Modified SkipGram Negative Sampling Model for Faster Convergence of Graph Embedding

Kostas Loumponias[0000−0002−6268−3893] ,
Andreas Kosmatopoulos[0000−0001−5334−741X] ,
Theodora Tsikrika[0000−0003−4148−9028] ,
Stefanos Vrochidis[0000−0002−2505−9178] ,
Ioannis Kompatsiaris[0000−0001−6447−9020]

Information Technologies Institute,
Centre for Research and Technology Hellas-CERTH, GR-54124, Thessaloniki, Greece
{loumponias, akosmato, theodora.tsikrika, stefanos, ikom}@iti.gr

Abstract. Graph embedding techniques have been introduced in recent years with the aim of mapping graph data into low-dimensional vector spaces, so that conventional machine learning methods can be exploited. In particular, in the DeepWalk model, truncated random walks are employed in random walk-based approaches to capture structural links-connections between nodes. The SkipGram model is then applied to the truncated random walks to compute the embedded nodes. In this work, the proposed DeepWalk model provides a faster convergence speed than the standard one by introducing a new trainable parameter in the model. Furthermore, experimental results on real-world datasets show that the performance in downstream community detection and link prediction task is improved by using the proposed DeepWalk model.

Keywords: Graph Embedding · DeepWalk · Community Detection · Link Prediction.

1 Introduction

In the past few years, there has been a significant increase in the volume of data generated by services that utilise various type of networks. Graphs analysis is used for representing information in various networks (e.g., citation networks, sensor networks, social networks [8] etc.) as graphs, taking into account the interactions between the network entities. Consequently, inherent properties of the network (a.k.a graph) can be discovered, using graph analytical tasks, such as node classification [2], community detection [15], link prediction [18] and visualization [20].

Recently, graph embedding methods that represent graph nodes in a vector space have been developed. The main goal of graph embedding methods is to map graph nodes into a low-dimensional latent vector space, while maximally preserving the properties of the graph structure. Therefore, node similarity in the original complex irregular spaces can be quantified based on various similarity

measures in the latent vector space (or embedded space). In addition, more accurate graph analytics tasks can be leveraged from the learned embedded space, as opposed to directly performing such tasks in the high-dimensional complex graph domain. Graph embedding methods can be classified into three main categories [3, 8]: (i) factorization-based, (ii) random walk-based, and (iii) deep learning-based.

Factorization-based methods describe the connections between nodes as a matrix and factorize this matrix to obtain the embedded nodes. The most common matrices used to represent the connections between nodes are the node adjacency matrix, Laplacian matrix, and node transition probability matrix. Based on the characteristics of the representative matrix, different approaches to factorization might be used. In the case of the Laplacian matrix, eigenvalue decomposition can be used if the obtained matrix is positive semi-definite. Gradient descent algorithms can be used to speed up the factorization-based methods.

Random walk-based methods are used to obtain the topological relationships between nodes by performing truncated random walks. To that end, a graph is converted into a collection of node sequences (using truncated random walks), in which the frequency of node pairs measures the structural distance between them. Then, machine-learning (ML) based methods are used for obtaining the embedding. The most common method used to calculate the embedded nodes using truncated random walks is the *SkipGram model* [21].

Deep learning-based methods apply well-established deep learning (DP) models on a whole graph (or the corresponding proximity matrix) to obtain the embedded nodes. Autoencoders have been utilised for dimensionality reduction [23] due to their ability to model non-linear structure in the data. Furthermore, as an extension of the standard Convolutional Neural Network, the Graph Convolution Neural Network [29] has been proposed to deal with non-Euclidian structural data, such as the graphs.

In this work, we focus on random walk-based methods, proposing a novel SkpGram model that provides a faster convergence speed in calculating the embedded nodes. The proposed approach introduces a new function, called *sigmoid b* $\sigma_b(x)$, in order to tackle some of the limitations of the typical sigmoid function, such as the saturated values [11]. Subsequently, the forward and back-propagation stage of the proposed SkipGram model are calculated and the calculated embedded nodes are utilised in the community and link prediction tasks. More precisely, the k-means algorithm [10] is applied to embedded nodes to calculate the network communities, while the logistic regression model [12] is used to predict the existence of edges (links) between two nodes in the graph.

This work extends our previous work [18] as follows: First of all, proofs of the proposed SkipGram model (not included in [18]) are provided in detail, along with a more comprehensive description of the standard SkipGram model. Moreover, further extensive evaluation experiments are performed for demonstrating the effectiveness of the proposed method by considering additional real-world datasets and by examining in depth the effect of the different hyperparameters

of the proposed model on the performance for the community detection and node classification tasks.

The rest of the paper is organised as follows: In Section 2, random walk-based methods are described. In Section 3, the proposed method is provided. In Section 4, experimental results are presented using real-world networks to demonstrate the effectiveness of the proposed framework. Finally, in Section 5, conclusions and future work are discussed.

2 Related Work

The DeepWalk (DW) method [25] adopts the SkipGram model, which is a neural language model for producing graph embeddings. More specifically, the SkipGram model attempts to maximize the likelihood of words that appear within the same sentence. Thus, DW first performs truncated random walks (with fixed length t) for each node of the graph to obtain a set of node sequences. This process is repeated n times (number of walks) and it follows that a node and a node sequence can be interpreted as a word and a sentence, respectively. Then, the SkipGram model is applied on the node sequence to maximize the likelihood of observing a node's neighbourhood conditioned on its embedded nodes. Hence, nodes with similar neighbourhoods (second order proximity) share similar embeddings.

