LONE SAMPLER : Graph Node Embeddings by Coordinated Local Neighborhood Sampling

Konstantin Kutzkov

Teva Pharmaceuticals
kutzkov@gmail.com

Abstract

Local graph neighborhood sampling is a fundamental computational problem that is at the heart of algorithms for node representation learning. Several works have presented algorithms for learning *discrete* node embeddings where graph nodes are represented by discrete features such as attributes of neighborhood nodes. Discrete embeddings offer several advantages compared to continuous word2vec-like node embeddings: ease of computation, scalability, and interpretability. We present LONE SAMPLER , a suite of algorithms for generating discrete node embeddings by Local Neighborhood Sampling, and address two shortcomings of previous work. First, our algorithms have rigorously understood theoretical properties. Second, we show how to generate approximate explicit vector maps that avoid the expensive computation of a Gram matrix for the training of a kernel model. Experiments on benchmark datasets confirm the theoretical findings and demonstrate the advantages of the proposed methods.

Introduction

Graphs are ubiquitous representation for structured data. They model naturally occurring relations between objects and, in a sense, generalize sequential data to more complex dependencies. Many algorithms originally designed for learning from sequential data are thus generalized to learning from graphs. Learning continuous vector representations of graph nodes, or *node embeddings*, have become an integral part of the graph learning toolbox, with applications ranging from link prediction (Grover and Leskovec 2016) to graph compression (Ahmed et al. 2017). The first algorithm (Perozzi, Al-Rfou, and Skiena 2014) for learning node embeddings generates random walks, starting from each node in the graph, and then feeds the sequences of visited nodes into a word embedding learning algorithm such as word2vec (Mikolov et al. 2013). The approach was extended to a more general setting where random walks can consider different properties of the local neighborhood (Grover and Leskovec 2016; Tang et al. 2015; Tsitsulin et al. 2018). An alternative method for training continuous node embeddings is based on matrix factorization of (powers of) the graph adjacency matrix.

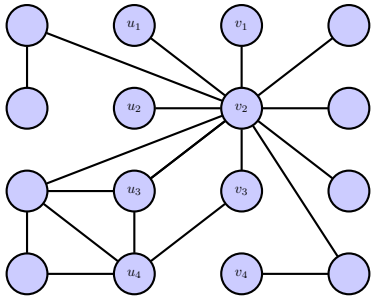

Figure 1: Is u_2 or u_4 more similar to u_3?

As an alternative, researchers proposed to use *coordinated* node sampling for training *discrete* node embeddings (Wu et al. 2018; Yang et al. 2019). In this setting, each sample is an independent estimator of the similarity between nodes. (There are different notions of node similarity but most reflect how easy it is to reach one node from another.) Thus, sampled nodes themselves can be coordinates of the embedding vectors. We can then compare two node embeddings by their Hamming distance. There are several advantages of discrete embeddings over continuous embeddings. First, we avoid the need to train a (possibly slow) word2vec-like model. Second, the samples are the original graph nodes and contain all meta-information provided in the original input, be it personal data of users of a social network or the weather conditions at railway stations. By sampling, all this information is preserved and this can lead to the design of interpretable algorithms. And finally, the algorithms are truly local and can deal with massive or distributed graphs if only access to the local neighborhood of each node is possible.

The ultimate goal for node embeddings is to represent the structural roles of nodes by fixed size vectors which reflect the similarity between nodes. However, there are different notions of node similarity. As an example, consider the toy graph in Figure 1. It seems that u_4 is similar to u_3 because they are part of the same local clique. But on the the other hand one can argue that u_1 is more similar to u_3 than u_4 because they are all directly connected to a hub node like v_2. The goal of the present work is to design scalable embedding algorithms that can handle different similarity objectives and have rigorously understood properties.

The main contributions of the paper are as follows:

- **Theoretical insights.** We formally define the problem of coordinated local node sampling and present novel discrete embedding algorithms that, unlike previous works based on heuristics, *provably* preserve the similarity between nodes with respect to different objectives. Furthermore, the algorithms are highly efficient.
- **Scalable model training.** We show how to use the discrete embeddings in scalable kernel models. More precisely, we design methods that approximate the Hamming kernel by explicit feature maps such that we can train a linear support-vector machine for node classification. Previous works require the computation of a Gram matrix which is unfeasible for massive graphs.

Organization of the Paper In the next section we present notation and outline the problem setting. In Section we present three algorithms for local neighborhood sampling according to different objectives. We analyze the computational complexity of each approach and the properties of the generated samples. In Section we present an approach to the generation of explicit feature maps for the Hamming kernel, thus enabling the use of discrete node embeddings in scalable kernel models. We discuss related work in Section . An experimental evaluation is presented in Section . The paper is concluded in Section .

Notation and Overview of Techniques

The input is a graph $G = (V, E)$ over $n = |V|$ nodes and $m = |E|$ edges. The distance $d(u, v)$ between nodes u and v is the minimum number of edges that need to be traversed in order to reach u from v, i.e., the shortest path from u to v. We consider undirected graphs, thus $d(u, v) = d(v, u)$. Also, we assume connected graphs, thus $d(u, v) < \infty$ for all $u, v \in V$. These assumptions are however only for the ease of presentation, all algorithms work for directed or disconnected graphs. The k-hop neighbors of node u is the set of nodes $N_k(u) = \{v \in V : d(u, v) \leq k\}$. The set of neighbors of node u is denoted as $N(u)$. We call the subgraph induced by $N_k(u)$ the local k-hop neighborhood of u. A *discrete embedding vector* is a fixed size vector whose entries come from a discrete set. For example, by sampling with replacements ℓ nodes from the k-hop neighborhood of each node we create embeddings that consists of other nodes. Or attributes of the neighboring nodes, if existent.

