# Understanding Negative Sampling in Graph Representation Learning

Zhen Yang*†, Ming Ding*†, Chang Zhou‡, Hongxia Yang‡, Jingren Zhou‡, Jie Tang†§

† Department of Computer Science and Technology, Tsinghua University
‡ DAMO Academy, Alibaba Group
zheny2751@gmail.com,dm18@mails.tsinghua.edu.cn
{ericzhou.zc,yang.yhx,jingren.zhou}@alibaba-inc.com
jietang@tsinghua.edu.cn

## ABSTRACT

Graph representation learning has been extensively studied in recent years, in which sampling is a critical point. Prior arts usually focus on sampling positive node pairs, while the strategy for negative sampling is left insufficiently explored. To bridge the gap, we systematically analyze the role of negative sampling from the perspectives of both objective and risk, theoretically demonstrating that negative sampling is as important as positive sampling in determining the optimization objective and the resulted variance. To the best of our knowledge, we are the first to derive the theory and quantify that a nice negative sampling distribution is $p_n(u|v) \propto p_d(u|v)^\alpha, 0 < \alpha < 1$. With the guidance of the theory, we propose MCNS, approximating the positive distribution with self-contrast approximation and accelerating negative sampling by Metropolis-Hastings. We evaluate our method on 5 datasets that cover extensive downstream graph learning tasks, including link prediction, node classification and recommendation, on a total of 19 experimental settings. These relatively comprehensive experimental results demonstrate its robustness and superiorities.

## CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**; • **Computing methodologies → Learning latent representations**.

## KEYWORDS

Negative Sampling; Graph Representation Learning; Network Embedding

**ACM Reference Format:**
Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, Jie Tang. 2020. Understanding Negative Sampling in Graph Representation Learning. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403218

*Equal contribution. Ming Ding proposed the theories and method. Zhen Yang conducted the experiments. Codes are available at https://github.com/zyang-16/MCNS.
§Corresponding Authors.


*KDD '20, August 23–27, 2020, Virtual Event, CA, USA*

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403218

## 1 INTRODUCTION

Recent years have seen the *graph representation learning* gradually stepping into the spotlight of data mining research. The mainstream graph representation learning algorithms include traditional Network Embedding methods (e.g. DeepWalk [27], LINE [32]) and Graph Neural Networks (e.g. GCN [17], GraphSAGE [13]), although the latter sometimes are trained end-to-end in classification tasks. These methods transform nodes in the graph into low-dimensional vectors to facilitate a myriad of downstream tasks, such as node classification [27], link prediction [45] and recommendation [40].

Most graph representation learning can be unified within a *Sampled Noise Contrastive Estimation* (SampledNCE) framework (§ 2), comprising of a trainable encoder to generate node embeddings, a positive sampler and a negative sampler to sample positive and negative nodes respectively for any given node. The encoder is trained to distinguish positive and negative pairs based on the inner products of their embeddings (See Figure 1 for illustration).

Massive network embedding works have investigated good criteria to sample positive node pairs, such as by random walk [27], the second-order proximity [32], community structure [37], etc. Meanwhile, the line of GNN papers focuses on advanced encoders, evolving from basic graph convolution layers [17] to attention-based [34], GAN-based [8], sampled aggregator [13] and WL kernel-based [39] layers. However, the strategy for negative sampling is relatively unexplored.

Negative sampling [24] is firstly proposed to serve as a simplified version of noise contrastive estimation [12, 25], which is an efficient way to compute the partition function of an unnormalized distribution to accelerate the training of word2vec. Mikolov et al. [24] set the negative sampling distribution proportional to the 3/4 power of degree by tuning the power parameter. Most later works on network embedding [27, 32] readily keep this setting. However, results in several papers [4, 40, 43] demonstrate that not only the distribution of negative sampling has a huge influence on the performance, but also the best choice varies largely on different datasets [4].

To the best of our knowledge, few papers systematically analyzed or discussed negative sampling in graph representation learning. In this paper, we first examine the role of negative sampling

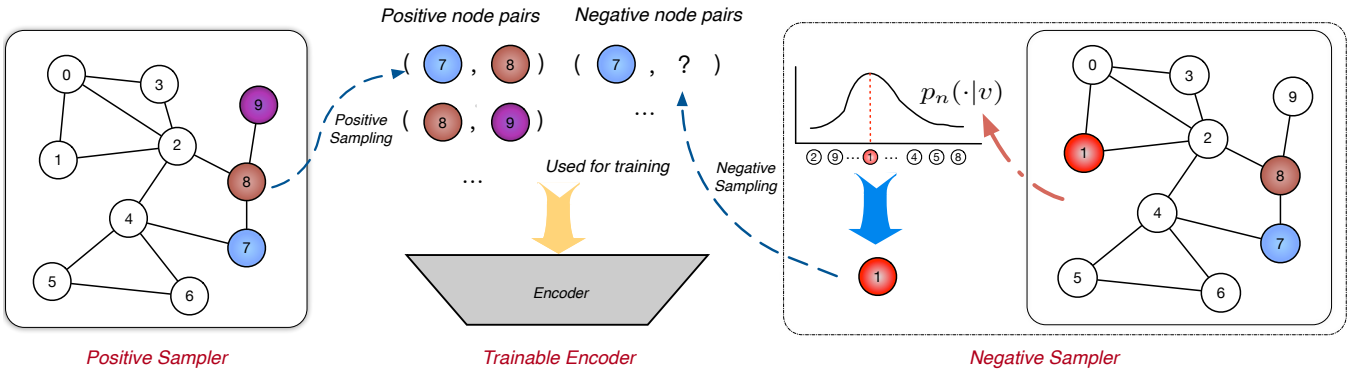


**Figure 1: The SampledNCE framework. Positive pairs are sampled implicitly or explicitly according to the graph representation methods, while negative pairs are from a pre-defined distribution, both composing the training data of contrastive learning.**

from the perspectives of objective and risk. Negative sampling is as important as positive sampling to determine the optimization objective and reduce the variance of the estimation in real-world graph data. According to our theory, the negative sampling distribution should be **positively but sub-linearly correlated** to their positive sampling distribution. Although our theory contradicts with the intuition "positively sampling nearby nodes and negatively sampling the faraway nodes", it accounts for the observation in previous works [10, 40, 43] where additional *hard* negative samples improve performance.

We propose an effective and scalable negative sampling strategy, **M**arkov **c**hain Monte Carlo **N**egative **S**ampling (MCNS), which applies our theory with an approximated positive distribution based on current embeddings. To reduce the time complexity, we leverage a special Metropolis-Hastings algorithm [23] for sampling. Most graphs naturally satisfy the assumption that adjacent nodes share similar positive distributions, thus the Markov chain from neighbors can be continued to skip the *burn-in* period and finally accelerates the sampling process without degradations in performance.

We evaluate MCNS on three most general graph-related tasks, link prediction, node classification and recommendation. Regardless of the network embedding or GNN model used, significant improvements are shown by replacing the original negative sampler to MCNS. We also collect many negative sampling strategies from information retrieval [35, 38], ranking in recommendation [15, 41] and knowledge graph embeddings [3, 42]. Our experiments in five datasets demonstrate that MCNS stably outperforms these negative sampling strategies.

## 2 FRAMEWORK

In this section, we introduce the preliminaries to understand negative sampling, including the unique embedding transformation, the SampledNCE framework and how the traditional network embeddings and GNNs are unified into the framework.

### 2.1 Unique Embedding Transformation

Many network embedding algorithms, such as DeepWalk [27], LINE [32] and node2vec [11], actually assign each node two embeddings [1], node embedding and contextual embedding. When sampled as context, the contextual embedding, instead of its node embedding is used for inner product. This implementation technique stems from word2vec [24], though sometimes omitted by following papers, and improves performance by utilizing the asymmetry of the contextual nodes and central nodes.

We can split each node $i$ into central node $v_i$ and contextual node $u_i$, and each edge $i \to j$ becomes $v_i \to u_j$. Running those network embedding algorithms on the transformed bipartite graph is strictly equivalent to running on the original graph.[2] According to the above analysis, we only need to anaylze graphs with unique embeddings without loss of generality.

**Algorithm 1:** The Sampled Noise Contrastive Estimation Framework

**Input:** Graph $G = (V, E)$, batch size $m$, Encoder $E_\theta$,
Positive Sampling Distribution $\hat{p_d}$,
Negative Sampling Distribution $p_n$,
Number of Negative Samples $k$.
**repeat**
  Sample $m$ nodes from $p(v) \propto deg(v)$
  Sample 1 positive node from $\hat{p_d}(u|v)$ for each node $v$
  Sample $k$ negative nodes from $p_n(u'|v)$ for each node $v$
  Initialize loss $\mathcal{L}$ as zero
  **for** *each node* $v$ **do**
    **for** *each positive node* $u$ *for* $v$ **do**
      $\mathcal{L} = \mathcal{L} - \log \sigma(E_\theta(u) \cdot E_\theta(v))$
    **end**
    **for** *each negative node* $u'$ *for* $v$ **do**
      $\mathcal{L} = \mathcal{L} - \log(1 - \sigma(E_\theta(u') \cdot E_\theta(v)))$
    **end**
  **end**
  Update $\theta$ by descending the gradients $\nabla_\theta \mathcal{L}$
**until** *Convergence*;

### 2.2 The SampledNCE Framework

We unify a large part of graph representation learning algorithms into the general SampledNCE framework, shown in Algorithm 1.

