Scalable Optimization for Embedding Highly-Dynamic and Recency-Sensitive Data

Xumin Chen
Tsinghua University
chen.xm.mu@gmail.com

Peng Cui
Tsinghua University
cuip@tsinghua.edu.cn

Lingling Yi
Tencent Technology (Shenzhen) Co Ltd
chrisyi@tencent.com

Shiqiang Yang*
Tsinghua University
yangshq@mail.tsinghua.edu.cn

ABSTRACT

A dataset which is highly-dynamic and recency-sensitive means new data are generated in high volumes with a fast speed and of higher priority for the subsequent applications. Embedding technique is a popular research topic in recent years which aims to represent any data into low-dimensional vector space, which is widely used in different data types and have multiple applications. Generating embeddings on such data in a high-speed way is a challenging problem to consider the high dynamics and the recency sensitiveness together with both effectiveness and efficient. Popular embedding methods are usually time-consuming. As well as the common optimization methods are limited since it may not have enough time to converge or deal with recency-sensitive sample weights. This problem is still an open problem.

In this paper, we propose a novel optimization method named Diffused Stochastic Gradient Descent for such highly-dynamic and recency-sensitive data. The notion of our idea is to assign recency-sensitive weights to different samples, and select samples according to their weights in calculating gradients. And after updating the embedding of the selected sample, the related samples are also updated in a diffusion strategy.

We propose a Nested Segment Tree to improve the recency-sensitive weight method and the diffusion strategy into a complexity no slower than the iteration step in practice. We also theoretically prove the convergence rate of D-SGD for independent data samples, and empirically prove the efficacy of D-SGD in large-scale real datasets.

KEYWORDS

recency-sensitive data, embedding, dynamic, diffusion

*Tsinghua National Laboratory for Information Science and Technology(TNList)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '18, August 19–23, 2018, London, United Kingdom
© 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3219898

ACM Reference Format:
Xumin Chen, Peng Cui, Lingling Yi, and Shiqiang Yang. 2018. Scalable Optimization for Embedding Highly-Dynamic and Recency-Sensitive Data. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3219819.3219898

1 INTRODUCTION

Embedding techniques, aiming to represent the original data samples into low-dimensional vector space, have aroused considerable research interest in recent years. Many methods have been proposed to transform a variety of data types into embedding vectors, such as words [6], documents [11], images [1], users[4] and even networks [16]. These embeddings significantly facilitates data analysis and prediction by exploiting off-the-shelf machine learning methods. Therefore embedding techniques have been widely used in real applications, such as recommendation systems [13] and network analytics [14].

Here we consider a typical application scenario, where real data are generated in a highly-dynamic and recency-sensitive way. In such a scenario, new data is generated in high volumes with a fast speed, and the newly generated data is of higher priority for the subsequent applications. Take news recommendation as an example. We analyzed a large-scale article reading dataset from WeChat[1]. 8.1 new articles, in average, are generated every second. The interaction behaviors, i.e. a user read an article, are generated with the speed of 1, 400 times per second in average. Meanwhile, we find that these article reading behaviors are very recency-sensitive, i.e. users tend to read the latest articles. As shown in Figure 1a, most of the articles have very short life-cycles. About 73% of articles will never be read after 6 hours since their generation. Putting the characteristics of highly-dynamic and recency-sensitive together, we confront a challenging problem with respect to learning embeddings: how to generate embeddings for new articles and update embeddings for existing articles in a very efficient way, and incorporate as much new data (e.g. newly generated article reading behaviors) as possible to make the embeddings more effective?

The popular embedding methods, such as word2vec [6], doc2vec [11] and node2vec [7], are learning-based methods and thus need to involve optimization processes which are usually time-consuming. The commonly-used optimization methods include Gradient Descent (GD) [15], Stochastic Gradient Descent (SGD) [3] or other

[1]The largest social network platform in China, developed by Tencent.

variants like Adam [10]. In ideal cases where the computing power is unlimited, all these optimization methods can work well in our setting if they finally converge. In real industrial experience, however, the provided computing power cannot guarantee optimization processes to converge within an acceptable time duration, especially in highly-dynamic scenarios. The key problem, then, becomes how to distribute the limited computing power to optimize the samples with different priorities. Although Weighted SGD [12] provide the flexibility of assigning weights to samples, but how to deal with recency-sensitive data where sample weights may change over time, and attain a satisfactory convergence rate in real applications is still an open problem.

In this paper, we propose a novel optimization method named Diffused Stochastic Gradient Descent (D-SGD) that is specifically designed for embedding highly-dynamic and recency-sensitive data. The notion of our idea is to assign recency-sensitive weights to different samples, and select samples according to their weights in calculating gradients. By setting a higher weight to a newly generated sample, it will be selected with larger probability and thus account more for the resulted gradient. After updating the embedding of the selected sample, the embeddings of other samples that are related with the updated sample also need to be updated. For example, after updating the embedding of an article, the embeddings of the users who have read the article need to be updated. Therefore, we design a weight diffusion strategy to progagate the high weights to the samples related to the selected sample, so that the related samples will be selected and updated with high probability subsequently and iteratively. Inevitably, the weight diffusion strategy induces some additional computational cost in selecting samples, which affects its performance in highly-dynamic scenarios. To overcome it, we further design a nested segment tree for weighted sample selection, and realize it in $O(\log(N))$ where N is the total sample size. Finally, we theoretically prove that the convergence rate of D-SGD is guaranteed if samples are independent, and empirically prove that D-SGD can achieve best performance in real data where samples are not independent.

It is worthwhile to highlight our contributions.

(1) We propose a novel optimization method D-SGD for embedding highly-dynamic and recency-sensitive data.
(2) We design a weight diffusion strategy to assign sample weights for recency-sensitive data and a distributed segment tree to improve the efficiency of weighted sample selection.
(3) We theoretically prove the convergence rate of D-SGD for independent data samples, and empirically prove the efficacy of D-SGD in large-scale real datasets.

The remained sections are organized as follows. We briefly review the related works in section 2, provide the motivation and formulation of dynamic embedding on recency data in section 3, introduce the details of D-SGD in section 4, present the experimental results in section 5 and draw the conclusions in section 6.

2 RELATED WORK

Here we briefly review the prior works in two closely related directions: dynamic embedding and online optimization.

