dimension. In terms of graph embeddings adopting autoencoder models means their usage for proximity matrix factorization, for e.g., adjacency matrix factorization for the reconstruction process. Such an autoencoder will also make the nodes with similar neighbourhood sets have similar embeddings. The following deep learning models are popular for graph embeddings learning: **EmbedNN** [25], **SDNE** [26].

## 4 Inductive learning embeddings on dynamic graphs

Most part of the existing methods requires the whole graph with all nodes for learning the embeddings, because otherwise they cannot generalize on to unseen instances. Inductive methods try to overcome this problem, which is especially relevant for large networks evolving in time. The following techniques are mentioned and used in the scientific research papers: (1) application of static embedding algorithms to each snapshot of the dynamic graph and rotational alignment of the resulting embeddings across all time steps [14]; (2) explicit imposing of temporal regularizer in order to ensure temporal smoothness across embeddings from time snapshots; (3) information propagation [27]; (4) loss optimization, which encourages smooth changes between vertices with edges; (5) learning a mapping from node's features by imposing a manifold regularizer obtained from the graph [28]. We are going to list the most effective methods for inductive learning embeddings problem: **Planetoid**, **DynGEM** [29], **Graph2Gauss**, **DepthLGP** [30].

**GraphSAGE** [14] is a method that generates node embeddings by sampling node neighborhood and aggregating attributes from its neighbors whil providing inductive framework supporting node feature usage to efficiently generate graph embeddings for previously unseen data.

For each vertex $v \in V$ we aggregate the information from its neighbors $u \in N(v)$ and itself:

$$h_v^k = \sigma([W_k * AGG(h_u^{k-1}, \forall v \in N(v)), B_k h_v^{k-1}]),$$

where $AGG$ is a generalized differentiable aggregation function. The details of this algorithm will be regarded in the next sections.

## 5 Experimental setup

In the experiments we tested the performance of GraphSAGE model with different aggregation function against our model GSM in three tasks: (1) the classification of academic articles of HSE into 25 scientific research areas using Scopus citation database; (2) classification of research papers on Machine Learning topics into 7 sub-categories (CORA dataset); (3) classification of posts placed on Reddit into 41 different communities. For all experiments we make the predictions for unseen nodes that were not used during the model's supervised training.

### 5.1 Baseline

We selected three models as baselines for the empirical results of inductive benchmark: GraphSage with mean, pooling and LSTM aggregation functions. In Graph-Sage modified model we used attention-based aggregation function, added normalization of the two last layers, dropout for the network's regularization and Adam optimization method instead of stochastic gradient descent (SGD). The cross-entropy was used as the loss function for supervised training. The authors of GraphSAGE used rectified linear units (ReLU) as the non-linearity function, K = 2 and neighborhood's samples with sizes 25 and 10. In our experi–ments the best results were obtained for LeakyReLU non-linearity function, implemented in PyTorch python module, K = 3 and neighborhood's samples with sizes 15, 10 and 5. As shown in the table of results, the increase of the depth by one unit did not significantly affected the results for CORA and Reddit datasets, but it worked for HSEcite data. The use of one more 'aggregator-encoder' bunch was helpful for the case, when feature matrix is too sparse and the number of features is significantly more than objects. For providing a fair comparison of results, all models shares had the same loss function, the way to sample the neighborhood, and the number of minibatch iterations. The final set of hyperparameters' values was formed on early stage through validation tests on the subsets of CORA, HSEcite and Reddit data, that were discarded from the further analysis.

### 5.2 Datasets

The experiments were conducted on three evolving graphs, which represents citation and social networks.

**HSEcite**. We collected our own dataset from Scopus citation database, gathering all papers, where the author's affiliation organization was National Re–search University Higher School of Economics for the time period 2011-2017.

The dataset consists of 6279 unique articles and 21 601 edges, which indicate that one paper cited another. It forms undirected and unweighted graph, containing 27836 features for each paper. First, we formed the dictionary of all keywords mentioned in the articles, and then transformed it to a binarized vector for each paper, where zero value means the absence of keyword in the article. In total, there are 25 scientific fields, which were coded from categorical to numerical values and are used as labels in this dataset. We want to predict paper subject categories in a multilabel classification task.

**Cora.** The dataset which represents a citation network of Machine Learning papers. It includes the labels of seven categories: Genetic Algorithms, Theory, Case Based, Neural Networks, Probabilistic Methods, Reinforcement Learning, and Rule Learning. In total, the dataset is the corpus of 2,708 labeled papers and 5,429 directed links, where each article cited or was cited by at least one other article. Besides seven classes, it contains 1,433 features – the number of words used in paper abstracts and stored in a dictionary. Each paper is described by a 0/1-valued word vector. The first file contains the paper's id, word attributes and class label, while the second one: unique id of cited paper and id of citing paper.