[1] Others, for example GraphSAGE [13] use a unique embedding.

[2] For bipartite graphs, always take nodes from $V$ part as central node and sample negative nodes from $U$ part.

The framework depends on an *encoder* $E_\theta$, which can be either an embedding lookup table or a variant of GNN, to generate node embeddings.

In the training process, $m$ positive nodes and $k \cdot m$ negative nodes are sampled. In Algorithm 1, we use the cross-entropy loss to optimize the inner product where $\sigma(x) = \frac{1}{1+e^{-x}}$. And the other choices of loss function work in similar ways. In link prediction or recommendation, we recall the top $K$ nodes with largest $E_\theta(v) \cdot E_\theta(u)$ for each node $u$. In classification, we evaluate the performance of logistic regression with the node embeddings as features.

### 2.3 Network Embedding and GNNs

Edges in natural graph data can be assumed as sampled from an underlying positive distribution $p_d(u, v)$. However, only a few edges are observable so that numerous techniques are developed to make reasonable assumptions to better estimate $p_d(u, v)$, resulting in various $\hat{p_d}$ in different algorithms to approximate $p_d$.

Network embedding algorithms employ various positive distributions implicitly or explicitly. LINE [32] samples positive nodes directly from adjacent nodes. DeepWalk [27] samples via random walks and implicitly defines $\hat{p_d}(u|v)$ as the stationary distribution of random walks [28]. node2vec [11] argues that homophily and structural equivalence of nodes $u$ and $v$ indicates a larger $p_d(u, v)$. These assumptions are mainly based on local smoothness and augment observed positive node pairs for training.

Graph neural networks are equipped with different encoder layers and perform implicit regularizations on positive distribution. Previous work [20] reveals that GCN is a variant of graph Laplacian smoothing, which directly constrains the difference of the embedding of adjacent nodes. GCN, or its sampling version GraphSAGE [13], does not explicitly augment positive pairs but regularizes the output embedding by local smoothness, which can be seen as an implicit regularization of $p_d$.

## 3 UNDERSTANDING NEGATIVE SAMPLING

In contrast to elaborate estimation of the positive sampling distribution, negative sampling has long been overlooked. In this section, we revisit negative sampling from the perspective of objective and risk, and conclude that "the negative sampling distribution should be positively but sub-linearly correlated to their positive sampling distribution".

### 3.1 How does negative sampling influence the objective of embedding learning?

Previous works [28] interpret network embedding as implicit matrix factorization. We discuss a more general form where the factorized matrix is determined by estimated data distribution $\hat{p_d}(u|v)$ and negative distribution $p_n(u'|v)$. According to [19] [28], the objective function for embedding is

$$\begin{aligned} J &= \mathbb{E}_{(u,v)\sim p_d} \log \sigma(\vec{u}^\top \vec{v}) + \mathbb{E}_{v\sim p_d(v)} [k\mathbb{E}_{u'\sim p_n(u'|v)} \log \sigma(-\vec{u'}^\top \vec{v})] \\ &= \mathbb{E}_{v\sim p_d(v)} [\mathbb{E}_{u\sim p_d(u|v)} \log \sigma(\vec{u}^\top \vec{v}) + k\mathbb{E}_{u'\sim p_n(u'|v)} \log \sigma(-\vec{u'}^\top \vec{v})], \end{aligned} \tag{1}$$

where $\vec{u}, \vec{v}$ are embeddings for node $u$ and $v$ and $\sigma(\cdot)$ is the sigmoid function. We can separately consider each node $v$ and show the objective for optimization as follows:

Theorem 1. ***(Optimal Embedding)*** *The optimal embedding satisfies that for each node pair* $(u, v)$,

$$\vec{u}^\top \vec{v} = -\log \frac{k \cdot p_n(u|v)}{p_d(u|v)}. \tag{2}$$

**Proof** Let us consider the conditional data and the negative distribution of node $v$. To maximize $J$ is equivalent to minimize the following objective function for each $v$,

$$\begin{aligned} J^{(v)} &= \mathbb{E}_{u\sim p_d(u|v)} \log \sigma(\vec{u}^\top \vec{v}) + k\mathbb{E}_{u'\sim p_n(u'|v)} \log \sigma(-\vec{u'}^\top \vec{v}) \\ &= \sum_u p_d(u|v) \log \sigma(\vec{u}^\top \vec{v}) + k \sum_{u'} p_n(u'|v) \log \sigma(-\vec{u'}^\top \vec{v}) \\ &= \sum_u [p_d(u|v) \log \sigma(\vec{u}^\top \vec{v}) + kp_n(u|v) \log \sigma(-\vec{u}^\top \vec{v})] \\ &= \sum_u [p_d(u|v) \log \sigma(\vec{u}^\top \vec{v}) + kp_n(u|v) \log \left(1 - \sigma(\vec{u}^\top \vec{v})\right)]. \end{aligned}$$

For each $(u, v)$ pair, if we define two Bernoulli distributions $P_{u,v}(x)$ where $P_{u,v}(x = 1) = \frac{p_d(u|v)}{p_d(u|v)+kp_n(u|v)}$ and $Q_{u,v}(x)$ where $Q_{u,v}(x = 1) = \sigma(\vec{u}^\top \vec{v})$, the equation above is simplified as follows:

$$J^{(v)} = -\sum_u (p_d(u|v) + kp_n(u|v)) H(P_{u,v}, Q_{u,v}),$$

where $H(p, q) = -\sum_{x\in\mathcal{X}} p(x) \log q(x)$ is the cross entropy between two distributions. According to *Gibbs Inequality*, the maximizer of $L^{(v)}$ satisfies $P = Q$ for each $(u, v)$ pair, which gives

$$\begin{aligned} \frac{1}{1 + e^{-\vec{u}^T \vec{v}}} &= \frac{p_d(u|v)}{p_d(u|v) + kp_n(u|v)} \\ \vec{u}^\top \vec{v} &= -\log \frac{k \cdot p_n(u|v)}{p_d(u|v)}. \end{aligned} \tag{3}$$

□

The above theorem clearly shows that the positive and negative distributions $p_d$ and $p_n$ have the same level of influence on optimization. In contrast to abundant research on finding a better $\hat{p_d}$ to approximate $p_d$, negative distribution $p_n$ is apparently underexplored.

### 3.2 How does negative sampling influence the expected loss (risk) of embedding learning?

The analysis above shows that with enough (possibly infinite) samples of edges from $p_d$, the inner product of embedding $\vec{u}^\top \vec{v}$ has an optimal value. In real-world data, we only have limited samples from $p_d$, which leads to a nonnegligible *expected loss* (*risk*). Then the loss function to minimize *empirical risk* for node $v$ becomes:

$$J_T^{(v)} = \frac{1}{T} \sum_{i=1}^{T} \log \sigma(\vec{u}_i^\top \vec{v}) + \frac{1}{T} \sum_{i=1}^{kT} \log \sigma(-\vec{u'}_i^\top \vec{v}), \tag{4}$$

where $\{u_1, ..., u_T\}$ are sampled from $p_d(u|v)$ and $\{u'_1, ..., {u'}_k^\top\}$ are sampled from $p_n(u|v)$.

In this section, we only consider optimization of node $v$, which can be directly generalized to the whole embedding learning. More precisely, we equivalently consider $\boldsymbol{\theta} = [\vec{u_0}^\top \vec{v}, ..., \vec{u}_{N-1}^\top \vec{v}]$ as the parameters to be optimized, where $\{u_0, ...u_{N-1}\}$ are all the N nodes

in the graph. Suppose the optimal parameter for $J_T^{(v)}$ and $J^{(v)}$ are $\boldsymbol{\theta}_T$ and $\boldsymbol{\theta}^*$ respectively. The gap between $\boldsymbol{\theta}_T$ and $\boldsymbol{\theta}^*$ is described as follows:

Theorem 2. *The random variable $\sqrt{T}(\boldsymbol{\theta}_T - \boldsymbol{\theta}^*)$ asymptotically converges to a distribution with zero mean vector and covariance matrix*

$$Cov\big(\sqrt{T}(\boldsymbol{\theta}_T - \boldsymbol{\theta}^*)\big) = \boldsymbol{diag}(\boldsymbol{m})^{-1} - (1 + 1/k)\mathbf{1}^\top\mathbf{1}, \tag{5}$$

*where* $\boldsymbol{m} = \big[\frac{kp_d(u_0|v)p_n(u_0|v)}{p_d(u_0|v)+kp_n(u_0|v)}, ..., \frac{kp_d(u_{N-1}|v)p_n(u_{N-1}|v)}{p_d(u_{N-1}|v)+kp_n(u_{N-1}|v)}\big]^\top$ *and* $\mathbf{1} = [1, ..., 1]^\top$.