In the same way as in DW, the node2vec (n2v) method [9] preserves higher-order proximity between nodes using truncated random walks (with fixed length) and the SkipGram model. The main difference between DW and node2vec is that n2v employs biased-random walks that provide a trade-off between breadth-first (BFS) and depth-first (DFS) graph searches. The results have shown that in many network tasks, such as community detection and node classification tasks, n2v produces higher-quality and more informative nodes embedded than DW.

In the DW and n2v methods, the embedded nodes are randomly initialized. However, such initializations may end up trapped in local optima, since the objective function of DW and n2v is non-convex. In order to tackle this limitation, hierarchical representation learning for networks (HARP) [5] was proposed, where it provides a novel method to initialize the model weights. HARP aims to find a smaller graph which approximates the global structure of the original graph. This simplified graph is utilised to learn a set of initial representations, which serve as good initializations for learning representations in the original graph. HARP is a general meta-strategy to improve all graph embedding methods, such as DW and n2v.

Moreover, the WALKLETS method [26] generates multi-scale representations of graph nodes, by sub-sampling short random walks on the nodes. By skipping some nodes, WALKLETS alters the random walk method used in DW and n2v. To that end, a similar to factorizing GraRep approach [4] is performed for multiple skip lengths. The resulting node sequences are used for training the SkipGram model. Finally, other variations of the above methods are Deep Random Walk [17] and Tri-party Deep Network Representation [24].

3 Graph Embedding: Random Walk Based Technique

In this section, a brief description of the DW method is provided, considering the negative sampling approach [22] in SkipGram model. Next, the proposed SkipGram model is presented, providing detailed proofs. Following that, the proposed SkipGram model is used as an initial step for the community detection and link prediction downstream tasks.

3.1 Standard SkipGram Model: Negative Sampling Approach

The DW (and n2v) method consists of two steps, (i) random walk sampling and (ii) the SkipGram model (see Fig. 1). In this work, we focus on the SkipGram model, thus, no additional details about random walks are reported from here on. Let $G = (V, E)$ be a graph, where V and $E \subseteq (V \times V)$ stands for the node and edge set of graph G, respectively. The standard SkipGram model corresponds to a fully connected neural network with one hidden layer (without any activation function) and multiple outputs (see Fig. 1). The main goal of SkipGram model is to predict the surrounding nodes (context nodes) of a given target node. To that end, the embedded vectors of the nodes are calculated.

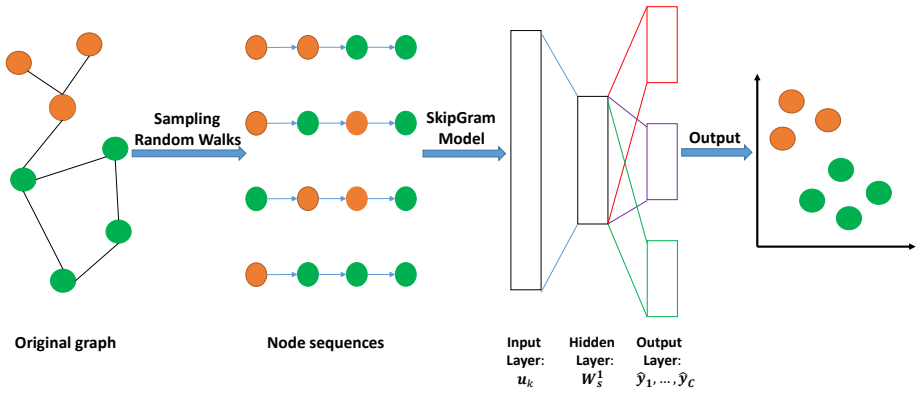

Fig. 1. The DeepWalk model.

Next, the notation of SkipGram model parameters are provided and described in detail. The input vector $\mathbf{u}_k$ is the one-hot vector of target node $u_k \in V$, $\mathbf{W}^1 \in M_{|V|\times d}$ [1] is the embedded matrix, where $|V|$ denotes the total number of the nodes and d the embedding size. Each row ($i = 1, 2, ..., |V|$) of $\mathbf{W}^1$ represents the embedded vector of node $u_i \in V$. $\mathbf{W}^2 \in M_{d\times|V|}$ is the output embedded matrix, while $\{\hat{\mathbf{y}}_{k-w}, ..., \hat{\mathbf{y}}_{k-1}, \hat{\mathbf{y}}_{k+1}, ..., \hat{\mathbf{y}}_{k+w}\}$ are the predicted context nodes (one-hot

[1] $M_{n\times m}$ denotes the set of matrices $n \times m$.

vectors) when the input-target node is u_k, (or equivalently $\mathbf{u}_k$), where w denotes the window size. The sets of predictions $\{\hat{\mathbf{y}}_{k+i}\}_{i=-w:w,i\neq 0}$ for convenience will be denoted as $\{\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_C\}$.

From now on, it assumed that $\mathbf{W}_s^1 = \mathbf{W}^{1'} \cdot \mathbf{u}_k$, where $\mathbf{W}_s^1$ corresponds to the s-row of $\mathbf{W}^1$, since $\mathbf{u}_k$ is one-hot-vector. Furthermore, $\mathbf{W}_s^1$ stands for the embedded vector of target node u_k. In the standard SkipGram model, the cost function for the embedded target $\mathbf{W}_s^1$ is calculated as

$$J(\theta) = -\sum_{c=1}^{C} \log \frac{\exp\left(\mathbf{W}_c^{2'} \cdot \mathbf{W}_s^1\right)}{\sum_{i=1}^{|V|} \exp\left(\mathbf{W}_i^{2'} \cdot \mathbf{W}_s^1\right)}, \tag{1}$$

where $\theta = [\mathbf{W}^1, \mathbf{W}^2]$ and $\mathbf{W}_c^2$ represents the c-th column of $\mathbf{W}^2$. In addition, the function

$$P(u_c|u_k;\theta) = \frac{\exp\left(\mathbf{W}_c^{2'} \cdot \mathbf{W}_s^1\right)}{\sum_{i=1}^{|V|} \exp\left(\mathbf{W}_i^{2'} \cdot \mathbf{W}_s^1\right)} \tag{2}$$

represents the conditional probability of observing a context node u_c (embedded output $\mathbf{W}_c^2$) given the target node u_k (embedded target $\mathbf{W}_s^1$) . It is clear that the cost function (1) is computationally inefficient, since the denominator in (1) requires $|V|$ iterations (total number of nodes). Due to this computational burden, cost function (1) is not used in most implementations of SkipGram model.