Dynamic embedding. There have been many methods proposed to learn embedding vectors of words[6], documents[11], images[1] and even networks. Network embedding is a class of methods to map network objects like vertices and edges into vectors. For instance, LINE [16], DeepWalk[14] and node2vec[7] are some popular network embedding methods in recent years. However, these optimization process of these algorithms are complicated, making them inadequate in embedding highly dynamic and recency-sensitive data.

Word embedding is also a popular research problem in recent years. Instead of paying attention on efficiency and recency, recent works are mostly focus on learning the embedding continuously. For example, [2] improves word2vec so that the embedding is trained to be similar with the old one when items change. [17] modified the dynamic word embedding to discover evolving semantic. Both of them need to scan all of the current dataset at a new timestamp which is not practical for our highly dynamic problem.

Online optimization algorithms. Online optimization is a subdomain of optimization theory[5] and [9]. Instead of providing general ways to find the optimal solution for a specified problem, works in this domain often study the convergence or other property for a optimization method. In [8], authors discuss a class of methods known as online convex optimization to tackle loss functions which change in many specified ways. It also theoretically proves boundaries and convergence if loss functions change more generally. These methods do not aim at our problem, which focus more on the practical performance in highly-dynamic and recency-sensitve scenarios. Though these cannot be applied into our problem, some of them like [12] provide the important basis to prove the boundary and convergence of our method.

On the other hand, element-wised or batch-wised optimization algorithms, like Stochastic Gradient Descent(SGD)[3], are widely used as general frameworks. Among these algorithms, Adam[10] is a state-of-art one which is widely used, and we use it as one of our baselines.

3 DYNAMIC EMBEDDING ON RECENCY DATA

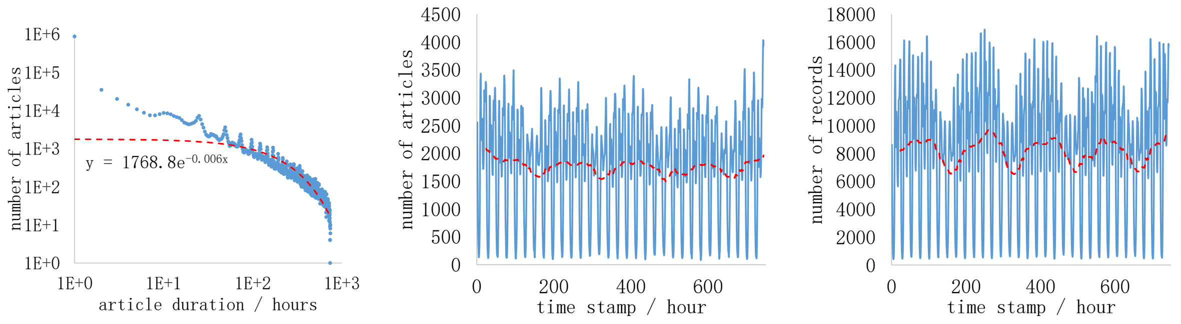
3.1 Motivation

Our study in this paper is mainly motivated by our observations in two real datasets collected from WeChat.

WeChat article reading dataset (WA). In WeChat, a widely-used function is 'Moments', where a user can post information there and his/her friends can read the posted information in their own 'Moments', like wall post function in Facebook. In this dataset, the users' article reading behaviors are recorded. Each record is formated into a triplet (*user*, *article*, *time_stamp*), meaning a *user* reads an *article* at time *time_stamp*.

WeChat friendship network dataset (WF). WeChat is an undirected social network. Users can establish friendship links with each other. In this dataset, the dynamic process of users establishing friendship links are recorded. Each record is formated into a triplet ($user_i$, $user_j$, *time_stamp*), meaning $user_i$ and $user_j$ establish a link at *time_stamp*.

The detailed statistics of these two datasets are summarized as Table 1.



(a) Duration of each article **(b) Number of new articles each hour** **(c) Number of new records each hour**

Figure 1: Dynamic and recency of the dataset

Table 1: Statistics of the datasets

Name	WeChat article	WeChat friend
max $\|V\|$	$39,951(users) + 1,312,327(articles)$	42, 589
max $\|E\|$	6, 105, 938	4, 291, 256
time span	1 month	6.5 years
time unit	second	second

From these two datasets, we observe similar highly-dynamic and recency-sensitive applications scenarios. For brevity, we only show the statistical analysis results of WA dataset in Figure 1. We first plot the distribution of article duration in Figure 1a, where the duration of an article is calculated by the difference between the first and last timestamps that the article gets read. We can see that the duration of more than 73% articles are less than 6 hours, and there is an obvious exponential decay in the distribution. The distribution demonstrates that users' article reading behaviors are very recency-sensitive and most users only care about latest articles. Also, we plot the number of new articles and new behavior records generated per hour in Figure 1b and 1c. From the embedding perspective, we need to generate new embeddings for 2000 new articles per hour, and update embeddings for 8000 existing articles and their related users per hour. Note that this is only a sampled dataset of WeChat, and the generating speed of new articles and behaviors is much higher. Such an application scenario requires effective and efficient method for dynamic embedding on recency data.

3.2 Problem Formulation

With generality, we represent data into a matrix $\mathbf{A}$. In WA dataset, the rows of $\mathbf{A}$ represent users, the columns represent articles, and a_{ij} represents the weight between user i and article j. In WF dataset, $\mathbf{A}$ is symmetric where both rows and columns represent users, and the a_{ij} represents the weight between user i and user j. Here $\mathbf{A}$ is also equivalent with a graph $G = (V, E)$, where V is the vertex set and E is the edge set. It is a bipartite graph in WA dataset and undirected plain graph in WF dataset. In later sections, we use either $\mathbf{A}$ or G to represent the data without confusion.

In a dynamic setting, the matrix $\mathbf{A}$ will evolve over time, resulting in a dynamic dataset $\mathbf{A}^{(t)}$ or $G^{(t)} = \left(V^{(t)}, E^{(t)}\right)$, where $t = 1, 2, \ldots$ is the timestamp. In this paper, the timestamp t is defined in event time, i.e. only one node or edge is updated between two consecutive timestamps. There are two cases in the dynamic dataset. One is rows or columns in $\mathbf{A}$ are added or deleted (i.e. new nodes are added into or existing nodes are deleted from G), and the other is the entries in $\mathbf{A}$ are updated (i.e. new edges are added into or existing edges are deleted from G). As adding nodes and deleting nodes can be equivalently reflected by adding and deleting edges, we unify the two cases by only considering the updating of entries in $\mathbf{A}$.

