Improving Dynamic Recommendation using Network Embedding for Context Inference

Thilina Thanthriwatta*, David S. Rosenblum*†
*Department of Computer Science, National University of Singapore, Singapore
{thilinat, david}@comp.nus.edu.sg
†Department of Computer Science, George Mason University, VA, USA
dsr@gmu.edu

***Abstract*—Network embedding, which is a method to learn low-dimensional latent representations of nodes in networks, can be effectively utilized to infer contexts in the context-aware recommender domain. One of the fundamental challenges of network embedding is how to effectively and efficiently learn embeddings from dynamic networks, whose nodes and edges change over time. Network embedding approaches designed for static networks are infeasible to use with dynamic networks for reasons of scalability. The use of network embedding for inferring contexts in the incremental recommender task poses two fundamental challenges: (1) efficiently inferring contextual information that changes over time; and (2) integrating learned contextual features with a recommender technique that can be updated incrementally. To address these challenges, we present a neural recommender approach that models user interactions in the dynamic setting. Furthermore, we introduce a novel dynamic network embedding method based on an efficient neighborhood sampling technique, which employs a temporally biased form of random walk. We have successfully applied our approach to Point-Of-Interest recommendation domain by improving efficiency in context inference and quality of recommendations.**

***Index Terms*—Context-Aware Recommender Systems, Dynamic recommender environments, Context inference, Network embedding**

I. Introduction

In contrast to conventional recommender systems, which only utilize information about users and items, Context-Aware Recommender Systems (CARSs) incorporate contextual information and make the recommendation problem multi-dimensional. Producing hand-engineered features leads to high computational costs. Hence, it is important to *efficiently* infer contextual features from dynamic recommender environments. Network embedding, which is a promising technique for learning latent representations of nodes based on the structural properties of networks [1], has been used to infer contextual features for recommendations that are based on the static setting [2]. However, we identified two main limitations of current work that integrated network embedding as a context inferring mechanism with the recommender task.

Modelling dynamic user preferences. Prior studies proposed online update mechanisms with different linear factorization methods such as Matrix Factorization (MF) and Factorization Machines (FMs) [3]–[6]. The approaches that integrate network embedding with the (context-aware) recommender setting typically rely on the batch learning (as in the static setting) [2], [7]. In such systems, after deployment, the learned model must be fully re-trained from scratch to provide recommendations to previously unseen users. Moreover, without re-training, recommender systems tend to provide suboptimal recommendations for the users whose preferences are substantially changed.

Learning representations from evolving networks. Most of the real-world networks are dynamic as they change their structures over time. Especially, networks tend to evolve due to the addition of new nodes and edges (e.g., user-item bipartite graphs). Static network embedding methods, such as node2vec [1], can be fully trained using extensive neighborhood sampling techniques and the use of a huge amount of data; however, these approaches have to be fully re-trained for each snapshot of a dynamic network, and re-training induces computationally-prohibitive time and space costs. This issue motivated practitioners to construct network embedding methods that are suitable for learning representations of the nodes in dynamically evolving networks. Despite successful attempts to implement dynamic network embedding approaches [8]–[10], such efforts have focused on traditional application domains of network embedding (such as vertex classification) and have overlooked the application of network embedding for recommender systems.

To answer these questions, we have devised a dynamic network embedding method based on *efficient* neighborhood sampling. In order to make network embedding fast, we define *perturbed nodes*, which represent the difference between consecutive graph snapshots, to focus on most discriminative nodes that changed the network. We believe that this node selection mechanism is important to enhance the efficiency of the embedding process. We therefore present a biased random walk process based on exponential decay to identify the neighborhood of nodes, while allowing the random walker to explore perturbed nodes with higher transition probabilities. To show the feasibility of our method, we consider Point-of-Interest (POI) recommendation as the application domain and utilize spatiotemporal aspects of user check-in behaviors as contextual information. We believe that our method can be generally applicable to any recommender domain with minimal modifications and domain knowledge (e.g., inferring social contexts from a dynamic trust network). We dynamically generate a network using the spatiotemporal relation-