We say $\tilde{q}$ is an $1 \pm \varepsilon$-approximation of a quantity q if $(1 - \varepsilon)q \leq \tilde{q} \leq (1 + \varepsilon)q$.

Sketch-Based Coordinated Sampling Our algorithms build upon sketching techniques for massive data summarization. In a nutshell, sketching replaces a vector $\mathbf{x} \in \mathbb{R}^n$ by a compact data structure $sketch_{\mathbf{x}} \in \mathbb{R}^d$, $d \ll n$, that approximately preserves many properties of the original $\mathbf{x}$. In coordinated sampling (Cohen 2016), given a universe of elements U, and a set of sets $\{S_i \subseteq U\}$, the goal is to draw samples from U such that each set S_i is represented by a compact summary $sketch_{S_i}$. It should hold $\texttt{sim}(sketch_{S_i}, sketch_{S_j}) \approx \texttt{sim}(S_i, S_j)$, i.e., the summaries approximately preserve the similarity between the original sets, for different similarity measures.

Algorithm 1: Coordinated local neighborhood sampling.

Input: Graph $G = (V, E)$
  for each $u \in V$ **do**
    Initialize $sketch_u$ with node u
  for $i = 1$ to k **do**
    for each $u \in V$ **do**
      Update $sketch_u$ by merging all $sketch_v$ for $v \in N(u)$ into $sketch_u$
  for $u \in V$ **do**
    Return a node from $sketch_u$ as a sample for u.

L_p **Sampling** The p-norm of vector $x \in \mathbb{R}^n$ is $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$ for $p \in \mathbb{N} \cup \{0\}$. We call L_p sampling a sampling procedure that returns each coordinate x_i from x with probability $\frac{|x_i|^p}{\|x\|_p^p}$.

Work Objective Let $\mathbf{f}_u^k$ be the k-hop frequency vectors of node u such that $\mathbf{f}_u^k[z]$ is the number of unique paths of length at most k from u to z. Let $s_u \in N_k(u)$ be the node returned by an algorithm $\mathcal{A}$ as a sample for node u. We say that $\mathcal{A}$ is a *coordinated sampling algorithm* with respect to a similarity measure $\texttt{sim} : V \times V \to [0, 1]$ iff

$$\Pr[s_u = s_v] = \texttt{sim}(\mathbf{f}_u^k, \mathbf{f}_v^k) \text{ for } u, v \in V$$

The goal of our work is the design of scalable algorithms for coordinated L_p sampling from local node neighborhoods with rigorously understood properties. We can also phrase the problem in graph algebra terms. Let $A \in \{0, 1\}^{n \times n}$ be the adjacency matrix of the graph. The objective is to implement coordinated L_p sampling from each row of $M_k = \sum_{i=0}^k A^i$ *without* explicitly generating the A^i. By sketching the local k-hop neighborhood frequency vector $\mathbf{f}_u^k$ of each node u we will design efficient L_p sampling algorithms.

LoNE Sampler

The general form of our approach is in Figure 1. We first initialize a sketch at each node u with the node u itself. For k iterations, for each node we collect the sketches from its neighbors and aggregate them into a single sketch. At the end we sample from each node's sketch. The algorithm is used to generate a single coordinate of the embedding vector of each node. For each coordinate we will use a different random seed.

Consider a trivial example. We initialize $sketch_u$ with a sparse n-dimensional binary vector such that $sketch_u[u] = 1$ is the only nonzero coordinate for all $u \in V$. Merging the sketches is simply entrywise vector addition. We can formally show that after k iterations $sketch_u$ is exactly the k-hop frequency vector of u, i.e., $sketch_u^{(k)}[v] = \mathbf{f}_u^k[v]$. We need to address the following issues: i) Storing the entire frequency vectors $\mathbf{f}_u^k$ is very inefficient. Even for smaller values of k, we are likely to end up with dense n-dimensional sketches as most real graphs have a small diameter. ii) How can we get coordinated samples from the different sketches?

Coordinated Uniform (L_0) Sampling

We first present a simple coordinated sampling algorithm for generating samples from the local neighborhood of each node. The approach builds upon *min-wise independent permutations* (Broder et al. 2000), a technique for estimating the Jaccard similarity between sets. Assume we are given two sets $A \subseteq U$ and $B \subseteq U$, where U is a universe of elements, for example all integers. We want to estimate the fraction $\frac{|A \cap B|}{|A \cup B|}$. Let $\pi : U \to U$ be a random permutation of the elements in U. With probability $\frac{|A \cap B|}{|A \cup B|}$ the smallest element in $A \cup B$ with respect to the total order defined by π is contained in $A \cap B$. The indicator variable denoting whether the smallest elements in $\pi(A)$ and $\pi(B)$ are identical is an unbiased estimator of the Jaccard similarity $J(A, B)$. The mean of $t = O(\frac{1}{\alpha \varepsilon^2})$ such estimators is an $1 \pm \varepsilon$-approximation of $J(A, B) \geq \alpha$ with high probability.