In order to overcome this limitation, the negative sampling process is used, which reduces the complexity of the SkipGram model. In a nut shell, the negative sampling process draws K number of negative samples (pair of nodes with low proximity) using the noise distribution $P_n(w)$ [22], for each positive pair (pair of nodes with high proximity). Thus, the logarithm of condition probability function (2) is approximated by

$$\log P(u_c|u_k;\theta) = \log \sigma\left(\mathbf{W}_c^{2'} \cdot \mathbf{W}_s^1\right) + \sum_{i=1}^{K} \log \sigma\left(-\mathbf{W}_{neg(i)}^{2'} \cdot \mathbf{W}_s^1\right), \tag{3}$$

where $\sigma(x)$ is the sigmoid function, while the row $\mathbf{W}_{neg(i)}^2$ is randomly selected from matrix $\mathbf{W}^2$, using the noise distribution $P_n(w)$. The first term of (3) indicates the logarithmic probability of u_c (embedded vector $\mathbf{W}_c^2$) to appear within the context window of the target node u_k (embedded vector $\mathbf{W}_s^1$), while, the second term indicates the logarithmic probability of node $u_{neg(i)}$ (embedded vector $\mathbf{W}_{neg(i)}^2$) not appearing in the context window of u_k.

In the negative sampling process, $K+1$ columns of the output embedded matrix $\mathbf{W}^2$ are updated, while in the embedded matrix $\mathbf{W}^1$ only the row $\mathbf{W}_s^1$ is updated, since the input $\mathbf{u}_k$ is one-hot vector. The number of negative samples K usually is set equal to 5. Finally, the update equations of SkipGram model parameters are calculated as follows:

$$\mathbf{c}_j = \mathbf{c}_i - \eta \cdot (\sigma(x_i) - t_i) \cdot \mathbf{W}_s^1, \tag{4}$$

$$\mathbf{W}_s^1 = \mathbf{W}_s^1 - \eta \cdot \sum_{i=1}^{K+1} (\sigma(x_i) - t_i) \cdot \mathbf{c}_i, \tag{5}$$

where $\mathbf{c}_i = \begin{cases} \mathbf{W}_c^2, & i = 1 \\ \mathbf{W}_{neg(j-1)}^2, & i = 2, ..., K+1 \end{cases}$, $t_i = \begin{cases} 1, & i = 1 \\ 0, & i = 2, ..., K+1 \end{cases}$,
$x_i = \mathbf{c}_i^{'} \cdot \mathbf{W}_s^1$ and η is the learning rate.

The term $(\sigma(x_i) - t_i)$ in the update equations (4), (5) is derived from the derivatives of $-\log \sigma(x_i)$ with respect to $\mathbf{c}_i$ and $\mathbf{W}_s^1$ for $i = 1, ..., K+1$, respectively. In the case of positive sample (i.e., $i = 1$) and low values of x_i (i.e., $x_i \to -\infty$), the term $(\sigma(x_i) - t_i)$ is maximized. Therefore, the SkipGram model updates-corrects the weights $\theta = [\mathbf{W}^1, \mathbf{W}^2]$ for low values of x_i, otherwise, when the values of x_i are high, the updates (values of $(\sigma(x_i) - t_i)$) are negligible. In the case of negative samples ($i \neq 1$) and low values of x_i, the updates are negligible, otherwise for high values of x_i the updates are maximized. Thus, the inner product $x_i = \mathbf{c}_i^{'} \cdot \mathbf{W}_s^1$ defines a proximity between the nodes u_k and u_c, and the aim of the SkipGram model is to maximize and minimize it for positive and negative samples, respectively.

3.2 Proposed SkipGram Model

In the standard negative sampling approach described earlier, the conditional probability function (2) is approximated using the sigmoid function (3). As it is known, the values of a probability function must lie in the interval $[0, 1]$; this makes the sigmoid function an appropriate choice. However, one of the main drawbacks of the sigmoid function is the saturated values of its derivatives (gradients). More specifically, the derivatives of $\log \sigma(x)$ converge to 0 for $x \to +\infty$ and to 1 for $x \to -\infty$. Therefore, for any low value of x, the derivative is essentially equivalent to 1. Thus, the range of the updates is constrained, and its maximum value is 1. This can lead to a significant computing cost until the weights θ converge, according to the gradient descent method.

In order to tackle the above restriction, the sigmoid b function is proposed

$$\sigma_b(x) = \frac{1}{1 + \exp(-b \cdot x)}, \tag{6}$$

where $b > 0$. It is clear that the values of the proposed function lie in the interval $[0, 1]$, since it approximates the conditional probability (3). Next, in Lemma 1 the derivatives of the proposed $\sigma_b(x)$ with respect to (w.r.t.) x and b are provided.