Here we define the problem of dynamic embedding on recency data based on commonly-used matrix factorization framework, as follow:

Definition 3.1 (Dynamic embedding on recency data). Given a dynamic dataset $\mathbf{A}^{(t)}$, finding the optimal embedding matrices $\mathbf{U}$ and $\mathbf{V}$, so that the following objective function can be minimized

$$\begin{aligned} J(\mathbf{U}, \mathbf{V}) &= \left\| \mathbf{W}^{(t)} \circ (\mathbf{A}^{(t)} - \mathbf{U}^{\mathsf{T}}\mathbf{V}) \right\|_F^2 \\ &= \sum_{i,j} w_{i,j}^{(t)} \left(\mathbf{u}_i^{\mathsf{T}} \cdot \mathbf{v}_j - a_{i,j}^{(t)} \right)^2 \end{aligned}, \quad (1)$$

where $\mathbf{W}^{(t)}$ is the recency weight matrix, and $w_{i,j}^{(t)}$ represents the weight of $a_{i,j}^{(t)}$.

Note that the recency weight matrix $\mathbf{W}^{(t)}$ change over time. We define the recency weighting strategy as follow:

$$w_{i,j}^{(t+1)} = \begin{cases} 1 & \text{if } (i,j) \text{ is the newly added or deleted edge} \\ \tau w_{i,j}^{(t)} & \text{elsewise} \end{cases}, \quad (2)$$

where $\tau \in (0, 1)$ is the decay factor that controls the degree of recency sensitivity. This recency weighting strategy enforces that the embeddings of the nodes linked by the newly updated (added or deleted) edge shouold be more emphasized in the learning process. As $\mathbf{W}^{(t)}$ changes very fast in highly dynamic environment, all optimization methods cannot guarantee to converge between two timestamps given limited computing power. Then how well the recency weights can be incorporated in the embedding learning heavily depend on the effectiveness and efficiency of the underlying optimization method.

4 OPTIMIZATION

4.1 A Base Optimization Method

For ease of understanding, we call an edge as a data sample and a node as an object in the optimization process. Let us first define an entry-wise loss function according to equation (1):

$$f(\mathbf{u}_i, \mathbf{v}_j, a^t_{i,j}) = \left(\mathbf{u}_i^\top \cdot \mathbf{v}_j - a^{(t)}_{i,j}\right)^2. \tag{3}$$

If we assume that the optimization process can converge between time t and $t+1$, we can regard the weights of samples are unchanged in the optimization process and directly use Weighted SGD [12] to optimize it. In each iteration step, Weighted SGD randomly select an edge according to $w^{(t)}_{i,j}$ and update the embedding matrices with gradients as follow:

$$\begin{aligned}\mathbf{u}_i(r) &\leftarrow \mathbf{u}_i(r-1) - \eta \frac{\partial f}{\partial \mathbf{u}_i}\left(\mathbf{u}_i(r-1), \mathbf{v}_j(r-1), a_{i,j}\right)\\ \mathbf{v}_j(r) &\leftarrow \mathbf{v}_j(r-1) - \eta \frac{\partial f}{\partial \mathbf{v}_j}\left(\mathbf{u}_i(r-1), \mathbf{v}_j(r-1), a_{i,j}\right)\end{aligned}, \tag{4}$$

where η is the learning rate and $r \geq 1$ is the iteration step. When it comes to timestamp $t+1$, we set $\mathbf{U}^{(t+1)}(0) = \mathbf{U}^{(t)}(r), \mathbf{V}^{(t+1)}(0) = \mathbf{V}^{(t)}(r)$.

In highly-dynamic environment, we need to consider the cases when the optimization process cannot converge between two timestamps. In such cases, the Weighted SGD method have the following limitations.

Redundant. Though the weights of samples are dynamic along different timestamps, the weight is fixed across different iteration steps given a certain timestamp t. This cause SGD to probably select the new sample multiple times as it always have high weights across all iteration steps before next timestamp. However, training the embedding of a node with the same linked edge repeatedly is meaningless if this embedding was not updated by other edges. Also, if the embedding of a node is updated, the embeddings of its neighbors should also be updated. But there is no mechanism in SGD to realize this propagation.

Incomplete training. As weighted SGD select edges according to $w^{(t)}_{i,j}$, if some existing edges have not been selected enough times to attain full training on their related nodes before a new edge coming, the weights of these existing edges will decrease and thus less likely to be selected any more, resulting in incomplete training on node embeddings.

Inefficient. In highly dynamic setting, the speed of edge updating is very fast. Even if we cannot guarantee the optimization process converge between two consecutive timestamps, more iteration steps in-between two timestamps can necessarily bring better optimization effect. However, in the base optimization method, the weights of all edges are updated at each timestamp, and weighted SGD selects samples according to the updated weights, inducing a complexity of $O(|E|)$ in random sample selection. Such a complexity is unaffordable in highly dynamic setting.

The limitations of the base optimization method motivate us to propose a new optimization method, Diffused SGD, specially designed for highly-dynamic and recency-sensitive data.

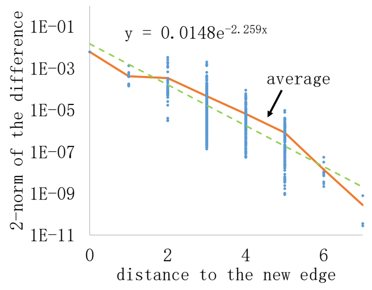


Figure 2: Difference of each object after inserting a relationship

4.2 Diffused SGD Method

4.2.1 Weight Diffusion Mechanism. In order to address the first two limitations, we first conduct empirical analysis in synthetic data to find the characteristics of embeddings in a dynamic dataset. More specifically, we study how embeddings change when data dynamically changes. We generate some networks with power-law degree distributions, and dynamically add a new edge at each timestamp. We use singular value decomposition (SVD) to generate node embeddings at each timestamp, and quantify the difference of embedding vectors in two consecutive timestamp for each node. Then we plot the distribution of embedding difference of a single node versus its distance to the newly added edge, as shown in Figure 2. We can see that the embedding differences exponentially decay with the distance becoming larger. This implies that the embedding updating of nodes should be diffused from the newly updated edge to neighboring nodes.