An algorithm for sampling uniformly from the k-hop neighborhood of node u easily follows, see Algorithm 2. We implement a random permutation on the n nodes by generating a random number for each node $r : V \to \{0, 1, \ldots, \ell - 1\}$. For a sufficiently large ℓ with high probability r is a bijective function and thus it implements a random permutation[1]. For each node u, $sketch_u$ is initialized with $(r(u), u)$, i.e., the sketch is just a single *(random number, node)* pair. The aggregation after each iteration is storing the pair with the smallest random number from u's neighbors, $sketch_u = \min_{(r_v, v): v \in N(u)} sketch_v$. After k iterations at each node u we have the smallest number from the set $\{r(v) : v \in N_k(u)\}$, i.e., we have sampled a node from $N_k(u)$ according to the permutation defined by the function r. The samples for any two nodes u and w are coordinated as we work with the same permutation on the set $N_k(u) \cup N_k(w) \subseteq V$. The next theorem is a straightforward corollary from the main result on minwise-independent permutations (Broder et al. 2000):

Theorem 1 *For all nodes $u \in V$, we can sample $s_u \in N_k(u)$ with probability $1/|N_k(u)|$ in time $O(mk)$ and space $O(m)$. For any pair of nodes u, v it holds*

$$\Pr[s_u = s_v] = \frac{|N_k(u) \cap N_k(v)|}{|N_k(u) \cup N_k(v)|}$$

In terms of linear algebra, the above algorithm is an efficient implementation of the following naïve approach: Let A be the the adjacency matrix of G. Randomly permute the columns of $M_k = \sum_{i=0}^{k} A^i$, and for each row in the updated M_k select the first nonzero coordinate. Thus, the algorithm implements coordinated L_0 sampling from each row of M_k but avoids the explicit generation of M_k.

Coordinated L_p Sampling

The solution for uniform sampling is simple and elegant but it does not fully consider the graph structure. It only considers if there is a path between two nodes u and v but, unlike in random walks, not how many paths there are between u and v. We design coordinated sampling algorithms such that easily accessible nodes are more likely to be sampled.

[1]For $\ell = n^2/\delta$ with probability $1 - \delta$ the function is bijective.

Algorithm 2: L_0 and L_1 sampling for single coordinate.

L_0 SAMPLING

Input: Graph $G = (V, E)$, random function $r : V \to (0, 1]$
  for each $u \in V$ **do**
    Initialize $sketch_u$ with $(r(u), u)$
  for $i = 1$ to k **do**
    for each $u \in V$ **do**
      for each $v \in N(v)$ **do**
        $sketch_u = \min(sketch_u, sketch_v)$
  for $u \in V$ **do**
    Return the node z from the pair $sketch_u = (r(z), z)$

L_1 SAMPLING

Input: Graph $G = (V, E)$, random function $r : V \to (0, 1]$, int ℓ
  for each $u \in V$ **do**
    Initialize $sketch_u$ with node, weight pair $(u, w(u) = 1/r(u))$
  for $i = 1$ to k **do**
    for each $u \in V$ **do**
      for each $v \in N(v)$ **do**
        Update $sketch_u$ with $sketch_v$, keeping the top ℓ (node, weight) pairs.
  for $u \in V$ **do**
    Return node z from $sketch_u$ if $w(z) \geq \|\mathbf{f^k_u}\|_1$

Let us present an approach to L_p sampling from data streams for $p \in (0, 2]$ (Jowhari, Saglam, and Tardos 2011). Let $\mathcal{S}$ be a data stream of pairs i, w_i where i is the item and $w_i \in \mathbb{R}$ is the weight update for item i, for $i \in [n]$. For example, the network traffic for a website where user i spends w_i seconds at the site. The objective is L_p sampling from the frequency vector of the stream, i.e., return each item i with probability roughly $|\mathbf{f}[i]|^p / \|\mathbf{f}\|_p^p$, where $\mathbf{f}[i] = \sum_{(i, w_i) \in \mathcal{S}} w_i$. (Items can occur multiple times in the stream.) The problem has a simple solution if we can afford to store the entire frequency vector $\mathbf{f}$. The solution in (Jowhari, Saglam, and Tardos 2011) is to reweight each item by scaling it by a random number $1/r_i^{1/p}$ for a uniform random $r_i \in (0, 1]$. Let $z_i = \mathbf{f}[i]/r_i^{1/p}$ be the new weight of item i. The crucial observation is that

$$\Pr[z_i \geq \|\mathbf{f}\|_p] = \Pr[r_i \leq \mathbf{f}[i]^p / \|\mathbf{f}\|_p^p] = \frac{\mathbf{f}[i]^p}{\|\mathbf{f}\|_p^p}$$

One can show that with constant probability there exists a unique item i with $z_i \geq \|\mathbf{f}\|_p$. Thus, if we know the value of $\|\mathbf{f}\|_p$, a space-efficient solution is to keep a sketch data structure from which we can detect the heavy hitter that will be the sampled item. Estimating the p-norm of a frequency vector for $p \in (0, 2]$ is a fundamental problem with known solutions (Charikar, Chen, and Farach-Colton 2004).

There are two challenges we need to address when applying the above approach to local graph neighborhood sampling. First, how do we get *coordinated* samples? Second, if we explicitly generate all entries in the frequency vector of the local neighborhood this would result in time complexity of $O(\sum_{i=0}^{k} nnz(A^i))$, $nnz(A)$ denoting the number of non-zero elements in the adjacency matrix A.

For the first issue, we achieve coordinated sampling by reweighting the nodes $u \in V$ by $1/r_u$ for random numbers

$r_u \in (0,1]$ for $u \in V$. An inefficient solution works then as follows. For each node u we generate a random number $r_u \in (0,1]$, initialize $\mathbf{w}_u^0[u] = 1/r_u$ and set $\mathbf{w}_u^0[v] = 0$ for all $v \neq u$. We iterate over the neighbor nodes and update the vector $\mathbf{w}_u^k = \mathbf{w}_u^{k-1} \oplus \sum_{v \in N(u)} \mathbf{w}_v^{k-1}$ where $\oplus$ denotes entrywise vector addition. We sample a node v iff $\mathbf{w}_u[v] \geq \|\mathbf{f}_u^k\|_1$. (Observe that $\|\mathbf{f}_u^k\|_1$ can be computed exactly by the same iterative procedure.)