Lemma 1. *The derivatives of* $\log \sigma_b(x)$ *w.r.t.* x *and* b *are equal to:*

$$\frac{\partial \log \sigma_b(x)}{\partial x} = b \cdot (1 - \sigma_b(x)), \forall x \in \mathbb{R}, \tag{7}$$

$$\frac{\partial \log \sigma_b(x)}{\partial b} = x \cdot (1 - \sigma_b(x)). \tag{8}$$

Proof.

$$\begin{aligned}\frac{\partial \log \sigma_b(x)}{\partial x} &= \frac{1}{\sigma_b(x)} \cdot \frac{\partial \sigma_b(x)}{\partial x} = \frac{1}{\sigma_b(x)} \cdot \frac{\partial}{\partial x}\left(\frac{1}{1+exp(-b\cdot x)}\right) \\ &= \frac{1}{\sigma_b(x)} \cdot \frac{b\cdot exp(-b\cdot x)}{(1+exp(-b\cdot x))^2} \\ &= b\cdot \frac{1}{\sigma_b(x)} \cdot \frac{1}{1+exp(-b\cdot x)} \cdot \frac{exp(-b\cdot x)}{1+exp(-b\cdot x)} \\ &= b\cdot(1-\sigma_b(x)).\end{aligned}$$

since $\sigma_b(x) \neq 0, \forall x \in \mathbb{R}$. Next, in the same way is proved that

$$\frac{\partial \log \sigma_b(x)}{\partial b} = x\cdot(1-\sigma_b(x)).$$

□

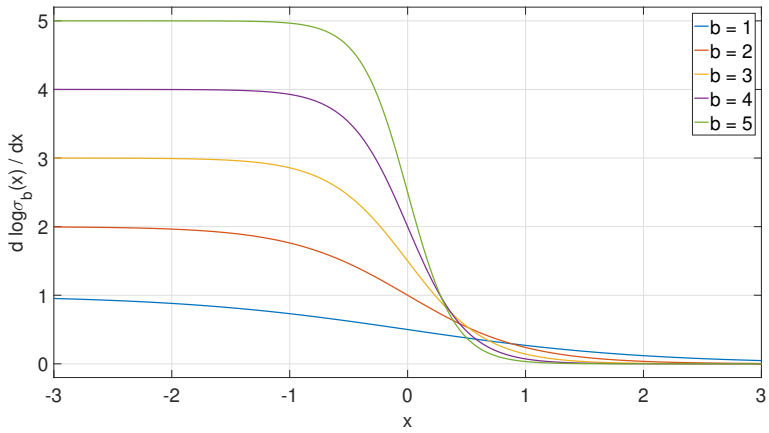

Fig. 2. The derivatives of $\log \sigma_b(x)$.

Figure 3.2 illustrates the derivative of $\log \sigma_b(x)$ (eq. 7) w.r.t x for different values of parameter b. It is clear that, in the case of $b = 1$, the derivative of $\log \sigma_b(x)$ is similar to the derivative of $\log \sigma(x)$. Furthermore, the range of updates (derivatives) for $b > 1$ are higher than the corresponding for $b = 1$. Next, in Proposition 1 the update equations of $\mathbf{W}_c^2, \mathbf{W}_{neg(i)}^2, \mathbf{W}_s^1$ and b using $\sigma_b(x)$ and the gradient descent technique are provided.

Proposition 1. *The update equations of the negative logarithm conditional probability function*

$$-\log P(u_c|u_k;\theta) = -\log \sigma_b\left(\boldsymbol{W}_c^{2'} \cdot \boldsymbol{W}_s^1\right) - \sum_{i=1}^{K} \log \sigma_b\left(-\boldsymbol{W}_{neg(i)}^{2'} \cdot \boldsymbol{W}_s^1\right), \quad (9)$$

w.r.t. $\boldsymbol{W}_c^2$, $\boldsymbol{W}_{neg(i)}^2$, $\boldsymbol{W}_s^1$ *and* b *using the gradient descent technique are*

$$\boldsymbol{c}_i = \boldsymbol{c}_i - \eta \cdot b \cdot (\sigma_b(x_i) - t_i) \cdot \boldsymbol{W}_s^1, \quad (10)$$

$$\boldsymbol{W}_s^1 = \boldsymbol{W}_s^1 - \eta \cdot b \cdot \sum_{i=1}^{K+1} (\sigma_b(x_i) - t_i) \cdot \boldsymbol{c}_i, \quad (11)$$

and

$$b = b - \eta_b \cdot \sum_{i=1}^{K+1} x_i \cdot (\sigma_b(x_i) - t_i), \quad (12)$$

where η_b *is the learning rate of parameter* b*.*

Proof. Initially, the derivatives of (9) w.r.t. $\mathbf{W}_c^2, \mathbf{W}_{neg(i)}^2, \mathbf{W}_s^1$ and b are calculated.

The derivative of (9) w.r.t. $\mathbf{W}_c^2$ is equal to

$$\begin{aligned} -\frac{\partial \log P(u_c|u_k;\theta)}{\partial \mathbf{W}_c^2} &= -\frac{\partial \log \sigma_b(x_c)}{\partial \mathbf{W}_c^2} = -\frac{\partial \log \sigma_b(x_c)}{\partial x_c} \cdot \frac{\partial x_c}{\partial \mathbf{W}_c^2} \\ &\overset{(7)}{=} -b \cdot (1 - \sigma_b(x_c)) \cdot \mathbf{W}_s^1 = b \cdot (\sigma_b(x_c) - 1) \cdot \mathbf{W}_s^1 \end{aligned} \quad (13)$$

where $x_c = \mathbf{W}_c^{2'} \cdot \mathbf{W}_s^1$.