ships among POIs ("spots"). Our dynamic network embedding method is applied to the network to infer spatiotemporal contexts (in the forms of dense latent vectors) of POIs. Inspired by FMs [11] and Neural Factorization Machines (NFMs) [12], we present a neural architecture to combine these dense contextual vectors with highly sparse recommendation inputs and allow the recommender system to perform incremental updates in the dynamic setting. We employ the *test-then-learn* evaluation in this study. In contrast to the prequential evaluation [5] where the model updates are performed after providing recommendations to each incoming user interaction (i.e., per-interaction updating), we refresh the model incrementally on *blocks* of incoming interactions (i.e., block-based updating). We use the term "incremental recommender systems" for systems using block-based updating, to avoid confusion with the term "online recommender systems", which implies per-interaction updating.

Contributions. To the best of our knowledge, this is the first attempt to develop an incremental context-aware recommender approach, which is evaluated by the *test-then-learn* strategy, by utilizing dynamic network embedding to model ever-changing spatiotemporal contexts of user check-in behaviors. We conducted extensive experiments on two real-world check-in datasets and demonstrated the superior performance of our approach in terms of prediction accuracy and efficiency, over other baselines.

II. Our approach

We now present DNE−CR, which consists of two main components: (1) DNE−CR−N, a dynamic network embedding that infers contextual information from evolving networks; and (2) DNE−CR−R, an integration of the contextual information with the neural recommender system that is updated incrementally. We denote the set of users and items (POIs) as $\mathcal{U}$ and $\mathcal{L}$, respectively. Let $\mathcal{P}$ be the stream of a chronologically-ordered sequence of user interactions (check-ins). A check-in can be defined as a tuple $< u, l, t' >$ that indicates user $u \in \mathcal{U}$ visiting $l \in \mathcal{L}$ at time t'. Let t_1 and t_2 be two time points ($t_2 > t_1$). Δb, which is a part of $\mathcal{P}$, denotes the block of user interactions (check-ins) that happened (made) between t_1 and t_2. We first provide recommendations for incoming users in Δb based on the already learned recommender model at t_1. Note that the users are presented with recommendations as they interact with the system in real-time. The users provide feedback (i.e., check-ins) for the presented rankings of POIs.

For an incremental update at t_2, firstly, a new network snapshot is constructed based on Δb to update contextual information (i.e., the embeddings of the spatiotemporal contexts of the POIs appeared in Δb). Then, a minimal update is performed on the recommender model using Δb while integrating the updated contextual information to refresh its parameters at t_2. Our method runs over a sequence of data blocks indexed by $t \in T$ where T is the total number of data blocks in the recommender environments. We denote t-th data block as $\mathcal{D}(t)$ in the rest of this paper.

A. Constructing Dynamic Networks

We define a dynamic network as follows:

Definition 1. *Dynamic Network*. A *dynamic network* is a sequence of discrete snapshots of directed (or undirected) networks G_1, G_2, ..., G_T. Let $G_t = (V_t, E_t)$ be the network at time t ($1 \leq t \leq T$). V_t and E_t denote the set of nodes and the set of edges of G_t, respectively. Each edge of G_t is associated with a non-negative weight value.

Modeling node embedding as a maximum likelihood optimization problem, we choose the Skip-Gram architecture [13], which achieves superior performance in the network embedding task [8]. To differentiate nodes, we define the notion of perturbed nodes in the dynamic setting as follows:

Definition 2. *Perturbed Nodes*. A node $v_t \in V_t$ that fulfills one of the following conditions is defined to be a *perturbed node*.

- Condition 1. $v_t \notin V_{t-1}$. Note that V_{t-1} is the set of nodes of the previous network snapshot, G_{t-1}.
- Condition 2. $v_t \in V_{t-1} \wedge v_t\bar{v} \in E_t$ where $\bar{v}$ fulfills Condition 1 (i.e., $\bar{v} \in V_t \wedge \bar{v} \notin V_{t-1}$). $v_t\bar{v}$ denotes the edge between nodes v_t and $\bar{v}$.
- Condition 3. $v_t \in V_{t-1} \wedge v_t \in V_N(v')$ where $V_N(v')$ denotes the neighbors of node v'. v' should satisfy Condition 2.
- Condition 4. $v_tv' \in E_{t-1}$ and if a new edge between nodes v_t and v' occurs in G_t.
- Condition 5. $v_t\hat{v} \in E_{t-1} \wedge \hat{v} \in V_{t-1} \wedge \hat{v} \notin V_t$.
- Condition 6. $v_t\hat{v} \in E_{t-1} \wedge v_t\hat{v} \notin E_t$.
- Condition 7. $v_t \in V^I$ where the set V^I contains $\kappa\%$ of most influential nodes sampled from the nodes that do not satisfy any other condition.