Our L_1 sampling solution in Algorithm 2 is based on the idea of *mergeable sketches* (Agarwal et al. 2013) such for any vectors $x, y \in \mathbb{R}^n$ it holds $sketch(x+y) = sketch(x) + sketch(y)$. Following the LONE SAMPLER template from Algorithm 1, we iteratively update the sketches at each node. The sketch collected in the i-th iteration at node u results from merging the sketches in $N(u)$ at iteration $i-1$, and summarizes the i-hop neighborhood $N_i(u)$. (Jowhari, Saglam, and Tardos 2011) use the CountSketch data structure (Charikar, Chen, and Farach-Colton 2004). But a CountSketch is just an array of counters from which the frequency of an item can be estimated. This wouldn't allow us to efficiently retrieve the heavy hitter from the sketch as we would need to query the sketch for all k-hop neighborsof each node. (In the setting in (Jowhari, Saglam, and Tardos 2011) a single vector is being updated in a streaming fashion. After preprocessing the stream we can afford to query the sketch for each vector index as this wouldn't increase the asymptotic complexity of the algorithm.)

Here comes the main algorithmic novelty of our approach. Observing that in our setting all weight updates are strictly positive, we design a solution by using another kind of summarization algorithms for frequent items mining, the so called *counter based algorithms* (Karp, Shenker, and Papadimitriou 2003). In this family of algorithms, the sketch consists of an explicitly maintained list of frequent items candidates. For a sufficiently large yet compact sketch a heavy hitter is guaranteed to be in the sketch, thus the sampled node will be the heaviest node in the sketch. The algorithm is very efficient as we only need to merge the compact sketches at each node in each iteration.

The above described algorithm can be phrased again in terms of the adjacency matrix A. Let $M_k = \sum_{i=0}^{k} A^i \cdot R^{-1}$ where $R \in \mathbb{R}^{n \times n}$ is a diagonal matrix with diagonal entries randomly selected from $(0,1]$. Then from the i-th row we return as sample the index j with the largest value $M_{i,j}$ for which the sampling condition $M_{i,j} \geq \|A_{i,:}^{(k)}\|_1$ is satisfied. Sampling in this way is coordinated because the j-th column of $A^{(k)}$ is multiplied by the same random value $1/r_j$, thus a node j with a small r_j is more likely to be sampled.

The formal proof for the next theorem is quite technical and provided in the appendix but at a high level we show that with high probability we can efficiently detect a node that satisfies the sampling condition using a sketch with $O(\log n)$ entries at each node.

Theorem 2 *Let G be a graph over n nodes and m edges, and let $\mathbf{f}_u^k$ be the frequency vector of the k-hop neighborhood of node $u \in V$. For all $u \in V$, we can sample a node $s_u \in N_k(u)$ with probability $\frac{\mathbf{f}_u^k[s_u]}{\|\mathbf{f}_u^k\|_1}$ in time $O(mk \log n)$ and space $O(m + n \log n)$. For each pair of nodes $u, v \in V$*

$$\Pr[s_u = s_v] = \sum_{x \in V} \min\left(\frac{\mathbf{f}_u^k[x]}{\|\mathbf{f}_u^k\|_1}, \frac{\mathbf{f}_v^k[x]}{\|\mathbf{f}_v^k\|_1}\right)$$

For L_2 sampling we can apply the same approach as for L_1 sampling. The only difference is that we need to reweight each node u by $1/r_u^{1/2}$ for $r_u \in (0,1]$ and compare the weight of the sample candidate with $\|\mathbf{f}_u^k\|_2$. However, unlike $\|\mathbf{f}_u^k\|_1$, we cannot compute exactly $\|\mathbf{f}_u^k\|_2$ during the k iterations. But we can efficiently *approximate* the 2-norm of a vector revealed in a streaming fashion, this is a fundamental computational problem for which algorithms with optimal complexity have been designed (Charikar, Chen, and Farach-Colton 2004). We show the following result:

Theorem 3 *Let G be a graph over n nodes and m edges, and let $\mathbf{f}_u^k$ be the frequency vector of the k-hop neighborhood of node $u \in V$. For all $u \in V$, we can sample a node $s_u \in N_k(u)$ with probability $(1 \pm \varepsilon)\frac{\mathbf{f}_u^k[s_u]^2}{\|\mathbf{f}_u^k\|_2^2}$ in time $O(mk(1/\varepsilon^2 + \log n))$ and space $O(m + n(1/\varepsilon^2 + \log n))$, for a user-defined $\varepsilon \in (0,1)$. For each pair of nodes $u, v \in V$*

$$\Pr[s_u = s_v] = (1 \pm \varepsilon) \sum_{x \in V} \min\left(\frac{\mathbf{f}_u^k[x]^2}{\|\mathbf{f}_u^k\|_2^2}, \frac{\mathbf{f}_v^k[x]^2}{\|\mathbf{f}_v^k\|_2^2}\right)$$

The similarity measures preserved in the above two theorems may appear non-intuitive. In the appendix we discuss that in a sense they approximate versions of the well-known *cosine similarity*.