**Proof** Our analysis is based on the Taylor expansion of $\nabla_\theta J_T^{(v)}(\boldsymbol{\theta})$ around $\boldsymbol{\theta}^*$. As $\boldsymbol{\theta}_T$ is the minimizer of $J_T^{(v)}(\boldsymbol{\theta})$, we must have $\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}_T) = \mathbf{0}$, which gives

$$\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}_T) = \nabla J_T^{(v)}(\boldsymbol{\theta}^*) + \nabla_\theta^2 J_T^{(v)}(\boldsymbol{\theta}^*)(\boldsymbol{\theta}_T - \boldsymbol{\theta}^*) + O(\|\boldsymbol{\theta}_T - \boldsymbol{\theta}^*\|^2) = \mathbf{0}. \tag{6}$$

Up to terms of order $O(\|\boldsymbol{\theta}^* - \boldsymbol{\theta}_T\|^2)$, we have

$$\sqrt{T}(\boldsymbol{\theta}_T - \boldsymbol{\theta}^*) = -\big(\nabla_\theta^2 J_T^{(v)}(\boldsymbol{\theta}^*)\big)^{-1}\sqrt{T}\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*). \tag{7}$$

Next, we will analyze $-\big(\nabla_\theta^2 J_T^{(v)}(\boldsymbol{\theta}^*)\big)^{-1}$ and $\sqrt{T}\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*)$ respectively by the following lemmas.

Lemma 1. *The negative Hessian matrix $-\nabla_\theta^2 J_T^{(v)}(\boldsymbol{\theta}^*)$ converges in probability to $\boldsymbol{diag}(\boldsymbol{m})$.*

**Proof** First, we calculate the gradient of $J_T^{(v)}(\boldsymbol{\theta})$ and Hessian matrix $\nabla_\theta^2 J_T^{(v)}(\boldsymbol{\theta})$. Let $\boldsymbol{\theta}_u$ be the parameter in $\boldsymbol{\theta}$ for modeling $\vec{\boldsymbol{u}}^\top\vec{\boldsymbol{v}}$ and $\boldsymbol{e}_{(u)}$ be the one-hot vector which only has a 1 on this dimension.

$$\begin{aligned}
J_T^{(v)}(\boldsymbol{\theta}) =& \frac{1}{T}\sum_{i=1}^{T}\log\sigma(\boldsymbol{\theta}_{u_i}) + \frac{1}{T}\sum_{i=1}^{kT}\log\sigma(-\boldsymbol{\theta}_{u_i'}) \\
\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}) =& \frac{1}{T}\sum_{i=1}^{T}\big(1-\sigma(\boldsymbol{\theta}_{u_i})\big)\boldsymbol{e}_{(u_i)} - \frac{1}{T}\sum_{i=1}^{kT}\sigma(\boldsymbol{\theta}_{u_i'})\boldsymbol{e}_{(u_i')} \\
\nabla_\theta^2 J_T^{(v)}(\boldsymbol{\theta}) =& \frac{1}{T}\sum_{i=1}^{T}\{-\sigma(\boldsymbol{\theta}_{u_i})\left(1-\sigma(\boldsymbol{\theta}_{u_i})\right)\boldsymbol{e}_{(u_i)}\boldsymbol{e}_{(u_i)}^\top + \mathbf{0}^\top\mathbf{0}\} \\
& - \frac{1}{T}\sum_{i=1}^{kT}\{\sigma(\boldsymbol{\theta}_{u_i'})\left(1-\sigma(\boldsymbol{\theta}_{u_i'})\right)\boldsymbol{e}_{(u_i')}\boldsymbol{e}_{(u_i')}^\top + \mathbf{0}^\top\mathbf{0}\}.
\end{aligned} \tag{8}$$

According to equation (3), $\sigma(\boldsymbol{\theta}_{u_i}) = \frac{p_d(u|v)}{p_d(u|v)+kp_n(u|v)}$ at $\boldsymbol{\theta} = \boldsymbol{\theta}^*$. Therefore,

$$\begin{aligned}
\lim_{T\to+\infty} -\nabla_\theta^2 J_T^{(v)}(\boldsymbol{\theta}^*) \xrightarrow{P} & \sum_u \sigma(\boldsymbol{\theta}_u)\left(1-\sigma(\boldsymbol{\theta}_u)\right)\boldsymbol{e}_{(u)}\boldsymbol{e}_{(u)}^\top \\
& \cdot (p_d(u) + kp_n(u)) \\
= & \sum_u \frac{kp_d(u|v)p_n(u|v)}{p_d(u|v)+kp_n(u|v)}\boldsymbol{e}_{(u)}\boldsymbol{e}_{(u)}^\top \\
= & \boldsymbol{diag}(\boldsymbol{m}).
\end{aligned} \tag{9}$$

□

Lemma 2. *The expectation and variance of $\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*)$ satisfy*

$$\mathbb{E}[\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*)] = \mathbf{0}, \tag{10}$$

$$Var\,[\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*)] = \frac{1}{T}\big(\boldsymbol{diag}(\boldsymbol{m}) - (1+1/k)\boldsymbol{m}\boldsymbol{m}^\top\big). \tag{11}$$

**Proof** According to equation (3)(8),

$$\mathbb{E}[\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*)] = \sum_u [p_d(u)\big(1-\sigma(\boldsymbol{\theta}_u^*)\big)\boldsymbol{e}_{(u)} - kp_n(u)\sigma(\boldsymbol{\theta}_u^*)\boldsymbol{e}_{(u)}] = \mathbf{0},$$

$$\begin{aligned}
\text{Cov}\,[\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*)] =& \mathbb{E}[\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*)\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*)^\top] \\
=& \frac{1}{T}\mathbb{E}_{u\sim p_d(u|v)}\left(1-\sigma(\boldsymbol{\theta}_u^*)\right)^2\boldsymbol{e}_{(u)}\boldsymbol{e}_{(u)}^\top \\
& + \frac{k}{T}\mathbb{E}_{u\sim p_n(u|v)}\sigma(\boldsymbol{\theta}_u^*)^2\boldsymbol{e}_{(u)}\boldsymbol{e}_{(u)}^\top \\
& + (\frac{T-1}{T} - 2 + 1 - \frac{1}{kT})\boldsymbol{m}\boldsymbol{m}^\top \\
=& \frac{1}{T}\big(\mathbf{diag}(\boldsymbol{m}) - (1+\frac{1}{k})\boldsymbol{m}\boldsymbol{m}^\top\big).
\end{aligned} \tag{12}$$

□

Let us continue to prove Theorem 2 by substituting (9)(12) into equation (7) and derive the covariance of $\sqrt{T}(\boldsymbol{\theta}_T - \boldsymbol{\theta}^*)$.

$$\begin{aligned}
\text{Cov}\big(\sqrt{T}(\boldsymbol{\theta}_T - \boldsymbol{\theta}^*)\big) =& \mathbb{E}[\sqrt{T}(\boldsymbol{\theta}_T - \boldsymbol{\theta}^*)\ \sqrt{T}(\boldsymbol{\theta}_T - \boldsymbol{\theta}^*)^\top] \\
\approx & T\mathbf{diag}(\boldsymbol{m})^{-1}\text{Var}\,[\nabla_\theta J_T^{(v)}(\boldsymbol{\theta}^*)]\big(\mathbf{diag}(\boldsymbol{m})^{-1}\big)^\top \\
=& \mathbf{diag}(\boldsymbol{m})^{-1} - (1+1/k)\mathbf{1}^\top\mathbf{1}.
\end{aligned}$$

□

According to Theorem 2, the mean squared error, aka risk, of $\vec{\boldsymbol{u}}^\top\vec{\boldsymbol{v}}$ is

$$\mathbb{E}\big[\|(\boldsymbol{\theta}_T - \boldsymbol{\theta}^*)_u\|^2\big] = \frac{1}{T}(\frac{1}{p_d(u|v)} - 1 + \frac{1}{kp_n(u|v)} - \frac{1}{k}). \tag{13}$$

## 3.3 The Principle of Negative Sampling

To determine an appropriate $p_n$ for a specific $p_d$, a trade-off might be needed to balance the rationality of the *objective* (to what extent the optimal embedding fits for the downstream task) and the expected loss, also known as *risk* (to what extent the trained embedding deviates from the optimum).

A simple solution is to sample negative nodes positively but sub-linearly correlated to their positive sampling distribution, i.e. $p_n(u|v) \propto p_d(u|v)^\alpha, 0 < \alpha < 1$.