**Reddit**. The multi-label dataset contains 232 965 nodes, which represent users posts in online forum Reddit, and 5 376 619 edges. The labels are the communities, so-called 'subreddits', to which the post belong to. This social network can be constructed as a post-to-post graph, where the nodes are connected to each other if the same user left his comments on both posts. For building a graph there were sampled 50 out of the largest communities. The first 20 days were used in training subset, 30% - for validation , and the rest - for testing. The features are also transformed into word vectors, containing the mean embedding of the post's title , the mean embedding of all its comments, scores and the number of comments of the post.

| Dataset | Network | —V— | —E— | Features | Labels |
|---|---|---|---|---|---|
| HSEcite | Citation | 6279 | 21601 | 27836 | 25 |
| CORA | Citation | 2708 | 5429 | 1433 | 7 |
| Reddit | Social | 232965 | 5376619 | 602 | 41 |

**Table 1.** The datasets description

### 5.3 Model framework

In this subsection, we describe the generation of embeddings by modified GraphSAGE algorithm (*Algorithm 1*). Assuming the model was trained and its parameters are fixed, we apply forward propagation algorithm. We suppose to learn the parameters of K aggregation functions ($K = 3$ in our case, while in the architecture of GraphSAGE the two depth layers were used) and the weight matrices $W^k$ for all k for further propagation between model's layers. For each depth layer $k \in K$ the function incrementally aggregates information across neighboring nodes, which is transferred into encoder function. The input of the algorithm takes the whole graph $G(V, E)$, all features $x_v, \forall vinV$. So, $k$ defines the step from the outer loop, $h^k$ stands for a representation of node at the current step. Each node $v$ aggregate the neighbors and the representations of nodes $h_u^{k-1}, \forall u \in N(v)$ into a vector $h_{N(v)}^{k-1}$ . By introducing the subset $V_0 \subset V$ we imply that each vertex can be dropped with probability (e.g. $p = 0.2$). In fact, in the experiments we used the implementation of the dropout procedure in PyTorch and Tensorflow. The aggregation depends on the previous iteration $k-1$ and $k = 0$ by the representations. After completing this step, the algorithm concatenates the aggregated neighbor's vector $h_{N(v)}^{k-1}$ with the last obtained representation of node $h_v^{k-1}$. This vector is transferred into fully connected layer with the use of any non-linear activation function (e.g., we used LeakyReLU). The output of this algorithm is denoted through $z_v$, the final representation.

**Algorithm 1:** GSM embedding generation algorithm (forward propagation)

**Input:** Graph $G(V, E)$; input features $\{x_v, \forall v \in V\}$; depth $K$; dropout probability p; weight matrices $W^k$, $\forall k \in \{1, \ldots, K\}$; non-linearity $\sigma$, attention aggregation function $AGG_k$, $\forall k \in \{1, \ldots, K\}$, neighborhood function $N : v \to 2^V$

**Output:** Vector representations $z_v$ for all $v \in V$

$h_v^0 \leftarrow x_v$, $\forall v \in V$ ;
**for** $k = 1 \ldots K$ **do**
  **if** $k \neq K$ **do**
    $\forall v \in V : P(v \in V_0) = p \to V_0 \subseteq V$ **do**
    $V = V_0$
  **for** $v \in V_0$ **do**
    $h_{N(v)}^0 \leftarrow AGG_k\ (\{h_u^{k-1}, \forall u \in N(v)\})$ ;
    $h_v^k \leftarrow \sigma(W^k \cdot CONCAT(h_v^{k-1}), h_{N(v)}^k)$;
  **end**
  $h_v^k \leftarrow h_v^k / ||h_v^k||_2$, $\forall v \in V$
**end**
$z_v \leftarrow h_v^k \forall v \in V$

The following aggregation functions were proposed and used by the authors of GraphSAGE algorithm for k-th layer:

1. Averaging (the weighted average of the neighbors, a linear approximation of a localized spectral convolution):

$$AGG = \sum_{u \in N(V)} \frac{h_u^{k-1}}{|N(v)|}.$$

2. Pooling (average or maximum value by element, there is no significant difference between mean- and max-pooling):

$$AGG = \gamma(Qh_u^{k-1}, \forall u \in N(v)).$$

3. LSTM (which is not symmetric, but holds larger extensive power):

$$AGG = LSTM([h_u^{k-1}, \forall u \in \pi(N(v))]).$$

We suggest to use attention-based aggregation function instead. Among the benefits of attention is that it gives the opportunity to work with inputs of variable size, focusing on the most important parts of the input. For computing a new representation of a single sequence, it is referred to self-attention [31]. The main idea is to compute the representations of each node, following the self-attention and neighbors attending strategy. Firstly, this operation is efficient, because it can be parallelized across pairs of neighbors. Secondly, it is not limited by node's degree, as far as we can adjust the arbitrary weights of neighbors. Thirdly, it is applicable to inductive learning tasks.