An Explicit Map for the Hamming Kernel

A typical application for node embeddings is node classification. Previous works (Wu et al. 2018; Yang et al. 2019) have proposed to train a kernel machine with a Hamming kernel. The Hamming kernel is defined as the overlap of vectors $x, y \in \mathbb{U}^d$ for a universe $\mathbb{U}$: $H(x,y) = \sum_{i=1}^{d} \mathbf{1}(\mathbf{x_i} = \mathbf{y_i})$, i.e. the number of positions at which x and y are identical. However, this approach requires the explicit generation of a precomputed Gram matrix with t^2 entries, where t is the number of training examples.

A solution would be to represent the embeddings by sparse explicit vectors and train a highly efficient linear SVM (Joachims 2006) model. For a universe size N, we can represent each vector by a binary Nd-dimensional vector b with exactly d nonzeros. But even a linear SVM would be unfeasible as we will likely end up with dense decision vectors with $O(Nd)$ entries. We can use TensorSketch (Pham and Pagh 2013), originally designed for the generation of explicit feature maps for the polynomial kernel. Mapping the discrete vectors to lower dimensional sketches with $O(1/\varepsilon^2)$ entries would preserve the Hamming kernel with an additive error of εd. Observing that d can be a rather small constant, we present a simple algorithm that is more space-efficient than TensorSketch for $d < 1/\varepsilon$. The algorithm hashes each nonzero coordinate of the explicit Nd-dimensional vector b to a vector $f(b)$ with d/ε coordinates. If there is a collision, i.e., $f(b)[i] = f(b)[j]$ for $i \neq j$, we set $f(b)[i] = 1$ without adding up $f(b)[i]$ and $f(b)[j]$ as in TensorSketch. We show the following result:

Theorem 4 *Let $\mathbb{U}$ be a discrete space and $x, y \in \mathbb{U}^d$ for $d \in \mathbb{N}$. For any $\varepsilon \in (0, 1]$ there is a mapping $f : \mathbb{U}^d \to \{0, 1\}^D$ such that $D = \lceil d/\varepsilon \rceil$ and $H(x, y) = f(x)^T f(u) \pm \varepsilon d$ with probability 2/3.*

Related Work

Random Walk Based Embeddings Pioneering approaches to learning word embeddings, such as word2vec (Mikolov et al. 2013) and GloVe (Pennington, Socher, and Manning 2014), have served as the basis for graph representation learning. The main idea is that for each graph node u, we learn how to predict u's occurrence from its context (Perozzi, Al-Rfou, and Skiena 2014). In natural language the context of each word is the set of surrounding words in a sequence, and for graph nodes the context is the set of local neighbors, thus random walks have been used to generate node sequences. Various algorithms have been proposed that allow some flexibility in selecting local neighbors according to different criteria (Tang et al. 2015; Grover and Leskovec 2016; Zhou et al. 2017; Tsitsulin et al. 2018).

Matrix Factorization A branch of node embeddings algorithms work by factorization of (powers of) the adjacency matrix of the graph (Ou et al. 2016; Zhang et al. 2018). These algorithms have well-understood properties but can be inefficient as even if the adjacency matrix is usually sparse, its powers can be dense, e.g., the average distance between any two users in the Facebook graph is only 4 (Backstrom et al. 2012). The computational complexity is improved using advanced techniques from linear algebra.

Deep Learning Node embeddings can be also learned using graph neural networks (Hamilton, Ying, and Leskovec 2017). GNNs are *inductive* and can be applied to previously unseen nodes, while the above discussed approaches, including ours, are *transductive* and work only for a fixed graph. This comes at the price of the typical deep learning disadvantages such as slow training and the need for careful hyperparameter tuning.

Coordinated Local Sampling Approaches close to ours are NetHash (Wu et al. 2018) and NodeSketch (Yang et al. 2019). NetHash uses minwise hashing to produce embeddings for attributed graphs. However, it builds individual rooted trees of depth k for each node and its complexity is $O(nt(m/n)^k)$ where t is the maximum number of attributes per node. LONE SAMPLER needs time $O(nt)$ to get the attribute with the minimum value for each node and thus the total complexity is $O(nt + mk)$. For $k > 1$ this is asymptotically faster by a factor of $t(m/n)^{k-1}$.

NodeSketch is a highly-efficient *heuristic* for coordinated sampling from local neighborhoods. It works by recursively sketching the neighborhood at each node until recursion depth k is reached. It builds upon an algorithm for min-max similarity estimation (Ioffe 2010). NodeSketch assigns first random weights to nodes, similarly to LONE SAMPLER . In the i-th recursive call of NodeSketch for each node u we collect samples from $N(u)$. For node u, a *single* node from $N(u)$ is selected and this is the crucial difference to LONE SAMPLER . Working with a single node for each iteration is prone to random effects, and the theoretical guarantees from (Ioffe 2010) hold only for sampling from the 1-hop neighborhood. Consider the graph in Figure 2. There are many paths from node u to node z in $N_2(u)$. In the first iteration of NodeSketch it is likely that most of u's immediate neighbors, the yellow nodes v_1 to v_{12}, will sample a blue node as each v_i is connected with many blue w nodes, i.e., $s_{v_i} = w_j$. Thus, in the second iteration it is likely that u ends up with a sample for u different from z. By keeping a sketch LONE SAMPLER *provably* preserves the information that node z is reachable from u by many different paths. And we control the importance we assign to sampling easily reachable nodes by reweighting the nodes by $r_i^{1/p}$.