**(Monotonicity)** From the perspective of objective, if we have $p_d(u_i|v) > p_d(u_j|v)$,

$$\begin{aligned}
\vec{\boldsymbol{u}}_i^\top\vec{\boldsymbol{v}} &= \log p_d(u_i|v) - \alpha\log p_d(u_i|v) + c \\
&> (1-\alpha)\log p_d(u_j|v) + c = \vec{\boldsymbol{u}}_j^\top\vec{\boldsymbol{v}},
\end{aligned}$$

where $c$ is a constant. The sizes of inner products is in accordance with positive distribution $p_d$, facilitating the task relying on relative sizes of $\vec{\boldsymbol{u}}^\top\vec{\boldsymbol{v}}$ for different $u$s, such as recommendation or link prediction.

**(Accuracy)** From the perspective of risk, we mainly care about the scale of the expected loss. According to equation (13), the expected squared deviation is dominated by the smallest one in $p_d(u|v)$ and $p_n(u|v)$. If an intermediate node with large $p_d(u|v)$ is

**Algorithm 2:** The training process with MCNS

**Input:** DFS sequence $T = [v_1, ..., v_{|T|}]$,
Encoder $E_\theta$, Positive Sampling Distribution $\hat{p_d}$,
Proposal Distribution $q$, Number of Negative Samples $k$.
**repeat**
  Initialize current negative node $x$ at random
  **for** *each node $v$ in DFS sequence $T$* **do**
    Sample 1 positive sample $u$ from $\hat{p_d}(u|v)$
    Initialize loss $\mathcal{L} = 0$
    //Negative Sampling via Metropolis-Hastings
    **for** $i = 1$ *to* $k$ **do**
      Sample a node $y$ from $q(y|x)$
      Generate a uniform random number $r \in [0, 1]$
      **if** $r < min\big(1, \frac{(E_\theta(v)\cdot E_\theta(y))^\alpha}{(E_\theta(v)\cdot E_\theta(x))^\alpha}\frac{q(x|y)}{q(y|x)}\big)$ **then**
        $x = y$
      **end**
      $\mathcal{L} = \mathcal{L} + \max\big(0, E_\theta(v)\cdot E_\theta(x) - E_\theta(u)\cdot E_\theta(v) + \gamma\big)$
      Update $\theta$ by descending the gradients $\nabla_\theta \mathcal{L}$
    **end**
  **end**
**until** *Early Stopping or Convergence*;

negatively sampled insufficiently (with small $p_n(u|v)$), the accurate optimization will be corrupted. But if $p_n(u|v) \propto p_d(u|v)^\alpha$, then

$$\mathbb{E}\big[\|(\theta_T - \theta^*)_u\|^2\big] = \frac{1}{T}\Big(\frac{1}{p_d(u|v)}\big(1 + \frac{p_d(u|v)^{1-\alpha}}{c}\big) - 1 - \frac{1}{k}\Big).$$

The order of magnitude of deviation only negatively related to $p_d(u|v)$, meaning that with limited positive samples, the inner products for high-probability node pairs are estimated more accurate. This is an evident advantage over evenly negative sampling.

# 4 METHOD

## 4.1 The Self-contrast Approximation

Although we deduced that $p_n(u|v) \propto p_d(u|v)^\alpha$, the real $p_d$ is unknown and its approximation $\hat{p_d}$ is often implicitly defined. – How can the principle help negative sampling?

We therefore propose a *self-contrast approximation*, replacing $p_d$ by inner products based on the current encoder, i.e.

$$p_n(u|v) \propto p_d(u|v)^\alpha \approx \frac{\big(E_\theta(u)\cdot E_\theta(v)\big)^\alpha}{\sum_{u'\in U}\big(E_\theta(u')\cdot E_\theta(v)\big)^\alpha}.$$

The resulting form is similar to a technique in RotatE [31], and very time-consuming. Each sampling requires $O(n)$ time, making it impossible for middle- or large-scale graphs. Our MCNS solves this problem by our adapted version of Metropolis-Hastings algorithm.

## 4.2 The Metropolis-Hastings Algorithm

As one of the most distinguished Markov chain Monte Carlo (MCMC) methods, the Metropolis–Hastings algorithm [23] is designed for obtaining a sequence of random samples from unnormalized distributions. Suppose we want to sample from distribution $\pi(x)$ but only know $\tilde{\pi}(x) \propto \pi(x)$. Meanwhile the computation of normalizer $Z = \sum_x \tilde{\pi}(x)$ is unaffordable. The Metropolis–Hastings algorithm constructs a Markov chain $\{X(t)\}$ that is ergodic and stationary with respect to $\pi$ —- meaning that, $X(t) \sim \pi(x), t \to \infty$.

The transition in the Markov chain has two steps:
(1) Generate $y \sim q(y|X(t))$ where the *proposal distribution* $q$ is arbitrary chosen as long as positive everywhere.
(2) Pick $y$ as $X(t+1)$ at the probability $\min\Big\{\frac{\tilde{\pi}(y)}{\tilde{\pi}(X(t))}\frac{q(X(t)|y)}{q(y|X(t))}, 1\Big\}$, or $X(t+1) \leftarrow X(t)$. See tutorial [6] for more details.

## 4.3 Markov chain Negative Sampling

The main idea for MCNS is to apply the Metropolis–Hastings algorithm for each node $v$ with $\tilde{\pi}(u|v) = \big(E_\theta(u)\cdot E_\theta(v)\big)^\alpha$. Then we are sampling from the self-contrast approximated distribution. However, a notorious disadvantage for the Metropolis–Hastings is a relatively long *burn – in* period, i.e. the distribution of $X(t) \approx \pi(x)$ only when $t$ is large, which heavily affects its efficiency.

Fortunately, the connatural property exists for graphs that nearby nodes usually share similar $p_d(\cdot|v)$, as evident from triadic closure in social network [16], collaborative filtering in recommendation [21], etc. Since nearby nodes have similar target negative sampling distribution, the Markov chain for one node is likely to still work well for its neighbors. Therefore, a neat solution is to traverse the graph by Depth First Search (DFS, See Appendix A.6) and keep generating negative samples from the last node's Markov chain (Figure 2).

The proposal distribution also influences convergence rate. Our $q(y|x)$ is defined by mixing uniform sampling and sampling from the nearest $k$ nodes with $\frac{1}{2}$ probability each. The hinge loss pulls positive node pairs closer and pushes negative node pairs away until they are beyond a pre-defined margin. Thus, we substitute the binary cross-entropy loss as a $\gamma$-skewed hinge loss,

$$\mathcal{L} = \max\big(0, E_\theta(v)\cdot E_\theta(x) - E_\theta(u)\cdot E_\theta(v) + \gamma\big), \tag{14}$$

where $(u, v)$ is a positive node pair and $(x, v)$ is negative. $\gamma$ is a hyperparameter. The MCNS is summarized in Algorithm 2.

# 5 EXPERIMENTS

To demonstrate the efficacy of MCNS, we conduct a comprehensive suite of experiments on 3 representative tasks, 3 graph representation learning algorithms and 5 datasets, a total of 19 experimental settings, under all of which MCNS consistently achieves significant improvements over 8 negative sampling baselines collected from previous papers [27, 40] and other relevant topics [3, 35, 38, 41].

## 5.1 Experimental Setup

*5.1.1* ***Tasks and Dataset.*** The statistics of tasks and datasets are shown in Table 1. Introductions about the datasets are in Appendix A.4.

**Table 1: Statistics of the tasks and datasets.**

| Task | Dataset | Nodes | Edges | Classes |
|---|---|---|---|---|
| Recommendation | MovieLens | 2,625 | 100,000 | / |
| | Amazon | 255,404 | 1,689,188 | / |
| | Alibaba | 159,633 | 907,470 | / |
| Link Prediction | Arxiv | 5,242 | 28,980 | / |
| Node Classification | BlogCatalog | 10,312 | 333,983 | 39 |