Motivated by this, we design a iteration step-wise weight diffusion mechanism. We use $p^{(t)}_{i,j}(r)$ as the weight and set $p^{(t)}_{i,j}(0) = 1$ if edge (i,j) is a new edge inserted at timestamp t. For each iteration step r, if edge (i,j) is selected and $\mathbf{u}_i$ and $\mathbf{v}_j$ are updated with equation (4), we update the weight of edge (i,j) and all of node i's neighbors in following way:

For edge (i,j), we have

$$p_{i,j}(r) \leftarrow \tau_e\left(i, p_{i,j}(r-1)\right); \tag{5}$$

for $(i,k) \in \mathbb{E} \wedge k \neq j$, we use

$$p_{i,k}(r) \leftarrow p_{i,k}(r-1) + \tau_n\left(i, p_{i,j}(r-1)\right); \tag{6}$$

and for other edges $(l,k) \in \mathbb{E} \wedge l \neq i$,

$$p_{l,k}(r) \leftarrow p_{l,k}(r-1); \tag{7}$$

where τ_e and τ_n are two decay functions, which are defined as

$$\begin{aligned}\tau_e(i,p) &= \frac{\tau p}{2}\\ \tau_n(i,p) &= \frac{\tau p}{2OD^{(t)}(i)}\end{aligned}, \tag{8}$$

where $OD^{(t)}(i)$ is the out-degree of vertex i in graph $G^{(t)}$. And in-degree is used when updating $\mathbf{v}_j$, symmetrically.

Equation (5) means that the weight of the new edge decays over iteration steps. In this way, the new edge will not be selected too many times to avoid redundant optimization. While equation

(6) means the weights of other links to the updated node i are increased, so that they will be selected with higher probability and the embedding updating can be diffused along neighborhood structures. Although this is a one-hop diffusion, it can realize global diffusion in a iterative way.

This method guarantees the probability of old edges to be less than new edges. Also, if an existing edge well trained, i.e. it is selected few times, its weight will not decay and make it more possibly to be selected in future. Those said, the first two limitations of the base optimization method can be largely alleviated by the weight diffusion mechanism. Furthermore, the mechanism can be flexibly combined into the existing optimization methods like Weighted SGD, and we can theoretically prove that such a mechanism will not affect the convergence of an optimization method, which will be introduced in section 4.3.

4.2.2 Nested Segment Trees for Weighted Sampling. In this section, we focus on addressing the third limitation of base optimiza-tiom method, the efficiency issue. As mentioned that the bottle-neck of efficiency is the weighted sampling. With the weight diffusion mechanism, we have already decreased the number of updated weights from $O(|E|)$ into $O(OD(i))$ in each step. However, we still need to further improve efficiency because:

(1) Even using a faster algorithm as mentioned in [16], generating a random table and conducting weighted sample selection still needs to traverse all of the edges.
(2) $O(OD(i))$ is still too large since there are often some hub nodes with very large degree in power-law graphs.

We can directly construct a Segment Tree [2] to maintain the random table, and thus we can randomly select an edge according to edge weights in $O(\log|E|)$. In order to further improve the efficiency, we propose a Nested Segment Trees with lazy propagation. With lazy propagation, we can change the consecutive elements of an array into the same value in $O(\log N)$ time, and also query the the values of consecutive elements in $O(\log N)$ time, where N is the number of elements of the array. Then we propose the nested segment tree to transform the neighbors in a graph to consecutive elements in an array.

Here are the steps of this method:

Nested Segment Trees with lazy propagation. Firstly, we build a segment tree $T^{(t)*}$ whose leaves represent every vertex in $V^{(t)}$.

For each vertex i, we build two segment trees $T_i^{(t)+}$ and $T_i^{(t)-}$ whose leaves respectively represent every neighbors through each edge satisfies $(i,j) \in E$ and $(j,i) \in E$.

The weight $w_{i,j}$ is separately stored in two leaf nodes (i,j) in T_i^+ and (j,i) in T_j^-. While the leaf node i in T^* represents the summation of weight on T_i^+ and T_i^- which does NOT represent $\sum_{(i,j)\in E} w_{i,j}$ but part of it. Each non-leaf node in T^*, T_i^+ and T_i^- represents the summation weight of the sub-tree whose root is this node.

When we need to plus a same value on each edge from i except (i,j), we are to modify two intervals on T_i^+, while two intervals on T_i^- when updating the edges to i except (j,i). With lazy modification method, we can apply this change in $O(\log(OD(i)))$. After that, we can calculate the new summation of T_i^+ and T_i^- and update T^* in $O(\log|V|)$.

When we need to roll an edge with the probability proportional to its weight, we firstly random a real number in range $\left[0, \sum w_{i,j}\right]$ while $\sum w_{i,j}$ is maintained by the root of T^* and then put the number on the root of T^*.

When the number is on any non-leaf node, compare it with the summation maintained by the left child. If less or equal, then move to the left child, else move to the right child after minus the summation on the left child.

When the number reaches a leaf node which represents i, we can determine this number is on either T_i^+ or T_i^-. Then we use the similar process to choose T_i^+ or T_i^- and one leaf on it which means we get an edge randomly by the real number.

When a vertex or an edge is inserted or deleted, we can use full-double and half-full strategy to get a complexity of $O(1)$ in average for tree reshape.

□

In summary, we can accomplish the weight diffusion and weighted sampling with $O(\log|E|)$ in average for each iteration step, and finish updating node embeddings with $O(d)$ where d is the dimension of the embedding of one node.

4.3 Boundary and convergence

In optimization theory, convergence is often scrutinized on smooth functions which is strongly, strictly or weakly convex. For instance, if we fix t and $\mathbf{V}$, $J(\mathbf{U})$ in (1) convex and smooth, and strongly convex under some conditions.