Experiments

We evaluate LONE SAMPLER on six publicly available graphs, summarized in Table 1. The first three datasets Cora, Citeseer, Pubmed (Sen et al. 2008) are citation networks where nodes correspond to scientific papers and edges to citations. Each node is assigned a label describing the research topic, and nodes are described by key words. LastFM and Deezer represent the graphs of the online social networks of music streaming sites LastFM Asia and Deezer Europe (Rozemberczki and Sarkar 2020). The links represent follower relationships and the vertex features are musicians liked by the users. The labels in LastFM are the users' nationality, and for Deezer – the users' gender. The GitWebML graph represents a social network where labels correspond to web or ML developers who have starred at least 10 repositories and edges to mutual follower relationships. Node features are location, starred repositories, employer and e-mail address (Rozemberczki, Allen, and Sarkar 2021).

We sample node attributes by using the base sampling algorithm. For example, for each keyword *kw* describing the citation network nodes we generate a random number r_{kw} and then at each node we sample keywords according to the corresponding algorithm (e.g., we take the word with smallest r_{kw} for L_0 sampling, sample according to min-max sampling for NodeSketch (Yang et al. 2019), etc.)

We compare LONE SAMPLER against NodeSketch. Our L_0 sampling yields identical results to NetHash but, as discussed, our algorithm is much more efficient. Note that we don't compare with continuous embeddings. Such a comparison can be found in (Yang et al. 2019) but, as argued in the introduction, discrete embeddings offer various advantages such as interpretability and scalability. For completeness, we provide such a comparison in the full version of the paper.

We generated $d = 50$ samples from the k-hop neighborhood of each node for $k \in \{1, 2, 3, 4\}$, resulting in 50-dimensional embeddings, using following methods: i) A standard random walk (RW) of length k that returns the last visited node. ii) NodeSketch (NS) as described in (Yang et al. 2019). iii) LONE SAMPLER for L_p sampling for $p \in \{0, 1, 2\}$, using a sketch with 10 nodes for $p \in \{1, 2\}$. Note that the random walk embeddings are of much lower quality and some of the results are omitted here.

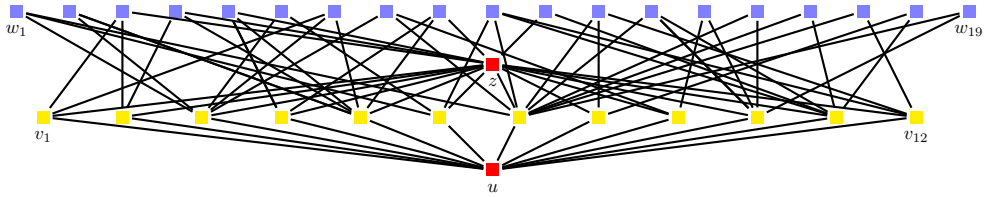

Figure 2: NodeSketch (Yang et al. 2019) might miss that there are many length-2 paths from u to z.

GRAPH	GRAPH STATISTICS						EMBEDDING TIME (in seconds)			
	nodes	edges	classes	features	feats per node	diameter	RW	NodeSketch	L_0	L_1/L_2
Cora	2.7K	5.4K	7	1.4K	18.2	19	2.1	2.7	1.5	25.7
Citeseer	3.3K	4.7K	6	3.7K	36.1	28	3.3	3.8	2.3	30.4
PubMed	19.7K	44.3K	3	500	50.5	18	17.3	35.2	15.3	214.3
Deezer	28.2K	92.7K	2	31K	33.9	21	37.0	62.5	33.4	344.4
LastFM	7.6K	27.8K	18	7.8K	195.3	15	20.4	68.6	18.9	128.5
GitWebML	37.7K	289K	2	4K	18.3	7	70.3	112.4	63.2	609.1

Table 1: Information on datasets and embedding generation time.

	kernel SVM	explicit map
Cora	0.3	0.7
Citeseer	0.5	1.2
Pubmed	26.3	12.2
Deezer	138	16.2
LastFM	2.1	2.4
GitWebML	OOM	15.9

Table 2: SVM training and inference time for NodeSketch (in seconds).

Embedding Evaluation

Running time The algorithms were implemented in Python 3 and run on a Linux machine with 16 GB main memory and a 4.3 GHz Ryzen 7 CPU. [2]. In the right half of Table 1 are results for the running time for 50-dimensional embeddings generation. We observe that L_0 sampling and NodeSketch are highly efficient, L_0 sampling is even faster than random walks as we need to generate only a single random number per node. L_1/L_2 sampling is slower which is due to the fact that we update a sketch at each node.

Sketch Quality We set $\varepsilon = 0.01$ and as $d = 50$ the expected error for each inner product is bounded by 0.5. In the full version of the paper we plot the ratio of the actual error to the expected error for the sketching procedure from Theorem 4, and for TensorSketch for a sample of 1,000 vector pairs. The higher variance in TensorSketch leads to outliers.

Node Classification We consider node classification for comparison of the different approaches. We use a linear SVM model with explicit feature maps as presented in Theorem 4, with tabulation hashing known to approximate the behavior of truly random hash functions (Pătraşcu and Thorup 2012). We split the data into 80% for training and 20% for testing, and use default SVM regularization parameters.

Training and Prediction Time In Table 2 we compare the running time for training and prediction of the linear SVM model as presented in Theorem 4, and an SVM model with precomputed kernel. The embedding vectors are computed by NodeSketch (the values for LONE SAMPLER are similar). Kernel SVMs are considerably slower for larger graphs and for GitWebML result in out-of-memory errors.

Classification Accuracy We evaluate the performance of the algorithms with respect to micro-AUC and macro-AUC. We report the mean and standard deviation of 10 independent train/test splits in Table 3. We see that overall L_1 and L_2 sampling achieve the best results. In particular, L_0 sampling yields good results only for a smaller neighborhood depth k, see details in the appendix. The reason is that for larger k and a small graph diameter many nodes end up with identical samples: for a connected graph with diameter k all nodes will have the same sampled node as their k-hop neighborhood is the node set V. In contrast, L_1 and L_2 sampling do not suffer from this drawback as samples depend on the graph structure, not just the set of reachable nodes.