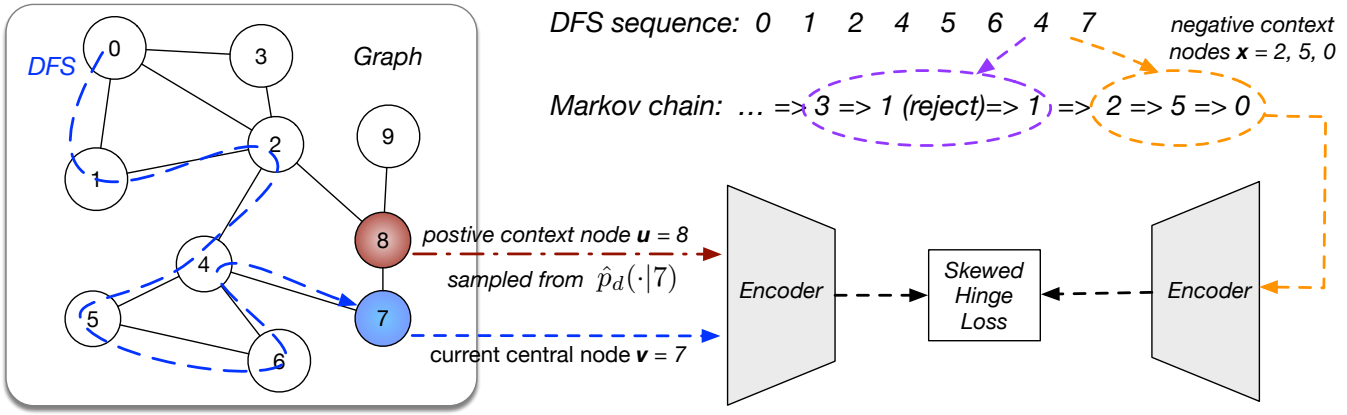


**Figure 2: A running example of MCNS. The central nodes are traversed by DFS, each samples three negative context nodes using Metropolis-Hastings algorithm by proceeding with the Markov chain.**

*5.1.2* ***Encoders.*** To verify the adaptiveness of MCNS to different genres of graph representation learning algorithms, we utilize an embedding lookup table or a variant of GNN as encoder for experiments, including DeepWalk, GCN and GraphSAGE. The detailed algorithms are shown in Appendix A.2.

*5.1.3* ***Baselines.*** Baselines in our experiments generally fall into the following categories. The detailed description of each baseline is provided in Appendix A.3.

- **Degree-based Negative Sampling.** The compared methods include Power of Degree [24], RNS [4] and WRMF [15], which can be summarized as $p_n(v) \propto deg(v)^\beta$. These strategies are static, inconsiderate to the personalization of nodes.
- **Hard-samples Negative Sampling.** Based on the observation that "hard negative samples" – the nodes with high positive probabilities – helps recommendation, methods are proposed to mine the hard negative samples by rejection (WARP [44]), sampling-max (DNS [41]) and personalized PageRank (PinSAGE [40]).
- **GAN-based Negative Sampling.** IRGAN [35] trains generative adversarial nets (GANs) to adaptively generate hard negative samples. KBGAN [3] employs a sampling-based adaptation of IRGAN on knowledge base completion.

## 5.2 Results

In this section, we demonstrate the results in the 19 settings. Note that all values and standard deviations reported in the tables are from ten-fold cross validation. We also leverage paired t-tests [14] to verify whether MCNS is significantly better than the strongest baseline. The tests in the 19 settings give a max *p-value* (Amazon, GraphSAGE) of $0.0005 \ll 0.01$, quantitatively proving the significance of our improvements.

*5.2.1* ***Recommendation.*** Recommendation is the most important technology in many E-commerce platforms, which evolved from collaborative filtering to graph-based models. Graph-based recommender systems represent all users and items by embeddings, and recommend items with largest inner products for a given user.

In this paper, $Hits@k$ [7] and mean reciprocal ranking (*MRR*) [30] serve as evaluation methodologies, whose definitions can be found in Appendix A.5. Moreover, we only use observed positive pairs to train node embeddings, which can clearly demonstrate that the improvement of performance comes from the proposed negative sampling. We summarize the performance of MCNS as well as the baselines in Table 2.

The results show that Hard-samples negative samplings generally surpass degree-based strategies with 5% ∼ 40% performance leaps in MRR. GAN-based negative sampling strategies perform even better, owing to the evolving generator mining hard negative samples more accurately. In the light of our theory, the proposed MCNS accomplishes significant gains of 2% ∼ 13% over the best baselines.

*5.2.2* ***Link Prediction.*** Link prediction aims to predict the occurrence of links in graphs, more specifically, to judge whether a node pair is a masked true edge or unlinked pair based on nodes' representations. Details about data split and evaluation are in Appendix A.5.

The results for link prediction are given in Table 3. MCNS outperforms all baselines with various graph representation learning methods on the Arxiv dataset. An interesting phenomenon about degree-based strategies is that both WRMF and commonly-used $Deg^{0.75}$ outperform uniform RNS sampling, completely adverse to the results on recommendation datasets. The results can be verified by examining the results in previous papers [24, 43], reflecting the different network characteristics of user-item graphs and citation networks.

*5.2.3* ***Node Classification.*** Multi-label node classification is a usual task to assess graph representation algorithms. The one-vs-rest logistic regression classifier[9] is trained in a supervised way with graph representations as the features of nodes. In this experiment, we keep the default setting as the DeepWalk's original paper for fair comparison on the BlogCatalog dataset.

Table 4 summarizes the Micro-F1 result on the BlogCatalog dataset. MCNS stably outperforms all baselines regardless of the training set ratio $T_R$. The trends are similar for Macro-F1, which is omitted due to the space limitation.

| | | MovieLens | | | Amazon | | Alibaba | |
|---|---|---|---|---|---|---|---|---|
| | | DeepWalk | GCN | GraphSAGE | DeepWalk | GraphSAGE | DeepWalk | GraphSAGE |
| *MRR* | $Deg^{0.75}$ | 0.025±.001 | 0.062±.001 | 0.063±.001 | 0.041±.001 | 0.057±.001 | 0.037±.001 | 0.064±.001 |
| | WRMF | 0.022±.001 | 0.038±.001 | 0.040±.001 | 0.034±.001 | 0.043±.001 | 0.036±.001 | 0.057±.002 |
| | RNS | 0.031±.001 | 0.082±.002 | 0.079±.001 | 0.046±.003 | 0.079±.003 | 0.035±.001 | 0.078±.003 |
| | PinSAGE | 0.036±.001 | 0.091±.002 | 0.090±.002 | 0.057±.004 | 0.080±.001 | 0.054±.001 | 0.081±.001 |
| | WARP | 0.041±.003 | 0.114±.003 | 0.111±.003 | 0.061±.001 | 0.098±.002 | 0.067±.001 | 0.106±.001 |
| | DNS | 0.040±.003 | 0.113±.003 | 0.115±.003 | 0.063±.001 | 0.101±.003 | 0.067±.001 | 0.090±.002 |
| | IRGAN | 0.047±.002 | 0.111±.002 | 0.101±.002 | 0.059±.001 | 0.091±.001 | 0.061±.001 | 0.083±.001 |
| | KBGAN | 0.049±.001 | 0.114±.003 | 0.100 ±.001 | 0.060±.001 | 0.089±.001 | 0.065±.001 | 0.087±.002 |
| | MCNS | **0.053±.001** | **0.122±.004** | **0.114±.001** | **0.065±.001** | **0.108±.001** | **0.070±.001** | **0.116±.001** |
| *Hits*@30 | $Deg^{0.75}$ | 0.115±.002 | 0.270±.002 | 0.270±.001 | 0.161±.003 | 0.238±.002 | 0.138±.003 | 0.249±.004 |
| | WRMF | 0.110±.003 | 0.187±.002 | 0.181±.002 | 0.139±.002 | 0.188±.001 | 0.121±.003 | 0.227±.004 |
| | RNS | 0.143±.004 | 0.362±.004 | 0.356±.001 | 0.171±.004 | 0.317±.004 | 0.132±.004 | 0.302±.005 |
| | PinSAGE | 0.158±.003 | 0.379±.005 | 0.383±.005 | 0.176±.004 | 0.333±.005 | 0.146±.003 | 0.312±.005 |
| | WARP | 0.164±.005 | 0.406±.002 | 0.404±.005 | 0.181±.004 | 0.340±.004 | 0.178±.004 | 0.342±.004 |
| | DNS | 0.166±.005 | 0.404±.006 | 0.410±.006 | 0.182±.003 | 0.358±.004 | 0.186±.005 | 0.336±.004 |
| | IRGAN | 0.207±.002 | 0.415±.004 | 0.408±.004 | 0.183±.004 | 0.342±.003 | 0.175±.003 | 0.320±.002 |
| | KBGAN | 0.198±.003 | 0.420±.003 | 0.401±.005 | 0.181±.003 | 0.347±.003 | 0.181±.003 | 0.331±.004 |
| | MCNS | **0.230±.003** | **0.426 ±.005** | **0.413±.003** | **0.207±.003** | **0.386±.004** | **0.201±.003** | **0.387±.002** |

**Table 2: Recommendation Results of MCNS with various encoders on three datasets. GCN is not evaluted on Amazon and Alibaba datasets due to its limited scalability. Similar trends hold for Hits@5 and Hits@10.**

| | Algorithm | DeepWalk | GCN | GraphSAGE |
|---|---|---|---|---|
| AUC | $Deg^{0.75}$ | 64.6±0.1 | 79.6±0.4 | 78.9±0.4 |
| | WRMF | 65.3±0.1 | 80.3±0.4 | 79.1±0.2 |
| | RNS | 62.2±0.2 | 74.3±0.5 | 74.7±0.5 |
| | PinSAGE | 67.2±0.4 | 80.4±0.3 | 80.1±0.4 |
| | WARP | 70.5±0.3 | 81.6±0.3 | 82.7±0.4 |
| | DNS | 70.4±0.3 | 81.5±0.3 | 82.6±0.4 |
| | IRGAN | 71.1±0.2 | 82.0±0.4 | 82.2±0.3 |
| | KBGAN | 71.6±0.3 | 81.7±0.3 | 82.1±0.3 |
| | MCNS | **73.1±0.4** | **82.6±0.4** | **83.5±0.5** |