In the experiments the attention component is a single (feedforward) neural network layer with weight vector $\boldsymbol{a} \in R^{2F'}$ and $\sigma$ non-linear activation function (e.g. LeakyReLU). The coefficients can be denoted as:

$$\alpha_{i,j} = \frac{exp(\sigma(\boldsymbol{a}^{T,}[W\boldsymbol{h}_i||W\boldsymbol{h}_j]))}{\sum_{k \in N_i} exp(\sigma(\boldsymbol{a}^{T,}[W\boldsymbol{h}_i||W\boldsymbol{h}_k]))},$$

where $|\cdot|$ - is the concatenation.

The first step is a linear transformation applied to every node, which is parametrized by a weight matrix $W \in R^{F' \cdot F}$, where $F$ and $F'$ are the numbers of features for each node in initial (input) and new(output) feature sets. Next, we compute self-attention on the nodes by calculating attention coefficients, that indicate the so-called importance of node $j-s$ features to another node $i$: $e_{ij} = a(W\boldsymbol{h}_i, W\boldsymbol{h}_j)$, where $a : R^F \cdot R^{F'} \to R$.

### 5.4 Evaluation Metrics

To test the performance of different embedding methods on multilabel classification task we report Micro-$F_1$ and Macro-$F_1$ scores. Micro-$F_1$ calculate metrics globally by counting the total true positives, false positives and false negatives, while Macro-$F_1$ score calculate metrics for each label, and find their unweighted mean. 80%/20% train/test split was used. XGBoost classifier was used in the experiments.

### 5.5 Discussion

In fact, the GSM model with attention-based aggregator function performed the best scores F1-micro and F1-macro on all three datasets (CORA, HSEcite and Reddit), showing better results in comparison to mean, pool and LSTM aggregators. One can observe the greater boost in accuracy for HSEcite dataset, which shows that attention-based aggregator better captures the information from neighboring nodes for this type of graph structure. All models, GraphSage and GSM variants, worked fast on CORA dataset with $K = 2$ and $K = 3$ depths due to the sizes of the feature matrix and the dataset. The runtime for the rest datasets, especially with $K = 3$ depth, was increased with a factor of $5 - 80$x, that depends on the neighborhood size sampling. In fact, the GSM model with attention-based aggregator function performed the best scores F1-micro and F1-macro ($0.881 - 0.873$ on CORA, $0.694 - 0.691$ on HSEcite, $0.918 - 0.911$ on Reddit), showing better results in comparison to mean, pool and LSTM aggregators.

| | CORA | | HSEcite | | Reddit | |
|---|---|---|---|---|---|---|
| Algorithm | F1-micro | F1-macro | F1-micro | F1-macro | F1-micro | F1-macro |
| GraphSAGE-mean | 0.866 | 0.864 | 0.583 | 0.581 | 0.893 | 0.889 |
| GraphSAGE-pool | 0.871 | 0.869 | 0.587 | 0.579 | 0.911 | 0.904 |
| GraphSAGE-LSTM | 0.873 | 0.871 | 0.522 | 0.519 | 0.915 | 0.898 |
| GSM-attention | **0.881** | **0.873** | **0.694** | **0.691** | **0.918** | **0.911** |

**Table 2.** The results of experiments for multilabel classification task

## 6 Conclusion

In this research we investigated the problem of learning graph embeddings with respect to its applications in dynamic networks and inductive formalization in order to generalize to unseen nodes. We presented an overview and the classification of the modern methods for graph embeddings and learning algorithms for both transductive and inductive approaches. Selecting GraphSAGE algorithm as one of the state-of-the-art approaches for inductive learning, we proposed an improved model based on GraphSAGE algorithm and set up the experiments on datasets (CORA, Reddit, HSEcite), including our own data. The results evaluated by F1-micro and -macro metrics show that our model outperforms simple Graph–SAGE models with mean, LSTM and pool aggregation functions, which were taken as baselines. The key advantages of GSM model: (i) GraphSAGE-based model with attention-based aggregation function has the better ability to generalize not only to unseen nodes, but also to unseen graphs; (ii) the results were significantly improved in comparison to GraphSAGE on the Reddit dataset, used in the original paper, CORA and on our own data HSEcite; (iii) we

showed the significance of deliberate choice of aggregation function, optimization method, more normalization layers and regularization (e.g. dropout) for a certain type of neural network architecture.

## Acknowledgments

Sections 2–5 were prepared under support by the Russian Science Foundation under grant 17-11-01294, performed at National Research University Higher School of Economics, Russia. Section 1 was prepared under support by RFBR grant 16-29-09583 "Development of methodology, methods and tools for identifying and countering the proliferation of malicious information campaigns in the Internet".