Here we analyze an extension of (1), as follow, which is obviously smooth.

$$\begin{aligned} J(\mathbf{U},\mathbf{V}) &= \left\|\mathbf{W}^{(t)}(r) \circ \left(\mathbf{A}^{(t)} - \mathbf{U}^\top\mathbf{V}\right)\right\|_F^2 \\ &= \sum_{(i,j)\in E^{(t)}} w_{i,j}^{(t)^2}(r)\left(\mathbf{u}_i^\top \cdot \mathbf{v}_j - a_{i,j}^{(t)}\right)^2 + \sum_{(i,j)\notin E^{(t)}} w_{i,j}^{(t)^2}(r)\left(\mathbf{u}_i^\top \cdot \mathbf{v}_j\right)^2 \end{aligned} \tag{9}$$

A function $f : X \to Y$ with gradient is convex if and only if there exists an $\mu \le 0$, for any $\mathbf{x}_1, \mathbf{x}_2 \in X$,

$$f(\mathbf{x}_2) - f(\mathbf{x}_1) \ge \nabla f(\mathbf{x}_1)^\top(\mathbf{x}_2 - \mathbf{x}_1) + \frac{\mu}{2}\|\mathbf{x}_2 - \mathbf{x}_1\|_2^2 . \tag{10}$$

If $\mu > 0$ we call it a strongly convex function, and μ is defined as the strongly convex parameter.

Theorem 4.1. *Equation* (9) *is strongly convex.*

Proof. For any $\mathbf{U}_1, \mathbf{U}_2 \in \mathbb{R}^{d\times|\mathbb{V}|}$,

$$\begin{aligned} & J(\mathbf{U}_2) - J(\mathbf{U}_1) - J(\mathbf{U}_1)^\top(\mathbf{U}_2 - \mathbf{U}_1) \\ =& \left\|\mathbf{W} \circ (\mathbf{U}_2 - \mathbf{U}_1)^\top \mathbf{V}\right\|_F^2 \\ =& \sum_{i\in V} \left(\mathbf{u}_{2_i} - \mathbf{u}_{1_i}\right)^\top (\mathbf{w}_i \circ \mathbf{V})(\mathbf{w}_i \circ \mathbf{V})^\top \left(\mathbf{u}_{2_i} - \mathbf{u}_{1_i}\right) \\ =& \sum_{i\in V}\sum_{j\in V} \left(w_{i,j}\left(\mathbf{u}_{2_i} - \mathbf{u}_{1_i}\right)^\top \mathbf{v}_j\right)^2 . \end{aligned}$$

This equation is non-negative, i.e. this is a convex function. We can also infer that $(\mathbf{w}_i \circ \mathbf{V})(\mathbf{w}_i \circ \mathbf{V})^\top$ is a non-negative-definite quadratic

[2] A brief introduction to basic Segment Tree with lazy propagation can be found at https://www.geeksforgeeks.org/lazy-propagation-in-segment-tree/

matrix since $\sum_{j \in V} \left(w_{i,j} \left(\mathbf{u}_{2_i} - \mathbf{u}_{1_i} \right)^\top \mathbf{v}_j \right)^2 \geq 0$. Furthermore, if we assume that $(\mathbf{w}_i \circ \mathbf{V})(\mathbf{w}_i \circ \mathbf{V})^\top$ is a non-singular matrix, which is almost always true in practice, then we have

$$\left(\mathbf{u}_{2_i} - \mathbf{u}_{1_i} \right)^\top (\mathbf{w}_i \circ \mathbf{V}) (\mathbf{w}_i \circ \mathbf{V})^\top \left(\mathbf{u}_{2_i} - \mathbf{u}_{1_i} \right) \geq \lambda_{\min} \left\| \mathbf{u}_{2_i} - \mathbf{u}_{1_i} \right\|_2^2,$$

where $\lambda_{\min}$ is the smallest eigenvalue of $(\mathbf{w}_i \circ \mathbf{V}) (\mathbf{w}_i \circ \mathbf{V})^\top$ which must be positive. Therefore, we can prove that (1) is strongly convex. □

In our case, the objective function is summed by strongly convex sub-functions $J(\mathbf{U}) = \sum_{i,j} J_{i,j}(\mathbf{U})$ optimized by SGD. The initial error ε_0 is defined as $\|\mathbf{U}(0) - \mathbf{U}_*\|_F^2$, where $\mathbf{U}_*^{(t)}$ is the optimized vector and $\mathbf{U}(r)$ is the vector after r training iterations. Current error is ε that $\mathbb{E} \|\mathbf{U}(r) - \mathbf{U}_*\|_F^2 \leq \varepsilon$. According to the theory of Weighted Stochastic Gradient Descent [12], we have the following theorem.

Theorem 4.2. *If we sample sub-functions with probability proportional to a weight function, the expecting step size r is bounded by ε, ε_0 and the Lipschitz constant.*

As a result, each step in our training step is expectedly getting closer to the optimized value of the objective function at that time whatever the weight function is.

Theorem 4.3. *As edge (i, j) added, i.e. $a_{i,j}$ is set from zero to non-zero. We have the difference of optimal embeddings in two consecutive timestamps bounded:*

$$\frac{\mu}{2} \sum_{k \neq i} \left\| \mathbf{u}_{k*}^{(t+1)} - \mathbf{u}_{k*}^{(t)} \right\|_2^2 + \left\| \sqrt{\frac{\mu}{2}} \left(\mathbf{u}_{i*}^{(t+1)} - \mathbf{u}_{i*}^{(t)} \right) + \sqrt{\frac{2}{\mu}} w_{i,j}^{(t+1)^2} a_{i,j}^{(t)} \mathbf{v}_j \right\|_2^2$$
$$\leq \left(\left| w_{i,j}^{(t+1)^2} - w_{i,j}^{(t)^2} \right| + \frac{2}{\mu} w_{i,j}^{(t+1)^4} a_{i,j}^{(t)^2} \right) \mathbf{v}_j^\top \mathbf{v}_j.$$

Proof. We denote

$$\Delta(\mathbf{U}) = J^{(t+1)}(\mathbf{U}) - J^{(t)}(\mathbf{U})$$
$$= \left(w_{i,j}^{(t+1)^2} - w_{i,j}^{(t)^2} \right) \left(\mathbf{u}_i^\top \mathbf{v}_j^{(t)} \right)^2 - w_{i,j}^{(t+1)^2} \left(2 a_{i,j}^{(t)} \mathbf{u}_i^\top \mathbf{v}_j^{(t)} - a_{i,j}^{(t)^2} \right).$$