According to Definition 2, we capture the perturbations of the network that occur from time $t-1$ to time t as the newly added and deleted nodes, plus the existing nodes (and their neighbors) to which the added and deleted nodes are/were connected. In this study, we only consider dynamically evolving networks that satisfy Conditions 1, 2, 3, 4 and 7, and Condition 5 and 6 are included merely for completeness of the definition. We do not consider multigraphs, and the scenario where a new edge occurs between the same pair of nodes by **replacing** an existing one is investigated under Condition 4. We enrich the pool of perturbed nodes by including influential nodes of the network snapshot, which do not satisfy other conditions. Degree centrality, which has a linear time complexity with respect to the number of nodes of a graph, is used to determine node influence.

For every perturbed node $v_t \in V_t$, we identify a pool of neighbor nodes $N(v_t) \subset V_t$ using random walks based on neighborhood sampling. We disregard the conventional approach of using all nodes in a network as source nodes to initiate random walks, as it poses an issue of scalability for the commonly used state-of-the-art node embedding approaches [1] in the dynamic setting.

Dynamic POI-POI Network. To learn spatiotemporal contexts via network embedding, first, we have to construct a dynamic network that illustrates ties among POIs by preserving spatiotemporal homophily. Inspired by Xie et al. [14], we define dynamic POI-POI network as follows:

Definition 3. *Dynamic POI-POI network*. Let P_t be the set of POIs that appears in t-th data block ($\mathcal{D}(t)$). For each pair $p \in P_t$ and $\bar{p} \in P_t$ of POIs that are successively checked in by user $u \in n_t$, a directed edge from p to $\bar{p}$ is added to t-th snapshot of POI-POI network G_t^P by representing u's transition from p to $\bar{p}$. Let n_t be the set of users in $\mathcal{D}(t)$.

To carry out neighborhood sampling, we deploy a biased random walk specifically respecting the temporal aspect and maximizing the probability of exploring more perturbed nodes instead of allowing the walker to loiter around nodes in which structures have not changed much. As our observations based on the check-ins made around in the Los Angeles area, users tend to check-in to a nearby POI within a short period of time (e.g., the next 24 hours). We found that 71.7% of successive check-ins are made within the next 24 hours. Moreover, 39.2% of successive check-ins are made within the next 24 hours and for a different POI situated within 2 km. Moreover, users tend to show a recurrent nature in check-in behaviors. For example, once a user visits a POI, she tends to visit the same set of nearby POIs on different occasions. By considering these observations, we define the weight w^p of the edge between node a and $\bar{a}$ as follows:

$$w^p_{a\bar{a}} := \frac{\vartheta e^{-\gamma(\Phi-1)}}{\Delta d \times \Delta \bar{t}}\,, \tag{1}$$

where the edge from a to $\bar{a}$ has continuously appeared in Φ number of network snapshots, where $\Phi > 0$, without changing ϑ. ϑ is the number of times the edge (a, $\bar{a}$) occurred, Δd denotes the geographical distance (measured in kilometers) between two POIs a and $\bar{a}$, and $\Delta \bar{t}$ is the average of time differences (measured in minutes) between successive check-ins which correspond to a and $\bar{a}$. Every time the same directed edge (a, $\bar{a}$) occurs the value of ϑ is incremented by 1, and the value $\Delta \bar{t}$ can change. $\gamma > 0$ is the decay constant. Due to the use of γ, the weight of each of the unchanged edges decreases over time, and this encourages the random walker to explore the nodes that are recently added and modified instead of visiting unchanged nodes repeatedly. In the first network snapshot where the edge (a, $\bar{a}$) occurs, Φ is set to 1. Note that, according to Condition 4 of Definition 2, an existing edge can be replaced with the same edge (i.e., recurring edges) but with a different weight due to the increment of ϑ. Such edges are considered as new edges, and Φ value of those will reset to 1.