The derivative of (9) w.r.t $\mathbf{W}_{neg(j)}^2$ (where $j = 1, ..., K$) is equal to

$$\begin{aligned} -\frac{\partial \log P(u_c|u_k;\theta)}{\partial \mathbf{W}_{neg(j)}^2} &= -\frac{\partial}{\partial \mathbf{W}_{neg(j)}^2} \sum_{i=1}^{K} \log \sigma_b(x_{neg(i)}^-) \\ &= -\frac{\partial \log \sigma_b(x_{neg(j)}^-)}{\partial x_{neg(j)}^-} \cdot \frac{\partial x_{neg(j)}^-}{\partial \mathbf{W}_{neg(j)}^2} \\ &\overset{(7)}{=} b \cdot (1 - \sigma_b(x_{neg(j)}^-)) \cdot \mathbf{W}_s^1 \\ &= b \cdot (\sigma_b(\mathbf{W}_{neg(j)}^{2'} \cdot \mathbf{W}_s^1)) \cdot \mathbf{W}_s^1 \end{aligned} \quad (14)$$

where $x_{neg(j)}^- = -\mathbf{W}_{neg(j)}^{2'} \cdot \mathbf{W}_s^1$.

The derivative of (9) w.r.t $\mathbf{W}_s^1$ is equal to

$$
\begin{aligned}
-\frac{\partial \log P(u_c|u_k;\theta)}{\partial \mathbf{W}_s^1} &= -\frac{\partial \log \sigma_b(x_c)}{\partial \mathbf{W}_s^1} - \frac{\partial}{\partial \mathbf{W}_s^1}\sum_{i=1}^{K}\log \sigma_b(x^-_{neg(i)}) \\
&= \frac{\partial \log \sigma_b(x_c)}{\partial x_c}\cdot\frac{\partial x_c}{\partial \mathbf{W}_s^1} - \sum_{i=1}^{K}\frac{\partial \log \sigma_b(x^-_{neg(i)})}{\partial x^-_{neg(i)}}\cdot\frac{\partial x^-_{neg(i)}}{\partial \mathbf{W}_s^1} \\
&\overset{(7)}{=} -b\cdot(1-\sigma_b(x_c))\cdot\mathbf{W}_c^2 + \sum_{i=1}^{K} b\cdot(1-\sigma_b(x^-_{neg(i)}))\cdot\mathbf{W}^2_{neg(i)} \\
&= \sum_{i=1}^{K+1} b\cdot(\sigma_b(x_i)-t_i)\cdot\mathbf{c}_i, \qquad (15)
\end{aligned}
$$

where the definitions of x_i, t_i and $\mathbf{c}_i$ are provided and described in subsection 3.1.

Next, the objective function (9) can be written as

$$
-\log P(u_c|u_k;\theta) = -\log\sigma_b\left(x_1\right) - \sum_{i=1}^{K}\log\sigma_b\left(-x_{i+1}\right),
$$

therefore, the derivative of the above objective function w.r.t b is calculated as:

$$
\begin{aligned}
-\frac{\partial \log P(u_c|u_k;\theta)}{\partial b} &= -\frac{\partial \log \sigma_b(x_1)}{\partial b} - \sum_{i=1}^{K}\frac{\partial \log \sigma_b(-x_{i+1})}{\partial b} \\
&\overset{(8)}{=} -x_1\cdot(1-\sigma_b(x_1)) + \sum_{i=1}^{K} x_{i+1}\left(1-\sigma_b(-x_{x+i})\right) \\
&= x_1\cdot(\sigma_b(x_1)-1) + \sum_{i=1}^{K} x_{i+1}\left(\sigma_b(x_{i+i})-0\right) \\
&= \sum_{i=1}^{K+1} x_i\left(\sigma_b(x_i)-t_i\right). \qquad (16)
\end{aligned}
$$

Finally, using the gradient descent technique [13] and the derivatives (13)-(16), the update equations of the proposed SkipGram model are calculated. □

The results in Proposition 1 show that the proposed model (SkipGram$_b$) and $\sigma_b(x)$ can be used to replace the standard one (SkipGram and $\sigma(x)$) in any machine learning model. However, the updates equations (10)-(12) must be adjusted to the particular machine learning model.

It is important to note that numerous activation functions have been presented in literature in an effort to improve performance in various scientific domains. The SM-Taylor softmax function was used in [1] for image classification tasks and the results showed that it performed better than the standard softmax

function. In addition, some of the best known and well established alternative activation functions are the Exponential Linear Unit (ELU) [6] and Scaled exponential Linear Unit (SELU) [14]. However, the aforementioned functions cannot be applied to embedding methods, since the function must be bounded in the interval $[0, 1]$ as previously explained.

3.3 Community Detection and Link Prediction Tasks

The proposed DeepWalk$_b$ (DW$_b$) method has similar framework to standard DW, with the main difference being the use of the SkipGram$_b$ model instead of the standard one. In Algorithm 1, the proposed framework for community detection is presented. More precisely, lines $1-7$ represent the DW$_b$ process. Then, the $k-$means algorithm is applied on the embedded nodes to detect the communities of the graph (Com). Moreover, the proposed SkipGram$_b$ model can be also applied to the n2v process, since DW and n2v differ only in how truncated random walks are performed, as shown in line 4 of Algorithm 1.