**Table 3: The results of link prediction with different negative sampling strategies on the Arxiv dataset.**

| | Algorithm | DeepWalk | | | GCN | | | GraphSAGE | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $T_R$(%) | 10 | 50 | 90 | 10 | 50 | 90 | 10 | 50 | 90 |
| Micro -F1 | $Deg^{0.75}$ | 31.6 | 36.6 | 39.1 | 36.1 | 41.8 | 44.6 | 35.9 | 42.1 | 44.0 |
| | WRMF | 30.9 | 35.8 | 37.5 | 34.2 | 41.4 | 43.3 | 34.4 | 41.0 | 43.1 |
| | RNS | 29.8 | 34.1 | 36.0 | 33.4 | 40.5 | 42.3 | 33.5 | 39.6 | 41.6 |
| | PinSAGE | 32.0 | 37.4 | 40.1 | 37.2 | 43.2 | 45.7 | 36.9 | 43.2 | 45.1 |
| | WARP | 35.1 | 40.3 | 42.1 | 39.9 | 45.8 | 47.7 | 40.1 | 45.5 | 47.5 |
| | DNS | 35.2 | 40.4 | 42.5 | 40.4 | 46.0 | 48.6 | 40.5 | 46.3 | 48.5 |
| | IRGAN | 34.3 | 39.6 | 41.8 | 39.1 | 45.2 | 47.9 | 38.9 | 45.0 | 47.6 |
| | KBGAN | 34.6 | 40.0 | 42.3 | 39.5 | 45.5 | 48.3 | 39.6 | 45.3 | 48.5 |
| | MCNS | **36.1** | **41.2** | **43.3** | **41.7** | **47.3** | **49.9** | **41.6** | **47.5** | **50.1** |

**Table 4: Micro-F1 scores for multi-label classification on the BlogCatalog dataset. Similar trends hold for Macro-F1 scores.**

## 5.3 Analysis

**Comparison with Power of Degree.** To thoroughly investigate the potential of degree-based strategies $p_n(v) \propto deg(v)^\beta$, a series of experiments are conducted by varying $\beta$ from -1 to 1 with GraphSAGE. Figure 3 shows that the performance of degree-based is consistently lower than that of MCNS, suggesting that MCNS learns a better negative distribution beyond the expressiveness of degree-based strategies. Moreover, the best $\beta$ varies between datasets and seems not easy to determine before experiments, while MCNS naturally adapts to different datasets.

**Parameter Analysis.** To test the robustness of MCNS quantitively, we visualize the MRR curves of MCNS by varying two most important hyperparameters, embedding dimension and margin $\gamma$. The results with GraphSAGE on Alibaba dataset are shown in Figure 4.

The skewed hinge loss in Equation (14) aims to keep at least $\gamma$ distance between the inner product of the positive pair and that of the negative pair. $\gamma$ controls the distance between positive-negative samples and must be positive in principle. Correspondingly, Figure 4 shows that the hinge loss begins to take effect when $\gamma \geq 0$ and reaches its optimum at $\gamma \approx 0.1$, which is a reasonable boundary. As the performance increases slowly with a large embedding dimension, we set it 512 due to the trade-off between the performance and time consumption.

**Efficiency Comparison.** To compare the efficiency of different negative sampling methods, we report the runtime of MCNS and hard-samples or GAN-based strategies (PinSAGE, WARP, DNS, KBGAN) with GraphSAGE encoder in recommendation task in Figure 5. We keep the same batch size and number of epochs for a fair comparison. WARP might need a large number of tries before finding negative samples, leading to a long running time. PinSAGE utilizes PageRank to generate "hard negative items", which increases running time compared with uniformly sampling. DNS and KBGAN

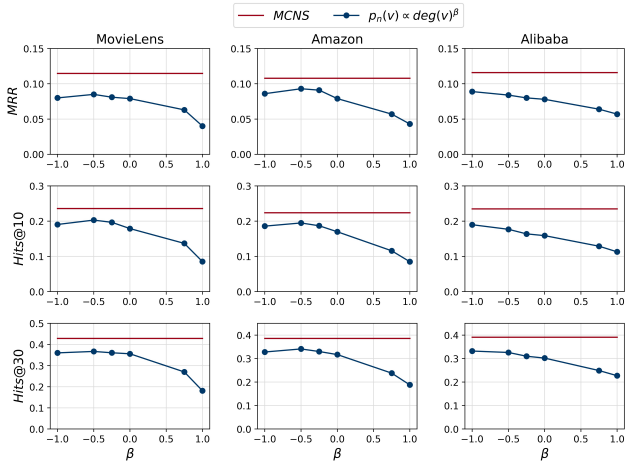


**Figure 3: Comparison between Power of Degree and MCNS. Each red line represents the performance of MCNS in this setting and the blue curves are performances of Power of Degree with various $\beta$.**

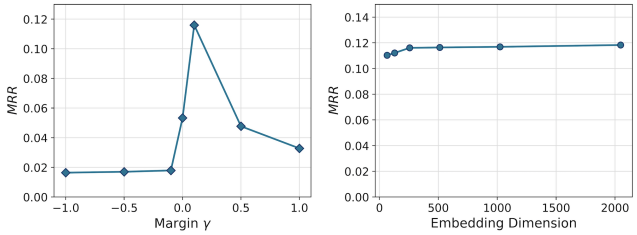


**Figure 4: The performance of MCNS on the Alibaba by varying $\gamma$ and embedding dimension.**

both first sample a batch of candidates and then select negative samples from them, but the latter uses a lighter generator. MCNS requires 17 mins to complete ten-epochs training while the fastest baseline KBGAN is at least 2× slower on the MovieLens dataset.

## 5.4 Experiments for Further Understanding

In section 3, we have analyzed the criterion about negative sampling from the perspective of objective and risk. However, doubts might arise about (1) whether sampling more negative samples is always helpful, (2) why our conclusion contradicts with the intuition "positively sampling nearby nodes and negatively sampling faraway nodes ".

To further deepen the understanding of negative sampling, we design two extended experiments on MovieLens with GraphSAGE encoder to verify our theory and present the results in Figure 6.

If we sample more negative samples, the performance increases with subsequently decrease. According to Equation (13), sampling more negative samples always decrease the risk, leading to an improvement in performance at first. However, performance begins to decrease after reaching the optimum, because extra bias is added to the objective by the increase of $k$ according to Equation (2). The bias impedes the optimization of the objective from zero-centered initial embeddings.

The second experiment in Figure 6 shows the disastrous consequences of negatively sampling more *distant nodes*, the nodes

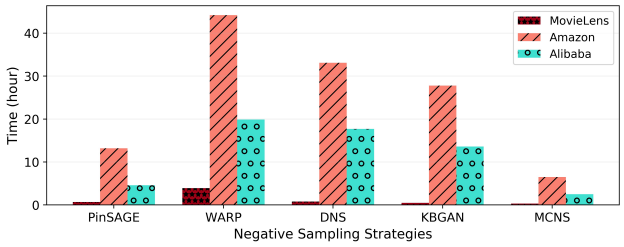


**Figure 5: Runtime comparisons for different negative sampling strategies with GraphSAGE encoder on the recommendation datasets.**

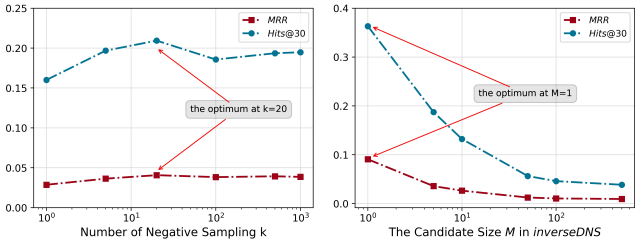


**Figure 6: The results of verification experiments. (Left) The effect of increasing the number of negative sampling $k$ in $Deg^{0.75}$. (Right) The effect of sampling more *distant nodes*.**

with small $p_d(u)$. To test the performances with varying degrees of the mismatching of $p_d$ and $p_n$, we design a strategy, *inverseDNS*, by selecting the one scored lowest in the candidate items. In this way, not only the negative sampling probabilities of the items are sampled negatively correlated to $p_d$, but also we can control the degree by varying candidate size $M$. Both MRR and $Hits@k$ go down as $M$ increases, hence verifies our theory.