Link Prediction We evaluate the generated sketches on a link prediction task following a similar approach as in (Yang et al. 2019). We remove 20% of the edges from each graph such that the graph remains connected. Then we sample 5% of all node pairs in the graph. We keep the 1,000 pairs with the largest overlap. For these 1,000 pairs we compute the precision@1000 and recall@1000. We report the mean and

[2]Code at https://github.com/konstantinkutzkov/lone_sampler

		NodeSketch	L_0	L_1	L_2
Cora	Micro AUC	0.972 ± 0.006 (hop 1)	0.970 ± 0.006 (hop 1)	**0.974 ± 0.005 (hop 1)**	*0.973 ± 0.005 (hop 1)*
	Macro AUC	*0.969 ± 0.006 (hop 1)*	0.967 ± 0.006 (hop 1)	**0.971 ± 0.005 (hop 1)**	*0.969 ± 0.005 (hop 1)*
Citeseer	Micro AUC	0.906 ± 0.008 (hop 2)	0.911 ± 0.009 (hop 1)	*0.915 ± 0.007 (hop 2)*	**0.917 ± 0.007 (hop 2)**
	Macro AUC	0.891 ± 0.010 (hop 4)	0.892 ± 0.008 (hop 2)	*0.898 ± 0.007 (hop 2)*	**0.903 ± 0.007 (hop 2)**
PubMed	Micro AUC	0.927 ± 0.003 (hop 1)	0.919 ± 0.002 (hop 1)	**0.938 ± 0.002 (hop 2)**	*0.932 ± 0.002 (hop 2)*
	Macro AUC	0.924 ± 0.003 (hop 1)	0.915 ± 0.002 (hop 1)	**0.935 ± 0.002 (hop 2)**	*0.928 ± 0.002 (hop 2)*
DeezerEurope	Micro AUC	0.588 ± 0.005 (hop 1)	*0.589 ± 0.004 (hop 1)*	*0.589 ± 0.006 (hop 2)*	**0.594 ± 0.005 (hop 2)**
	Macro AUC	0.561 ± 0.005 (hop 1)	0.564 ± 0.004 (hop 1)	*0.565 ± 0.006 (hop 2)*	**0.577 ± 0.005 (hop 2)**
LastFM	Micro AUC	**0.971 ± 0.003 (hop 2)**	0.966 ± 0.002 (hop 1)	0.969 ± 0.003 (hop 1)	*0.970 ± 0.003 (hop 1)*
	Macro AUC	0.935 ± 0.003 (hop 1)	0.924 ± 0.002 (hop 1)	*0.936 ± 0.003 (hop 1)*	**0.939 ± 0.003 (hop 1)**
GitWebML	Micro AUC	0.899 ± 0.003 (hop 1)	0.904 ± 0.002 (hop 1)	**0.908 ± 0.002 (hop 1)**	*0.905 ± 0.002 (hop 1)*
	Macro AUC	0.848 ± 0.003 (hop 1)	0.854 ± 0.002 (hop 1)	**0.862 ± 0.002 (hop 1)**	*0.855 ± 0.002 (hop 1)*

Table 3: The best result for each approach for the different hop depths. The overall best result is given in bold font, and the second best – in italics.

		NodeSketch	L_0	L_1	L_2
Cora	Precision	0.014 ± 0.003 (hop 1)	0.014 ± 0.004 (hop 1)	*0.016 ± 0.003* (hop 1)	**0.017 ± 0.003 (hop 1)**
	Recall	0.266 ± 0.057 (hop 1)	0.271 ± 0.063 (hop 1)	*0.317 ± 0.045 (hop 1)*	**0.334 ± 0.041 (hop 1)**
Citeseer	Precision	*0.020 ± 0.003 (hop 2)*	0.017 ± 0.004 (hop 4)	0.020 ± 0.003 (hop 2)	**0.021 ± 0.003 (hop 2)**
	Recall	*0.453 ± 0.041 (hop 2)*	0.385 ± 0.055 (hop 2)	0.452 ± 0.045 (hop 2)	**0.491 ± 0.066 (hop 2)**
PubMed	Precision	0.001 ± 0.001 (hop 1)	*0.011 ± 0.003 (hop 1)*	**0.025 ± 0.007 (hop 1)**	0.007 ± 0.003 (hop 1)
	Recall	0.002 ± 0.002 (hop 1)	*0.025 ± 0.006 (hop 1)*	**0.056 ± 0.015 (hop 1)**	0.016 ± 0.007 (hop 1)
DeezerEurope	Precision	< 0.001	< 0.001	< 0.001	< 0.001
	Recall	< 0.001	< 0.001	< 0.001	< 0.001
LastFM	Precision	0.024 ± 0.005 (hop 1)	0.031 ± 0.004 (hop 1)	*0.038 ± 0.004 (hop 1)*	**0.047 ± 0.005 (hop 1)**
	Recall	0.084 ± 0.016 (hop 1)	0.112 ± 0.016 (hop 1)	*0.136 ± 0.016 (hop 1)*	**0.167 ± 0.018 (hop 1)**
GitWebML	Precision	0.001 ± 0.001 (hop 1)	0.002 ± 0.001 (hop 1)	**0.018 ± 0.005 (hop 1)**	*0.016 ± 0.004 (hop 1)*
	Recall	0.002 ± 0.002 (hop 1)	0.003 ± 0.002 (hop 1)	**0.031 ± 0.008 (hop 1)**	*0.027 ± 0.006 (hop 1)*

Table 4: The best result (precision@1000 and recall@1000) for each approach for the different hop depths for link prediction. The overall best result is given in bold font, and the second best – in italics.

standard deviation for the best neighborhood depth for each approach in Table 4. LONE SAMPLER consistently achieves better results than NodeSketch, one order of magnitude better for the PubMed and GitWebML graphs. All approaches fail on Deezer because of the large number of node attributes which make the overlap very small. We observe that the average overlap for the LONE SAMPLER approaches is larger than for NodeSketch, and this has a more pronounced effect for link prediction, see discussion in appendix.