Then we can infer that

$$J^{(t+1)} \left(\mathbf{U}_*^{(t+1)} \right) = J^{(t)} \left(\mathbf{U}_*^{(t+1)} \right) + \Delta \left(\mathbf{U}_*^{(t+1)} \right) \geq J^{(t)} \left(\mathbf{U}_*^{(t)} \right) + \Delta \left(\mathbf{U}_*^{(t+1)} \right)$$
$$J^{(t+1)} \left(\mathbf{U}_*^{(t)} \right) = J^{(t)} \left(\mathbf{U}_*^{(t)} \right) + \Delta \left(\mathbf{U}_*^{(t)} \right)$$

$$\frac{\mu^{(t+1)}}{2} \left\| \mathbf{U}_*^{(t+1)} - \mathbf{U}_*^{(t)} \right\|_F^2$$
$$\leq J^{(t+1)} \left(\mathbf{U}_*^{(t)} \right) - J^{(t+1)} \left(\mathbf{U}_*^{(t+1)} \right) - \nabla J^{(t+1)} \left(\mathbf{U}_*^{(t+1)} \right)^\top \left(\mathbf{U}_*^{(t)} - \mathbf{U}_*^{(t+1)} \right)$$
$$\leq \Delta \left(\mathbf{U}_*^{(t)} \right) - \Delta \left(\mathbf{U}_*^{(t+1)} \right)$$
$$= \left(w_{i,j}^{(t+1)^2} - w_{i,j}^{(t)^2} \right) \left(\left(\mathbf{u}_{i*}^{(t+1)^\top} \mathbf{v}_j \mathbf{v}_j^\top \mathbf{u}_{i*}^{(t+1)} \right) - \left(\mathbf{u}_{i*}^{(t)^\top} \mathbf{v}_j \mathbf{v}_j^\top \mathbf{u}_{i*}^{(t)} \right) \right)$$
$$- 2 w_{i,j}^{(t+1)^2} a_{i,j}^{(t)} \left(\mathbf{u}_{i*}^{(t+1)} - \mathbf{u}_{i*}^{(t)} \right)^\top \mathbf{v}_j$$
$$\leq \left| w_{i,j}^{(t+1)^2} - w_{i,j}^{(t)^2} \right| \mathbf{v}_j^\top \mathbf{v}_j - 2 w_{i,j}^{(t+1)^2} a_{i,j}^{(t)} \left(\mathbf{u}_{i*}^{(t+1)} - \mathbf{u}_{i*}^{(t)} \right)^\top \mathbf{v}_j$$

Then we can infer the formula in Theorem 4.3 is correct. □

Thus we have the difference between $\mathbf{U}_*^{(t+1)}$ and $\mathbf{U}_*^{(t)}$ bounded. It is obvious that the boundary of $\mathbf{u}_i$ is looser than other nodes, which may explain why we need to pay special attention to the nodes linked with the newly added edge. When edge (i, j) is deleted, we can get a similar conclusion.

In a similar way, we can also prove the following theorem:

Theorem 4.4. *For edge (i, j), if $w_{i,j}^{(t)}(r)$ is decreased into $w_{i,j}^{(t)}(r+1)$, we have*

$$\frac{\mu}{2} \sum_{k \neq i} \|\mathbf{u}_{k*}(r+1) - \mathbf{u}_{k*}(r)\|_2^2 +$$
$$\left\| \sqrt{\frac{\mu}{2}} \left(\mathbf{u}_{i*}(r+1) - \mathbf{u}_{i*}(r) \right) + \sqrt{\frac{2}{\mu}} \left(w_{i,j}^2(r+1) - w_{i,j}^2(r) \right) a_{i,j} \mathbf{v}_j \right\|_2^2$$
$$\leq \left(\left| w_{i,j}^2(r+1) - w_{i,j}^2(r) \right| + \frac{2}{\mu} \left(w_{i,j}^2(r+1) - w_{i,j}^2(r) \right)^2 a_{i,j}^2 \right) \mathbf{v}_j^\top \mathbf{v}_j.$$

As the $w_{i,j}(r)$ is decreasing in an exponential speed, this boundary finally converge to zero.

Because the movements of optimal embeddings in different timestamps are bounded, the distance between the current embeddings and the optimal ones can not change dramatically if we change the objective function locally and slightly. As the convergence rate is bounded by the difference between the current embeddings and the optimal embeddings, we can tweak the objective function in some subtle way to reflect the characteristic of the real data without damaging the convergence process.

For a smooth and convex function without the condition of strong convex, we may not prove the above boundaries. In this case, although the optimal embeddings of next timestamp may move a long distance from the current optimial embeddings, there are such embeddings nearby the current optimal embeddings that are sub-optimal for next timestamp but can achieve similar performances as the optimal embeddings of next timestamp.

5 EXPERIMENT

5.1 Experiment setup

In our experiment, we use the two datasets which are introduced in section 3.

We use the two datasets introduced in section 3.1 in our experiments. We select articles which are read for at least 50 times in the WA dataset and use all of the data in WF dataset. Also, we use the first 10% of data as the initial training data in both datasets. For every timestamp t in the remained data, we use $G^{(t)}$ as the training set, the newly updated edge in $G^{(t+1)}$ as the testing set. Note that only one edge is updated between two consecutive timestamps in our setting, so each test case only contains one edge.

We compare our method with baselines in different aspects. First, we select the following optimization framework as baselines:

- Stochastic Gradient Descent(SGD): A general, classic and widely used optimization framework. It randomly selects one sample, which means an edge in our setting, per iteration step. The probability of selecting a sample is proportional to its weight, which is either static or globally decay. The learning rate is set as 0.05.

- Adam: A state-of-the-art optimization framework which is often used for sparse data with multiple biases. This framework is usually batched, and we also select batches from data with static weights or globally decayed weights. Learning rate is set as 0.007 and $\beta_1 = 0.9$, $\beta_2 = 0.999$.

We also summarize the different recency strategies here:

- Static Weight(SW): For any edge, either new or existing, we use the same weight in optimization framework.
- Globally Decay(GL): New edges have a largest weight while existing ones' are decayed as time goes by. We use the equation (2) as the strategy with $\tau = 0.05$. Since the weights are changing, we use basic segment tree to optimize its efficiency.