## 6 RELATED WORK

**Graph Representation Learning.** The mainstream of graph representation learning on graphs diverges into two main topics: traditional network embedding and GNNs. Traditional network embedding cares more about the distribution of positive node pairs. Inspired by the skip-gram model [24], DeepWalk [27] learns embeddings via sampling "context" nodes for each vertex with random walks, and maximize the log-likelihood of observed context nodes for the given vertex. LINE [32] and node2vec [11] extend DeepWalk with various positive distributions. GNNs are deep learning based methods that generalize convolution operation to graph data. [17] design GCNs by approximating localized 1-order spectral convolution. For scalability, GraphSAGE [13] employs neighbor sampling to alleviate the receptive field expansion. FastGCN [5] adopts importance sampling in each layer.

**Negative Sampling.** Negative sampling is firstly proposed to speed up skip-gram training in word2vec [24]. Word embedding models sample negative samples according to its word frequency distribution proportional to the 3/4 power. Most later works on network embedding [11, 32] follow this setting. In addition, negative sampling has been studied extensively in recommendation. Bayesian Personalized Ranking [29] proposes uniform sampling for negative samples. Then, dynamic negative sampling (DNS) [41] is

proposed to sample informative negative samples based on current prediction scores. Besides, the PinSAGE [40] adds "hard negative items" to obtain fine enough "resolution" for recommender system. The negative samples are sampled uniformly with the rejection in WARP [38]. Recently, GAN-based negative sampling method has also been adopted to train better node representations for information retrieval [35], recommendation [36], word embeddings [2], knowledge graph embeddings [3]. Another lines of research focus on exploiting sampled softmax [1] or its debiasing variant [46] for extremely large scale link prediction tasks.

## 7 CONCLUSION

In this paper, we study the effect of negative sampling, a practical approach adopted in the literature of graph representation learning. Different from the existing works that decide a proper distribution for negative sampling in heuristics, we theoretically analyze the objective and risk of the negative sampling approach and conclude that the negative sampling distribution should be positively but sub-linearly correlated to their positive sampling distribution. Motivated by the theoretical results, we propose MCNS, approximating the ideal distribution by self-contrast and accelerating sampling by Metropolis-Hastings. Extensive experiments show that MCNS outperforms 8 negative sampling strategies, regardless of the underlying graph representation learning methods.

## ACKNOWLEDGEMENTS

The work is supported by the NSFC for Distinguished Young Scholar (61825602), NSFC (61836013), and a research fund supported by Alibaba Group.

## REFERENCES

[1] Yoshua Bengio and Jean-Sébastien Senécal. 2008. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks* 19, 4 (2008), 713–722.
[2] Avishek Joey Bose, Huan Ling, and Yanshuai Cao. 2018. Adversarial Contrastive Estimation. (2018), 1021–1032.
[3] Liwei Cai and William Yang Wang. 2018. KBGAN: Adversarial Learning for Knowledge Graph Embeddings. In *NAACL-HLT'18*. 1470–1480.
[4] Hugo Caselles-Dupré, Florian Lesaint, and Jimena Royo-Letelier. 2018. Word2vec applied to recommendation: Hyperparameters matter. In *RecSys'18*. ACM, 352–356.
[5] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. *ICLR'18* (2018).
[6] Siddhartha Chib and Edward Greenberg. 1995. Understanding the metropolis-hastings algorithm. *The american statistician* 49, 4 (1995), 327–335.
[7] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 39–46.
[8] Ming Ding, Jie Tang, and Jie Zhang. 2018. Semi-supervised learning on graphs with generative adversarial nets. In *CIKM'18*. ACM, 913–922.
[9] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of machine learning research* 9, Aug (2008), 1871–1874.
[10] Hongchang Gao and Heng Huang. 2018. Self-Paced Network Embedding. (2018), 1406–1415.
[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD'16*. ACM, 855–864.
[12] Michael U Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research* 13, Feb (2012), 307–361.
[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS'17*. 1024–1034.
[14] Henry Hsu and Peter A Lachenbruch. 2007. Paired t test. *Wiley encyclopedia of clinical trials* (2007), 1–3.
[15] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *ICDM'08*. Ieee, 263–272.
[16] Hong Huang, Jie Tang, Sen Wu, Lu Liu, and Xiaoming Fu. 2014. Mining triadic closure patterns in social networks. In *WWW'14*. 499–504.
[17] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *ICLR'17* (2017).
[18] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *TKDD'07* 1, 1 (2007), 2–es.
[19] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS'14*. 2177–2185.
[20] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI'18*.
[21] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.
[22] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR'15*. ACM, 43–52.
[23] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics* 21, 6 (1953), 1087–1092.
[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS'13*. 3111–3119.
[25] Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *NIPS'13*. 2265–2273.
[26] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *ICDM'08*. IEEE, 502–511.
[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD'14*. ACM, 701–710.
[28] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM'18*. ACM, 459–467.
[29] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI'09*. AUAI Press, 452–461.
[30] Kazunari Sugiyama and Min-Yen Kan. 2010. Scholarly paper recommendation via user's recent research interests. In *JCDL'10*. ACM, 29–38.
[31] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197* (2019).
[32] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW'15*. 1067–1077.
[33] Cunchao Tu, Han Liu, Zhiyuan Liu, and Maosong Sun. 2017. Cane: Context-aware network embedding for relation modeling. In *ACL'17*. 1722–1731.
[34] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR'18* (2018).
[35] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR'17*. ACM, 515–524.
[36] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. 2018. Neural memory streaming recommender networks with adversarial training. In *KDD'18*. ACM, 2467–2475.
[37] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community preserving network embedding. In *AAAI'17*.
[38] Jason Weston, Samy Bengio, and Nicolas Usunier. 2011. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI'11*.
[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).
[40] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD'18*. ACM, 974–983.
[41] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing Top-N Collaborative Filtering via Dynamic Negative Item Sampling. In *SIGIR'13*. ACM, 785–788.
[42] Yongqi Zhang, Quanming Yao, Yingxia Shao, and Lei Chen. 2019. NSCaching: Simple and Efficient Negative Sampling for Knowledge Graph Embedding. (2019), 614–625.
[43] Zheng Zhang and Pierre Zweigenbaum. 2018. GNEG: Graph-Based Negative Sampling for word2vec. In *ACL'18*. 566–571.
[44] Tong Zhao, Julian McAuley, and Irwin King. 2015. Improving latent factor models via personalized feature projection for one class recommendation. In *CIKM'15*. ACM, 821–830.
[45] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable graph embedding for asymmetric proximity. In *AAAI'17*.
[46] Chang Zhou, Jianxin Ma, Jianwei Zhang, Jingren Zhou, and Hongxia Yang. 2020. Contrastive Learning for Debiased Candidate Generation in Large-Scale Recommender Systems. arXiv:cs.IR/2005.12964

# A APPENDIX

In the appendix, we first report the implementation notes of MCNS, including the running environment and implementation details. Then, the encoder algorithms are presented. Next, the detailed description of each baseline is provided. Finally, we present datasets, evaluation metrics and the DFS algorithm in detail.

## A.1 Implementation Notes

**Running Environment.** The experiments are conducted on a single Linux server with 14 Intel(R) Xeon(R) CPU E5-268 v4 @ 2.40GHz, 256G RAM and 8 NVIDIA GeForce RTX 2080TI-11GB. Our proposed MCNS is implemented in Tensorflow 1.14 and Python 3.7.

**Implementation Details.** All algorithms can be divided into three parts: *negative sampling*, *positive sampling* and *embedding learning*. In this paper, we focus on negative sampling strategy. A negative sampler selects a negative sample for each positive pair and feeds it with a corresponding positive pair to the encoder to learn node embeddings. With the exception of PinSAGE, the ratio of positive to negative pairs is set to $1:1$ for all negative sampling strategies. PinSAGE uniformly selects 20 negative samples per batch and adds 5 "hard negative samples" per positive pair. Nodes ranked at top-100 according to PageRank scores are randomly sampled as "hard negative samples". The candidate size $S$ in DNS is set to 5 for all datasets. The value of $T$ in KBGAN is set to 200 for Amazon and Alibaba datasets, and 100 for other datasets.

For recommendation task, we evenly divide the edges in each dataset into 11 parts, 10 for ten-fold cross validation and the extra one as a validation set to determine hyperparameters. For recommendation datasets, the node types contain $U$ and $I$ representing *user* and *item* respectively. Thus, the paths for information aggregation are set to $U-I-U$ and $I-U-I$ for GraphSAGE and GCN encoders. All the important parameters in MCNS are determined by a rough grid search. For example, the learning rate is the best among $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$. Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$ serves as the optimizer for all the experiments. The values of hyperparameters in different datasets are listed as follows:

| | MovieLens | Amazon | Alibaba |
|---|---|---|---|
| learning rate | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ |
| embedding dim | 256 | 512 | 512 |
| margin $\gamma$ | 0.1 | 0.1 | 0.1 |
| batch size | 256 | 512 | 512 |
| $M$ in Hits@k | ALL | 500 | 500 |

For link prediction task, we employ five-fold cross validation to train all methods. The embedding dimension is set to 256, and the margin $\gamma$ also be set to 0.1 for all strategies. The model parameters are updated and optimized by stochastic gradient descent with Adam optimizer.