Given a latent vector of a perturbed node v_t of G_t^P, the log-probability of observing $N(v_t)$ is maximized by the following optimization formula:

$$\max_f \sum_{v_t \in V_t^p} \log Pr(N(v_t)|f(v_t))\,, \tag{2}$$

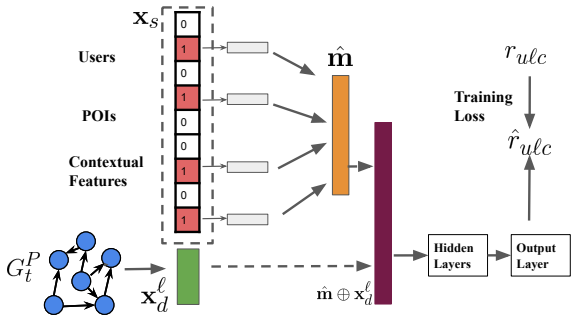

Fig. 1: Model Architecture.

where V_t^p is the set of perturbed nodes in G_t^P and $f(v_t) \rightarrow \mathbb{R}^d$. d is the dimension of the latent vector (i.e., embedding) of node v_t. We assume the standard conditional independence among the nodes of a neighborhood given the latent vector of v_t as follows:

$$Pr(N(v_t)|f(v_t)) = \prod_{n_i \in N(v_t)} Pr(n_i|f(v_t))\,. \tag{3}$$

We terminate a random walk starting from v once all the nodes of ego-graph of v with distance 2 (2-hop neighborhood) have been visited. If the random walker cannot visit all nodes of the ego-graph, then it loiters up to the maximum walk length. We use Stochastic Gradient method with negative sampling to learn optimized model parameters in the network embedding using Eq. 2.

B. Integration of Spatiotemporal Contexts with a Neural Recommender Model

Figure 1 depicts the neural network architecture of `DNE-CR-R`. We describe how an individual instance is used for training. The input layer consists of two components of the feature vector—sparse and dense. Let $\mathbf{x}_s \in \mathbb{R}^{\bar{n}}$ denote the sparse component of the feature vector, where $\bar{n}$ is the number of features values. It encodes information of users and POIs, such as user IDs, POI IDs and other contextual information (e.g., *times of the day*). One-hot encoding [11] is used for the encoding of $\mathbf{x}_s$. The dense component $\mathbf{x}_d^\ell \in \mathbb{R}^d$ is the d-dimensional spatiotemporal context embedding of POI ℓ. Note that ℓ is the POI encoded in $\mathbf{x}_s$.

In the embedding layer, we multiply i-th element of $\mathbf{x}_s$ by a k-dimensional corresponding embedding $\mathbf{e}_i \in \mathbb{R}^k$. Note that, due to the sparse nature of $\mathbf{x}_s$ and the multiplication property of zero, the elements with zero can be omitted, and we only consider the embeddings of the non-zero elements (i.e., feature values) in $\mathbf{x}_s$. Inspired by FMs [11] and NFMs [12], we aggregate these embeddings by considering their second-order interactions. Thus, each pair of embeddings $\mathbf{e}_i$, $\mathbf{e}_j$ will be used to compute another vector $\mathbf{e}_{ij} = (\mathbf{e}_i \odot \mathbf{e}_j)_k$. Finally, the embeddings of all pairs are summed together to obtain $\hat{\mathbf{m}} \in \mathbb{R}^k$.