Algorithm 1 DeepWalk$_b$ for community detection

Require: Graph G, number of communities k, window size w, embedding size d, walk length t, number of walks n

1: **for** `i=1:n` **do**
2: $O = Shuffle(V)$
3: **for** $u_i \in O$ **do**
4: $RW_{u_i} = \text{RandomWalk}(G, u_i, t)$
5: $\theta = \text{SkipGram}_b(RW_{u_i}, w)$
6: **end for**
7: **end for**
8: $Com = k\text{-means}(\theta, k)$
9: **return** Com

The process of link prediction and evaluation are provided in Algorithm 2. Three sub-graphs, denoted G_{tr}, G_{mod} and G_{ts} are explicitly derived from the original graph G. To that end, the StellarGraph tool [9] is used to split the graph $G = (V, E)$ to $G_{tr} = (V, E_{tr})$, $G_{mod} = (V, E_{mod})$ and $G_{ts} = (V, E_{ts})$. The embedded nodes θ_{tr} are first calculated using the *train* graph G_{tr}. Then, using four distinct operators ($oprtr$): Hadamard product, $L1$, $L2$ norm [19] and the simple average, the similarities between embedded nodes are calculated. Afterwards, the classifiers, one for each operator, are calculated using the logistic regression model (whether the nodes are connected or not). Then, the classifiers are evaluated considering the *model selection* graph G_{mod} and the embedded nodes θ_{tr}. The operator with the highest accuracy score is used to evaluate the classifier for the *test* graph G_{ts}.

Algorithm 2 DeepWalk$_b$ for link prediction

Require: Graph G, window size w, embedding size d, walk length t, number of walks n

G_{tr}, G_{mod}, G_{ts} = split(G)
θ_{tr} = DW$_b$(G_{tr}, w, d, t, n)
for i=1:ν_{oprtr} **do**
    clsfr(i) = Logistic Regression(θ_{tr}, operator(i))
    AccScore(i) = evaluate(clsfr(i), G_{mod}, θ_{tr}, oprtr(i))
end for
i_{max} = argmax(AccScore)
$clsfr_{max}$, $oprtr_{max}$ = clsfr(i_{max}), oprtr(i_{max})
θ_{ts} = DW$_b$(G_{tsr}, w, d, t, n)
Test Score = evaluate($clsfr_{max}$, G_{ts}, θ_{ts}, $oprtr_{max}$)

4 Experimental Evaluation

In this section, the proposed method, DW$_b$ is evaluated against the baseline of DW on community detection and link prediction tasks. Implementation details are discussed before presenting results on real-world datasets.

4.1 Data Description and Implementation Details

The proposed method DW$_b$ and DW are evaluated on five publicly available real-world datasets with ground-truth communities: Cora, CiteSeer, PubMed [27], ego-Facebook and Amazon [16]. Cora (2708 nodes and 5429 edges), CiteSeer (3312 nodes and 4732 edges) and PubMed (19717 nodes and 44338 edges) contain publications, ego-Facebook (2871 nodes and 62334 edges) contains social circles formed from users of Facebook, while Amazon (15,716 nodes and 48739 edges) contains products found in the Amazon website that are frequently bought together. Table 4.1 showcases the real-world datasets used in the evaluation, as well as the number of their communities.

Table 1. Real-world datasets.

Dataset	Nodes	Edges	Communities
Cora	2,708	5,429	7
CiteSeer	3,312	4,732	6
PubMed	19,717	44,338	3
ego-Facebook	2,871	63,334	147
Amazon	15,716	48,739	1,229

Next, the parameters used in methods DW$_b$ and DW are $w = 10$ (window size), $d = 128$ (embedding size), $t = 80$ (walk-length), while the number of epochs and the batch-size are equal to 15 and 1000, respectively. The learning rate η is set equal to 0.02 for both methods. In [18] it was shown that a larger

η value increases the convergence speed of DW with a trade-off in community detection performance. However, DW_b outperforms DW in convergence speed even for larger η values [18]. Thus, the scope of this work is to evaluate DW_b on the learning rate of parameter b. To that end, the experimental sets in DW_b are conducted considering different values of learning rate, $\eta_b \in \{0.01, 0.05, 0.2\}$. Finally, both methods use the stochastic gradient descent technique with momentum 0.9, for the back propagation process.

In each experiment, the Adjusted Rand Index (ARI), the Normalized Mutual Information (NMI) and the graph's modularity (Mod) [28] are calculated for the community detection task, while the Area Under the Curve (AUC) score [7] is calculated for the link prediction task. In addition, the edge sets of the subgraphs generated by the StellarGraph tool are defined in all experimental sets as follows: E_{ts} includes 90% of the total edges (E), while the E_{tr} and E_{mod} include 75% and 25% of E_{ts}, respectively.

4.2 Evaluation of DeepWalk$_b$ Model

Table 2. The best performances of DW and DW_b regarding the metrics ARI, NMI, Mod and AUC. The columns *cd epochs* and *lp epochs* contain the required number of epochs in order for the community detection metrics (i.e. ARI, NMI, Mod) and link prediction metric (AUC), respectively, to converge.

Dataset	Method	ARI	NMI	Mod	AUC	cd epochs	lp epochs
Cora	DW_b $\eta_b = 0.01$	0.389	0.457	0.745	0.893	2	6
	DW_b $\eta_b = 0.05$	0.392	0.463	0.743	0.893	1	2
	DW_b $\eta_b = 0.2$	0.389	0.457	0.741	0.903	2	2
	DW	0.390	0.463	0.747	0.879	7	14
CiteSeer	DW_b $\eta_b = 0.01$	0.127	0.457	0.725	0.913	2	4
	DW_b $\eta_b = 0.05$	0.151	0.462	0.727	0.905	2	3
	DW_b $\eta_b = 0.2$	0.124	0.462	0.737	0.911	1	4
	DW	0.127	0.463	0.701	0.870	8	12
PubMed	DW_b $\eta_b = 0.01$	0.318	0.299	0.602	0.761	1	1
	DW_b $\eta_b = 0.05$	0.319	0.301	0.601	0.771	1	1
	DW_b $\eta_b = 0.2$	0.318	0.299	0.603	0.877	1	1
	DW	0.302	0.296	0.601	0.768	6	6
ego-Facebook	DW_b $\eta_b = 0.01$	0.361	0.647	0.478	0.943	4	6
	DW_b $\eta_b = 0.05$	0.364	0.648	0.461	0.921	3	6
	DW_b $\eta_b = 0.2$	0.361	0.646	0.453	0.923	2	7
	DW	0.318	0.621	0.369	0.892	13	8
Amazon	DW_b $\eta_b = 0.01$	0.569	0.904	0.941	0.995	2	2
	DW_b $\eta_b = 0.05$	0.572	0.904	0.946	0.996	2	1
	DW_b $\eta_b = 0.2$	0.570	0.904	0.943	0.996	1	1
	DW	0.576	0.903	0.820	0.970	11	15