By combining the optimization frameworks with recency strategies, we get 4 baselines: SGD-SW, SGD-GL, Adam-SW and Adam-GL. Our method D-SGD exploit SGD framework empowered with the weight diffusion mechanism and nested segment tree with lazy propagation. We use equation (8) as the weight decay functions with $\tau = 0.005$. Learning rate is set as 0.05. For all of the methods above, we use the same negative sampling ratio 5, and the same embedding dimension $d = 40$.

In order to simulate a highly-dynamic and recency-sensitive scenario and conduct fair comparisons among baselines and our method, we design the following settings:

- Similar Runing Time (ST): We fix the running time for all the above methods, and evaluate the resulted embeddings in their prediction accuracy in real applications. In order to simulate highly dynamic scenarios, we set the runnning time to a relative short duration.
- Batch-wise Re-train (BRT): Another commonly used setting is batch-wise retrain. We run re-training for every 3.3% of data and use the resulted embeddings to predict the following 3.3% of data. In this setting, we let the baseline algorithms to converge.

In experiments, we mainly report the results of BRT in Adam and SGD. Due to the efficiency issue of SGD-BRT-GL, we omit its results. Our method is specially designed for highly-dynamic and recency-sensitive data, so we only report its performance in ST setting.

Note that, for all methods except the BRT mode, training and testing are conducted in an edge-wised manner. When a new edge is added, we first use the current embeddings to predict it and count it into the testing performance, and then use the edge to train and update embeddings, and use the updated embeddings to predict next new edge. Iteratively, we can report the average testing performance.

5.2 Recommendation and Link Prediction

We first testify all the methods on their prediction performances in highly-dynamic and recency-sensitive setting. We first define the following two variables:

- Running duration δt: for each timestamp t, each optimization method can only run for a time duration δt. It is used to simulate highly dynamic scenarios, where only a small optimization duration is allowed between two timestamps.
- Recency ratio c: for each article we test the prediction performance of first c reading records, and report the average performance over all articles. It is used to simulate recency-sensitive scenarios, where the edges linking to a new node is limited.

For different $\max r$ and different c, we calculate the AUC(Area under the curve) of each algorithm their prediction accuracy in both recommendation or link prediction. In ths experiment, we constraint all methods to run. The results on WA dataset are shown in Figure 3a, 3b and 3c.

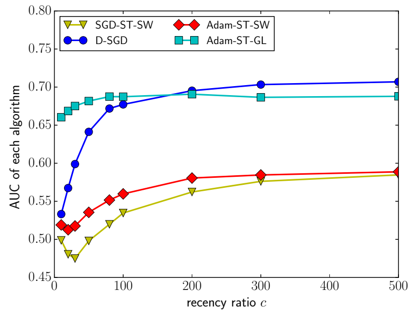
From the figure we have the following observations.

(1) Overall, D-SGD performs much better than other baselines, no matter varying runnng duration δt or recency ratio c.
(2) Given a certain running duration, we can see that the prediction performance of D-SGD is the best in most settings. When the recency ratio is very small, all methods performs similar, as there is no sufficient information about the new articles to make reliable prediction. But as the recency ratio c becomes larger, the performance of D-SGD grows much faster than other baselines, demonstrating that D-SGD can fully and effectively exploit the new edges to update embeddings.
(3) Given a certain recency ratio, the overall performance of D-SGD is the best. When the running duration is very small, D-SGD do not show any advantage, but when the running duration becomes larger, he performance of D-SGD improves much faster than other baselines, demonstrating that the edge weights are effectively diffused in each iteration.
(4) When both the recency ratio and running duration is very small, Adam-ST-GL works better than D-SGD. This is reasonable because Adam-ST-GL will repeatedly select the new edges, and thus the embeddings of new articles will be updated frequently. But when the recency ratio or running duration becomes larger, the advantage of Adam-ST-GL disappears, because of its redundant optimization on new edges.

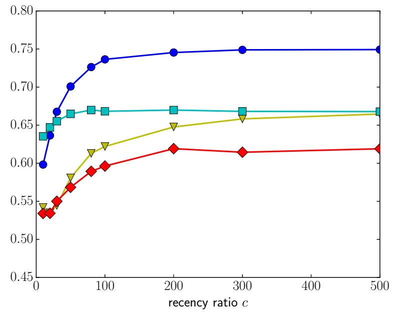
Furthermore, we testify the overall performances of all the methods in both two datasets, and the results are shown in Figure 3.

For WA dataset, we let the baseline methods run for a similar duration as our method, i.e. follow the ST setting. We can see that our D-SGD significantly and consistently outperforms all other baselines. Adam-ST-GL, the best method with global weight decay strategy(GL) in our experiment has a better performance than static weighted models. It has a improvement of 0.161 (with relative improvement of 28.1%) than the best static weighted model which is Adam-SW in this dataset . Our method(D-SGD) performs much better and more stable than Adam-ST-GW in this dataset, and get a improvement of 0.197 (with relative improvement of 34.3%). We also testify SGD and Adam methods in BRT mode, where we let these methods converge every time. However, their performances are not satisfactory. SGD-BRT-SW performs similarly as Adam-ST-GW and much worse than D-SGD, but it takes much longer running time than Adam-ST-GW and D-SGD.

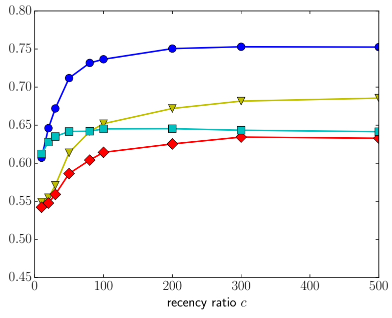
Compared with WA dataset, WF dataset in Figure 3e shows less recency and dynamic characteristic. The edges there are considered to be effective for a much longer time, and the updating speed