In multi-label classification task, we first employ all the edges to learn node embeddings. Then, these embeddings serve as features to train the classifier. Embedding dimension is set to 128 and the margin $\gamma$ is set as 0.1 on the BlogCatalog dataset for all negative sampling strategies. The optimizer also adopts Adam updating rule.

## A.2 Encoders

We perform experiments on three algorithms as follows.

- **DeepWalk** [27] is the most representative graph representation algorithm. According to the Skip-gram model [24], DeepWalk samples nodes co-occurring in the same window in random-walk paths as positive pairs, which acts as a data augmentation. This strategy serves as "sampling from $\hat{p_d}$" in the view of SampledNCE (§2.2).
- **GCN** [17] introduces a simple and effective neighbor aggregation operation for graph representation learning. Although GCN samples positive pairs from direct edges in the graph, the final embeddings of nodes are implicitly constrained by the smoothing effect of graph convolution. However, due to its poor scalability and high memory consumption, GCN cannot handle large graphs.
- **GraphSAGE** [13] is an inductive framework of graph convolution. A fixed number of neighbors are sampled to make the model suitable for batch training, bringing about its prevalence in large-scale graphs. The **mean**-aggregator is used in our experiments.

## A.3 Baselines

We give detailed negative sampling strategies as follows. For each encoder, the hyperparameters for various negative sampling strategies remain the same.

- **Power of Degree** [24]. This strategy is first proposed in word2vec, in which $p_n(v) \propto deg(v)^{\beta}$. In word embedding, the best $\alpha$ is found to be 0.75 by experiments and followed by most graph embedding algorithms [27, 32]. We use $Deg^{0.75}$ to denote this widely-used setting.
- **RNS** [4]. RNS means sampling negative nodes at uniform random. Previous work [4] mentions that in recommendation, RNS performs better and more robust than $Deg^{0.75}$ during tuning hyperparameters. This phenomenon is also verified in our recommendation experiments by using this setting as a baseline.
- **WRMF** [15, 26]. Weighted Regularized Matrix Factorization is an implicit MF model, which adapts a weight to reduce the impact of unobserved interactions and utilizes an alternating-least-squares optimization process. To solve the high computational costs, WRMF proposes three negative sampling schemes, including Uniform Random Sampling (RNS), Item-Oriented Sampling (ItemPop), User-Oriented Sampling (Unconsider in our paper).
- **DNS** [41]. Dynamic Negative Sampling (DNS) is originally designed for sampling negative items to improve pair-wise collaborative filtering. For each user $u$, DNS samples negative $S$ items $\{v_0, ..., v_{S-1}\}$ and only retains the one with the highest scoring function $s(u, v)$. DNS can be applied to graph-based recommendation by replacing $s(u, v)$ in BPR [29] with the inner product of embedding $\vec{u}\vec{v}^{\top}$.
- **PinSAGE** [40]. In PinSAGE, a technique called "hard negative items" is introduced to improve its performance. The hard negative items refer to the high ranking items according to users' Personalized PageRank scores. Finally, negative items are sampled from top-K hard negative items.
- **WARP** [38, 44]. Weighted Approximate-Rank Pairwise(WARP) utilizes SGD and a novel sampling trick to approximate ranks.

For each positive pair $(u, v)$, WARP proposes uniform sampling with rejection strategy to find a negative item $v'$ until $\vec{u}\vec{v}'^{\top} - \vec{u}\vec{v}^{\top} + \gamma > 0$.

- **IRGAN** [35]. IRGAN is a GAN-based IR model that unify generative and discriminative information retrieval models. The generator generates adversarial negative samples to fool the discriminator while the discriminator tries to distinguish them from true positive samples.
- **KBGAN** [3]. KBGAN proposed a adversarial sampler to improve the performances of knowledge graph embedding models. To reduce computational complexity, KBGAN uniformly randomly selects $T$ nodes to calculate the probability distribution for generating negative samples. Inspired by it, NMRN [36] employed a GAN-based adversarial training framework in streaming recommender model.

## A.4 Datasets

The detailed descriptions for five datasets are listed as follows.

- **MovieLens**[3] is a widely-used movie rating dataset for evaluating recommender systems. We choose the MovieLens-100k version that contains 100,000 interaction records generated by 943 users on 1682 movies.
- **Amazon**[4] is a large E-commerce dataset introduced in [22], which contains purchase records and review texts whose time stamps span from May 1996 to July 2014. In the experiments, we take the data from the commonly-used "electronics" category to establish a user-item graph.
- **Alibaba**[5] is constructed based on the data from another large E-commerce platform, which contains user's purchase records and items' attribute information. The data are organized as a user-item graph for recommendation.
- **Arxiv GR-QC** [18] is a collaboration network generated from the e-print arXiv where nodes represent authors and edges indicate collaborations between authors.
- **BlogCatalog** is a network of social relationships provided by blogger authors. The labels represent the topic categories provided by the authors and each blogger may be associated to multiple categories. There are overall 39 different categories for BlogCatalog.

## A.5 Evaluation Metrics

**Recommendation.** To evaluate the performance of MCNS for recommendation task, $Hits@k$ [7] and mean reciprocal ranking ($MRR$) [30] serve as evaluation methodologies.

- **Hits@k** is introduced by the classic work [30] to measure the performance of personalized recommendation at a coarse granularity. Specifically, we can compute $Hits@k$ of a recommendation system as follows:
  (1) For each user-item pair $(u, v)$ in the test set $D_{test}$, we randomly select $M$ additional items $\{v'_0, ..., v'_{M-1}\}$ which are never showed to the queried user $u$.
  (2) Form a ranked list of $\{\vec{u}\vec{v}^{\top}, \vec{u}\vec{v}'^{\top}_0, ..., \vec{u}\vec{v}'^{\top}_{M-1}\}$ in descending order.
  (3) Examine whether the rank of $\vec{u}\vec{v}^{\top}$ is less than $k$ (hit) or not (miss).
  (4) Calculate $Hits@k$ according to $Hits@k = \frac{\#hit@k}{|D_{test}|}$.
- **MRR** is another evaluation metric for recommendation task, which measures the performance at a finer granularity than $Hits@k$. $MRR$ assigns different scores to different ranks of item $v$ in the ranked list mentioned above when querying user $u$, which is defined as follows:

$$MRR = \frac{1}{|D_{test}|} \sum_{(u_i, v_i) \in D_{test}} \frac{1}{rank(v_i)}.$$

In our experiment, we set $k$ as $\{10, 30\}$, and $M$ is set to 500 for the Alibaba and Amazon datasets. For the MovieLens dataset, we use all unconnected items for each user as $M$ items to evaluate the hit rate.

**Link Prediction.** The standard evaluation metric AUC is widely applied to evaluate the performance on link prediction task [11, 33, 45]. Here, we utilize *roc_auc_score* function from scikit-learn to calculate prediction score. The specific calculation process is demonstrated as follows.

(1) We extract 30% of true edges and remove them from the whole graph while ensuring the residual graph is connected.
(2) We sample 30 % false edges, and then combine 30% of true edges and false edges into a test set.
(3) Each graph embedding algorithm with various negative sampling strategies is trained using the residual graph, and then the embedding for each node is obtained.
(4) We predict the probability of a node pair being a true edge according to the inner product. We finally calculate AUC score according to the probability via *roc_auc_score* function.

**Multi-label Classification.** In the multi-label classification, we adopt Micro-F1 and Macro-F1 as evaluation metrics, which are widely used in many previous works [11, 24]. In our experiments, the F1 score is calculated by scikit-learn. In detail, we randomly select a percentage of labeled nodes, $T_R \in \{10\%, 50\%, 90\%\}$ to learn node embeddings. Then the learned embeddings is used as features to train the classifier and predict the labels of the remaining nodes. For fairness in comparison, we used the same training set to train the classifier for all negative sampling strategies. For each training ratio, we repeat the process 10 times and report the average scores.

## A.6 DFS

Here we introduce a DFS strategy to generate traversal sequence $L$ where adjacent nodes in the sequence is also adjacent in the graph.

**Algorithm 3:** DFS

**Input:** Current node $x$.
Global variables: graph $G = (V, E)$, sequence $L$.
Append $x$ to $L$
**for** *each unvisited neighbor* $y$ *of* $x$ *in* $G$ **do**
  DFS($y$)
  Append $x$ to $L$
**end**

[3] https://grouplens.org/datasets/movielens/100k/
[4] http://jmcauley.ucsd.edu/data/amazon/links.html
[5] we use Alibaba dataset to represent the real data we collected.