## References

1. A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 29–42.
2. R. Laptsuev, M. Ananyeva, D. Meinster, I. Makarov, I. Karpov, and L. Zhukov, "Information propagation strategies in online social networks," *Large Scale Networks - Computational Aspects and Applications - Computational Aspects and Applications*, pp. 1–8, 2018.
3. R. M. Khayrullin, I. Makarov, and L. E. Zhukov, "Predicting psychology attributes of a social network user," in *Proceedings of EEML Workshop*. Ceur WP, 2017, pp. 1–10.
4. F. Kyriakopoulos, S. Thurner, C. Puhr, and S. W. Schmitz, "Network and eigenvalue analysis of financial transaction networks," *The European Physical Journal B*, vol. 71, no. 4, p. 523, 2009.
5. I. Makarov, O. Gerasimova, P. Sulimov, and L. E. Zhukov, "Recommending co-authorship via network embeddings and feature engineering: The case of national research university higher school of economics," in *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries*, ser. JCDL '18. New York, NY, USA: ACM, 2018, pp. 365–366. [Online]. Available: http://doi.acm.org/10.1145/3197026.3203911
6. I. Makarov, O. Bulanov, O. Gerasimova, N. Meshcheryakova, I. Karpov, and L. E. Zhukov, "Scientific matchmaker: Collaborator recommender system," in *Analysis of Images, Social Networks and Texts*, W. M. van der Aalst, D. I. Ignatov, M. Khachay, S. O. Kuznetsov, V. Lempitsky, I. A. Lomazova, N. Loukachevitch, A. Napoli, A. Panchenko, P. M. Pardalos, A. V. Savchenko, and S. Wasserman, Eds. Cham: Springer International Publishing, 2018, pp. 404–410.
7. I. Makarov, O. Bulanov, and L. E. Zhukov, "Co-author recommender system," in *Models, Algorithms, and Technologies for Network Analysis*, V. A. Kalyagin, A. I. Nikolaev, P. M. Pardalos, and O. A. Prokopyev, Eds. Cham: Springer International Publishing, 2017, pp. 251–257.
8. A. Kurmukov, M. Ananyeva, Y. Dodonova, B. Gutman, J. Faskowitz, N. Jahanshad, P. Thompson, and L. Zhukov, "Classifying phenotypes based on the community structure of human brain networks," in *Graphs in Biomedical Image Analysis, Computational Anatomy and Imaging Genetics*. Springer, 2017, pp. 3–11.

9. S. Brohee and J. Van Helden, "Evaluation of clustering algorithms for protein-protein interaction networks," *BMC bioinformatics*, vol. 7, no. 1, p. 488, 2006.
10. P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.
11. P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *arXiv preprint arXiv:1711.08752*, 2017.
12. H. Cai, V. W. Zheng, and K. Chang, "A comprehensive survey of graph embedding: problems, techniques and applications," *IEEE Transactions on Knowledge and Data Engineering*, 2018.
13. J. Leskovec, "Deep learning for network biology. part 1: Network propagation and node embeddings," *Tutorial*, 2018.
14. W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.
15. A. Mees, P. Rapp, and L. Jennings, "Singular-value decomposition and embedding dimension," *Physical Review A*, vol. 36, no. 1, p. 340, 1987.
16. 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*. ACM, 2016, pp. 1105–1114.
17. X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding." in *AAAI*, 2017, pp. 203–209.
18. J. Wang, "Locally linear embedding," in *Geometric Structure of High-Dimensional Data and Dimensionality Reduction*. Springer, 2012, pp. 203–220.
19. 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*. ACM, 2015, pp. 891–900.
20. 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*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.
21. 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*. ACM, 2014, pp. 701–710.
22. C. Tu, W. Zhang, Z. Liu, M. Sun *et al.*, "Max-margin deepwalk: Discriminative learning of network representation." in *IJCAI*, 2016, pp. 3889–3895.
23. 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*. ACM, 2016, pp. 855–864.
24. B. Rozemberczki and R. Sarkar, "Fast sequence-based embedding with diffusion graphs," in *International Workshop on Complex Networks*. Springer, 2018, pp. 99–107.
25. J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," *Neural Networks: Tricks of the Trade*, pp. 639–655, 2012.
26. 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*. ACM, 2016, pp. 1225–1234.
27. X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

28. M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of machine learning research*, vol. 7, no. Nov, pp. 2399–2434, 2006.
29. J. Ma, P. Cui, and W. Zhu, "Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks," *AAAI*, 2018.
30. A. Bojchevski and S. Günnemann, "Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking," *arXiv preprint arXiv:1707.03815*, 2017.
31. P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, vol. 1, no. 2, 2017.