Optimal Values for the Hyperparameters In the appendix we analyze the optimal values for the different hyperparameters: the neighborhood depth k, the dimensionality of the embeddings d, and the sketch size ℓ for L_1/L_2 sampling. While the optimal k appears to be graph specific, we observe the improvements for larger d and ℓ to follow a diminishing returns pattern. It is worth noting that even for very small embedding dimensions we still achieve very good results for node classification. Also, for larger d NodeSketch catches up with L_1/L_2 sampling as more samples compensate for the inconsistencies due to its heuristic nature.

Conclusions and Future Work

We presented new algorithms for coordinated local graph sampling with rigorously understood theoretical properties. We demonstrate that using sketching techniques with well-understood properties also has practical advantages and can lead to more accurate algorithms in downstream graph learning tasks. Also, explicit feature maps of discrete node embeddings open the door to highly scalable classification algorithms. We made observations about the graph diameter and the performance of the different sampling strategies. More advanced concepts from graph theory will likely lead to a better understanding of the performance and limitations of the algorithms.

Finally, we would like to pose an open question. Can we learn structural roles (Rossi et al. 2020) by assigning appropriate node attributes? In particular, can we combine the sampling procedure with approaches to graph labeling such that node labels convey additional information?

References

Agarwal, P. K.; Cormode, G.; Huang, Z.; Phillips, J. M.; Wei, Z.; and Yi, K. 2013. Mergeable summaries. *ACM Trans. Database Syst.*, 38(4): 26:1–26:28.

Ahmed, N. K.; Duffield, N. G.; Willke, T. L.; and Rossi, R. A. 2017. On Sampling from Massive Graph Streams. *Proc. VLDB Endow.*, 10(11): 1430–1441.

Backstrom, L.; Boldi, P.; Rosa, M.; Ugander, J.; and Vigna, S. 2012. Four degrees of separation. In *Web Science 2012*.

Broder, A. Z.; Charikar, M.; Frieze, A. M.; and Mitzenmacher, M. 2000. Min-Wise Independent Permutations. *J. Comput. Syst. Sci.*, 60(3): 630–659.

Charikar, M.; Chen, K. C.; and Farach-Colton, M. 2004. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1): 3–15.

Cohen, E. 2016. Coordinated Sampling. In *Encyclopedia of Algorithms*, 449–454.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD 2016*.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *NIPS 2017*.

Ioffe, S. 2010. Improved Consistent Sampling, Weighted Minhash and L1 Sketching. In *ICDM 2010*.

Joachims, T. 2006. Training linear SVMs in linear time. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006*, 217–226.

Jowhari, H.; Saglam, M.; and Tardos, G. 2011. Tight bounds for Lp samplers, finding duplicates in streams, and related problems. In *PODS 2011*.

Karp, R. M.; Shenker, S.; and Papadimitriou, C. H. 2003. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28: 51–55.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS 2013*.

Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *KDD 2016*.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global Vectors for Word Representation. In *EMNLP 2014*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: online learning of social representations. In *KDD 2014*.

Pham, N.; and Pagh, R. 2013. Fast and scalable polynomial kernels via explicit feature maps. In *KDD*, 239–247. ACM.

Pătraşcu, M.; and Thorup, M. 2012. The Power of Simple Tabulation Hashing. *J. ACM*, 59(3): 14.

Rossi, R. A.; Jin, D.; Kim, S.; Ahmed, N. K.; Koutra, D.; and Lee, J. B. 2020. On Proximity and Structural Role-based Embeddings in Networks: Misconceptions, Techniques, and Applications. *ACM Trans. Knowl. Discov. Data*, 14(5): 63:1–63:37.

Rozemberczki, B.; Allen, C.; and Sarkar, R. 2021. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2).

Rozemberczki, B.; and Sarkar, R. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *CIKM 2020*, 1325–1334.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Gallagher, B.; and Eliassi-Rad, T. 2008. Collective Classification in Network Data. *AI Mag.*, 29(3): 93–106.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. LINE: Large-scale Information Network Embedding. In *WWW 2015*.

Tsitsulin, A.; Mottin, D.; Karras, P.; and Müller, E. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *WWW 2018*.

Wu, W.; Li, B.; Chen, L.; and Zhang, C. 2018. Efficient Attributed Network Embedding via Recursive Randomized Hashing. In *IJCAI 2018*.

Yang, D.; Rosso, P.; Li, B.; and Cudré-Mauroux, P. 2019. NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching. In *KDD 2019*.

Zhang, Z.; Cui, P.; Wang, X.; Pei, J.; Yao, X.; and Zhu, W. 2018. Arbitrary-Order Proximity Preserved Network Embedding. In *KDD 2018*.

Zhou, C.; Liu, Y.; Liu, X.; Liu, Z.; and Gao, J. 2017. Scalable Graph Embedding for Asymmetric Proximity. In *AAAI 2017*.