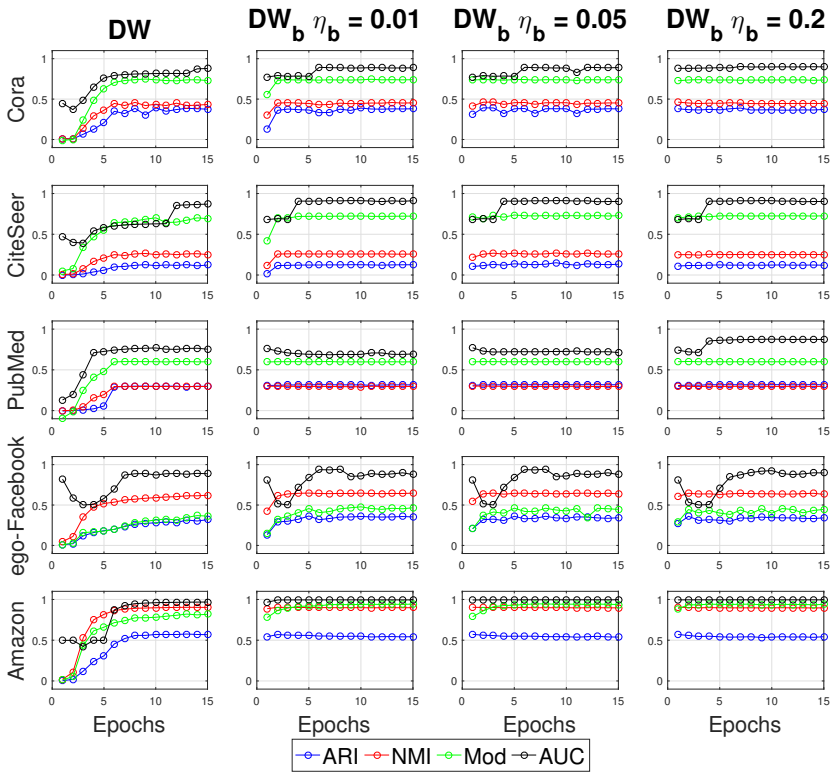

Fig. 3. The performances of DW and DW_b in community detection and link prediction task, considering the Cora, PubMed, CiteSeer, ego-Facebook and Amazon dataset, for different values of learning rate η_b.

Table 2 details the performances of DW_b (for various values of η_b) and standard DW in community detection and link prediction tasks, where *cd epochs* and *lp epochs* denote the required number of epochs in order the community detection metrics (i.e. ARI, NMI, Mod) and link prediction metric (AUC) to converge. Moreover, Figure 3 illustrates the performance of DW and DW_b considering the metrics ARI, NMI, Mod and AUC (y axis) for different values of η_b in Cora, CiteSeer, PubMed, eqo-Facebook and Amazon dataset, where the x axis (in the sub-figures) stands for the number of epochs.

As it is expected, DW_b converges faster than the standard DW in all datasets for both tasks due to the trainable parameter b. It is clear that the learning rate η_b has a low impact on convergence speed for DW_b. Only in the Cora and ego-Facebook datasets, (significantly) fewer epochs are required using higher values of η_b in link prediction and community detection task, respectively. Furthermore, the parameter b gets larger values during the training stage, when large values of η_b are used, while when community detection and link predictions metrics

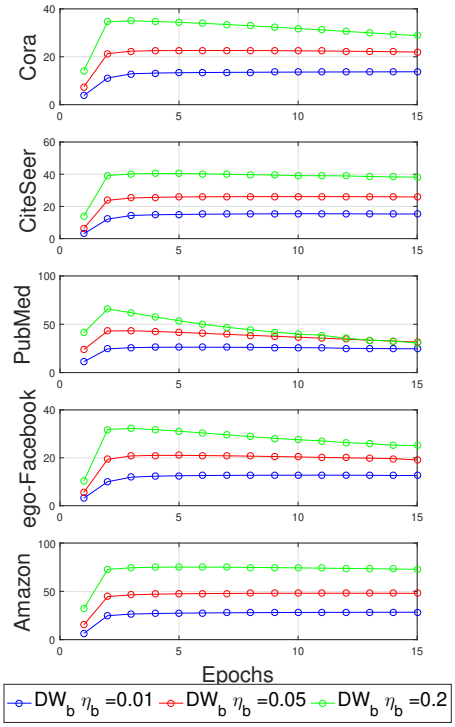

Fig. 4. The values of trainable parameter b per epoch, considering the Cora, PubMed, CiteSeer, ego-Facebook and Amazon datasets, for different values of learning rate η_b.

converge, the values of b either remain unchanged or decrease (see Figure 4). This property of parameter b is expected, since when the DW_b model converges, large updates to the model parameters are no longer required.