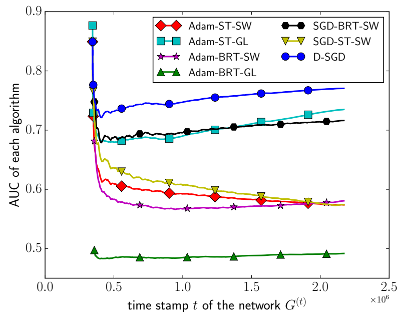
(a) AUC when $\max r = 3$



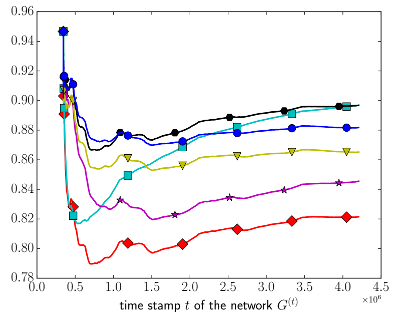
(b) AUC when $\max r = 6$



(c) AUC when $\max r = 9$



(d) AUC for WeChat article data



(e) AUC for WeChat friend network

Figure 3: The AUC result for our datasets

of edges are not very fast. Still, D-SGD achieves satisfactory performances. The weight decay strategy and the weight diffusion mechanism still works well in all baselines. As mentioned the characteristics of this dataset are not very consistent with our targeting scenarios. In such a case, it is not strange that SGD-BRT-SW performs slightly better than D-SGD, because it take much longer time for optimization and reaches convergence at each timestamp. As well as the Adam-ST-GL, it performs much worse than the D-SGD when data is highly dynamic in the early stage. It also costs much longer time when the timestame grows up, which will be shown in the next subsection. Although not the best, the performance of D-SGD is still stable and reliable, demonstrating the wide applicability of D-SGD.

These results fully demonstrate that D-SGD, especially the weight diffusion mechanism in it, can effectively address the challenges brought by highly-dynamic and recency-sensitive scenarios.

5.3 Efficiency

Here we design experiments to evaluate the efficiency of our method and baselines. We run all of the tests on CPU Intel(R) Xeon(R) CPU E5-2630 whose main frequency is 2.30GHz without any parallel mechanism.

To make a fair comparison, we compare the running time of each method when they reach the same AUC performance. Here we set the target AUC to be 0.73. For different timestamp t, we calculate how much time will cost for each method to train the following 10^3 edges and reach the target AUC. Adam-SW and Adam-GW have never reached to 0.73, so we report the running time of these two methods when they reach their best AUC. Table 2 shows the result.

Table 2: Execution time of training 10^3 edges for each method(seconds)

timestamp t	D-SGD	D-SGD-WT	SGD-SW	Adam-SW	Adam-GL
1×10^0	0.0234	0.134	0.231	1.84	1.58
1×10^3	0.0307	0.147	0.238	1.79	1.54
3×10^3	0.0295	0.169	0.261	1.82	1.60
1×10^4	0.0347	0.255	0.279	1.88	1.82
3×10^4	0.0336	0.469	0.423	1.98	2.44
1×10^5	0.0441	1.20	0.388	2.15	4.63
3×10^5	0.0568	3.61	0.498	2.39	10.9
1×10^6	0.0739	15.1	0.668	2.77	32.6
3×10^6	0.0664	45.9	0.684	2.87	96.2

From this table, we can find that D-SGD is much more efficient than the other baselines. Though Adam-GL is a good methods sometimes, its time cost for each edge is $O(t + d \max r)$(where d is the dimension of the embedding and r is the number of iteration steps) which is not practical in real applications. We also compared the D-SGD without Nested Segment Trees(short for D-SGD-WT). Without this optimization, the time cost for each edge is also $O(t + d \max r)$. Other two methods also cost several times more than D-SGD, even their performance much worse than D-SGD.

6 CONCLUSIONS AND FUTURE WORK

We discover that the timeliness is widely existing in recency networks, ans we propose to use it to speedup the representation learning on a recency network. Furthermore, we find that the embedding has a diffusion characteristic. To speedup the learning method for practical usage, we applied this characteristic and propose a general framework for embedding learning on a recency network. We proved that our framework has a boundary and convergence when the loss function satisfies some conditions, and the experiment shows this framework have a good performance on both effect and efficiency in realistic datasets.

To deal with highly-dynamic and recency-sensitive data, we propose an optimization method Diffused Stochastic Gradient Descent. In calculating gradients of this method samples are selected according to their weights, while the weights are maintained according to the recency-sensitive weights. The related samples are also updated in a diffusion strategy after updating the embedding of the selected sample.

To improve the efficiency of this progress, we propose a Nested Segment Tree. We also prove the convergence rate theoretically for independent data samples and prove the efficacy in large-scale real datasets.

In the future, we are to parallelize this framework since the components of it is able to be parallelized. As well, we are going to achieve this framework on a production environment.

ACKNOWLEDGMENTS

This work was supported in part by National Program on Key Basic Research Project No. 2015CB352300, National Natural Science Foundation of China Major Project No. U1611461; National Natural Science Foundation of China No. 61772304, 61521002, and 61531006. Thanks for the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the Young Elite Scientist Sponsorship Program by CAST. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Rehab H Alwan, Fadhil J Kadhim, and Ahmad T Al-Taani. 2005. Data embedding based on better use of bits in image pixels. *International Journal of signal processing* 2, 2 (2005), 104–107.

[2] Robert Bamler and Stephan Mandt. 2017. Dynamic word embeddings. In *International Conference on Machine Learning*. 380–389.

[3] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 177–186.

[4] Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. 2017. Deep coevolutionary network: Embedding user and item features for recommendation. (2017).

[5] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. 2014. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*. 1646–1654.

[6] Yoav Goldberg and Omer Levy. 2014. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* (2014).

[7] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[8] Elad Hazan et al. 2016. Introduction to online convex optimization. *Foundations and Trends® in Optimization* 2, 3-4 (2016), 157–325.

[9] Elad Hazan, Alexander Rakhlin, and Peter L Bartlett. 2008. Adaptive online gradient descent. In *Advances in Neural Information Processing Systems*. 65–72.

[10] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[11] Jey Han Lau and Timothy Baldwin. 2016. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368* (2016).

[12] Deanna Needell, Rachel Ward, and Nati Srebro. 2014. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems*. 1017–1025.

[13] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1933–1942.

[14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[15] Jan Snyman. 2005. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*. Vol. 97. Springer Science &amp; Business Media.

[16] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[17] Zijun Yao, Yifan Sun, Weicong Ding, Nikhil Rao, and Hui Xiong. 2018. Dynamic Word Embeddings for Evolving Semantic Discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 673–681.