In addition, the proposed DW_b performs the community detection and link prediction tasks as well as (or better than) standard DW in fewer epochs. The best performances (for both methods) for all metrics within 15 epochs are detailed in Table 2. As it can be seen, DW_b provides the same performances (differences less than 0.02) regardless the learning rate η_b for all experimental sets with the exception of two cases. In the CiteSeer dataset, DW_b with $\eta_b = 0.05$ has ARI score equal to 0.151, while the other methods have ARI score close to 0.12. Moreover, in the ego-Facebook dataset, DW_b with $\eta_b = 0.01$ has AUC score equal to 0.943, while the rest DW_b ($\eta_b = 0.05$ and $\eta_b = 0.2$) and DW have AUC score close to 0.92 and 0.89, respectively. Next, the standard DW model provides a poor performance in link prediction task compared to DW_b in all dataset (differences greater than 0.02). Finally, in ego-Facebook dataset, DW_b

(regardless of η_b) outperforms DW in all metrics, while providing better Mod. scores on the CiteSeer and Amazon datasets.

5 Conclusions

The scope of this work was to propose a method for accelerating the convergence of the standard DW model, while preserving the accuracy in community detection and link predictions tasks. To this end, the DW_b model with the additional trainable parameter b was introduced. The new update equations of the proposed DW_b were proved in detail. Then, the calculated embedded nodes were used to detect communities and predict links between the nodes, using the k-means algorithm and the logistic regression model, respectively. According to the experimental results using real-world datasets, DW_b converged faster than DW in all experimental settings. Additionally, DW_b provided better AUC score than DW on all datasets, as well as better performance on the other metrics (i.e., ARI, NMI, Mod) in most cases. Finally, the experimental results showed that different values of learning rate η_b have a low impact on the convergence speed of DW_b, due to the ability of the proposed method to increase or decrease the values of b in a proper way.

References

1. Banerjee., K., C.., V., Gupta., R., Vyas., K., H.., A., Mishra., B.: Exploring alternatives to softmax function. In: Proceedings of the 2nd International Conference on Deep Learning Theory and Applications - DeLTA,. pp. 81–86. INSTICC, SciTePress (2021). https://doi.org/10.5220/0010502000810086
2. Bhagat, S., Cormode, G., Muthukrishnan, S.: Node classification in social networks. In: Social network data analytics, pp. 115–148. Springer (2011)
3. Cai, H., Zheng, V.W., Chang, K.C.C.: A comprehensive survey of graph embedding: Problems, techniques, and applications. IEEE Transactions on Knowledge and Data Engineering **30**(9), 1616–1637 (2018)
4. Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: Proceedings of the 24th ACM international on conference on information and knowledge management. pp. 891–900 (2015)
5. Chen, H., Perozzi, B., Hu, Y., Skiena, S.: Harp: Hierarchical representation learning for networks. In: Proceedings of the AAAI conference on artificial intelligence. vol. 32 (2018)
6. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 (2015)
7. Fawcett, T.: An introduction to roc analysis. Pattern recognition letters **27**(8), 861–874 (2006)
8. Goyal, P., Ferrara, E.: Graph embedding techniques, applications, and performance: A survey. Knowledge-Based Systems **151**, 78–94 (2018)
9. 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. pp. 855–864 (2016)

10. Hartigan, J.A., Wong, M.A.: Algorithm as 136: A k-means clustering algorithm. Journal of the royal statistical society. series c (applied statistics) **28**(1), 100–108 (1979)
11. Hochreiter, S.: The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6**(02), 107–116 (1998)
12. Hosmer Jr, D.W., Lemeshow, S., Sturdivant, R.X.: Applied logistic regression, vol. 398. John Wiley & Sons (2013)
13. Ketkar, N.: Stochastic gradient descent. In: Deep learning with Python, pp. 113–132. Springer (2017)
14. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. Advances in neural information processing systems **30** (2017)
15. Kosmatopoulos, A., Loumponias, K., Chatzakou, D., Tsikrika, T., Vrochidis, S., Kompatsiaris, I.: Random-walk graph embeddings and the influence of edge weighting strategies in community detection tasks. In: Proceedings of the 2021 Workshop on Open Challenges in Online Social Networks. pp. 9–13 (2021)
16. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014)
17. Li, J., Zhu, J., Zhang, B.: Discriminative deep random walk for network classification. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1004–1013 (2016)
18. Loumponias, K., Kosmatopoulos, A., Tsikrika, T., Vrochidis, S., Kompatsiaris, I.: A faster converging negative sampling for the graph embedding process in community detection and link prediction tasks. In: Proceedings of the 3rd International Conference on Deep Learning Theory and Applications - Volume 1: DeLTA,. pp. 86–93. INSTICC, SciTePress (2022). https://doi.org/10.5220/0011142000003277
19. Luo, X., Chang, X., Ban, X.: Regression and classification using extreme learning machine based on l1-norm and l2-norm. Neurocomputing **174**, 179–186 (2016)
20. Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(11) (2008)
21. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
22. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems **26**, 3111–3119 (2013)
23. Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., Zhang, C.: Adversarially regularized graph autoencoder for graph embedding. arXiv preprint arXiv:1802.04407 (2018)
24. Pan, S., Wu, J., Zhu, X., Zhang, C., Wang, Y.: Tri-party deep network representation. Network **11**(9), 12 (2016)
25. 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 (2014)
26. Perozzi, B., Kulkarni, V., Chen, H., Skiena, S.: Don't walk, skip! online learning of multi-scale network embeddings. In: Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017. pp. 258–265 (2017)
27. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI magazine **29**(3), 93–106 (2008)
28. Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. The Journal of Machine Learning Research **11**, 2837–2854 (2010)

29. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: International conference on machine learning. pp. 6861–6871. PMLR (2019)