In the fusion layer, we concatenate $\hat{\mathbf{m}}$ and $\mathbf{x}_d^\ell$. There are two reasons for selecting the concatenation operation: (1) its ability to preserve information and (2) its ability to combine

two vector spaces without using additional model parameters. To model higher-order interactions, we use a stack of hidden layers as follows:

$$\begin{aligned} \mathbf{Z}_1 &= \phi_1(\mathbf{W}_1[\hat{\mathbf{m}} \oplus \mathbf{x}_d^\ell] + \mathbf{b}_1) \\ &\cdots \\ \mathbf{Z}_N &= \phi_N(\mathbf{W}_N \mathbf{Z}_{N-1} + \mathbf{b}_N)\,, \end{aligned} \tag{4}$$

where N denotes the number of hidden layers, and ϕ_i, $\mathbf{W}_i$, and $\mathbf{b}_i$ represent the activation function, weight matrix, and bias vector of layer i, respectively. As the activation function, we use the Rectified Linear Unit (ReLU) function, which facilitates modeling the non-linear relationships of user interactions and provides faster convergence. Finally, the prediction score $\hat{r}_{u\ell c}$ for user u for item (POI) ℓ under context c (in other words, for the input feature vector that contains both $\mathbf{x}_s$ and $\mathbf{x}_d^\ell$) is computed as:

$$\hat{r}_{u\ell c} = \sum_{i=1}^{\bar{n}} \mathbf{w}_i \mathbf{x}_s^i + \mathbf{h}^T \mathbf{Z}_N\,, \tag{5}$$

where $\mathbf{h}$ is the weights of the prediction layer. $\sum_{i=1}^{\bar{n}} \mathbf{w}_i \mathbf{x}_s^i$ models the weights of the sparse feature values. $\mathbf{x}_s^i$ and $\mathbf{w}_i$ denote the i-th feature value of $\mathbf{x}_s$ and its weight, respectively. We then transform the predicted value using a Sigmoid function. We define the objective function L based on log loss as follows:

$$L = -\sum_{(u,\ell,c)\in\Omega} \left[r_{u\ell c} \log \hat{r}_{u\ell c} + (1 - r_{u\ell c}) \log(1 - \hat{r}_{u\ell c}) \right], \tag{6}$$

where Ω denotes the observed instances and $r_{u\ell c}$ is the actual rating (check-in) given by user u for item ℓ under context c. We use Adagrad [15] as our optimizer since it provides a faster convergence over the vanilla Stochastic Gradient Descent. To alleviate the overfitting problem, L_2 regularization is used.

C. Incremental Updates

Our aim is to first provide recommendations for incoming user interactions (i.e., instances) in the current data block $D(t)$. After that, we generate a new network snapshot and update spatiotemporal contexts of POIs that have been observed in $D(t)$. In the update process of node embeddings, we initialize the Skip-Gram model by the pre-trained Skip-Gram model (i.e., the Skip-Gram model produced at the previous time step). Based on the user feedback on the recommendations provided earlier, we update the model parameters of `DNE-CR-R`. In general, a block of new interactions should not drastically change the learned parameters. This allows us to set a small number of iterations (i.e., we use two iterations) as the stopping criterion of the optimization during incremental updates. Note that *only* the embeddings of perturbed nodes are updated in the dynamic setting. Let $\mathbf{B'}_{\bar{p}}$ be the spatiotemporal context of POI $\bar{p} \in V_t^p$ inferred based on G_t^P. We define it as:

$$\mathbf{B'}_{\bar{p}} = \begin{cases} \mathbf{B'}_{\bar{p}}, & \text{if } \bar{p} \text{ is a new node} \\ \theta \mathbf{B'}_{\bar{p}} + (1-\theta)\mathbf{B'}_{\bar{p}\bar{r}}, & \text{otherwise}\,, \end{cases} \tag{7}$$

where $\mathbf{B'}_{\bar{p}\bar{r}}$ is the existing latent vector (i.e., the most updated latent vector, which is computed by previous graph snapshots) of $\bar{p}$. The recency weight $\theta \in [0, 1]$ determines how much prior information should be propagated to the computation of the new latent vectors of existing POIs.

III. Experiments

Datasets. We used a publicly available check-in dataset, Gowalla [16][1]. From this dataset, we extracted check-ins made in two different cities to constitute two new datasets for our experiments. The **NYG** and **LAG** datasets contain the check-ins made in the New York city area[2] and the Los Angeles area[3], respectively. The NYG and LAG contain the check-ins made until 2010-05-07. For our experiments, we extracted three additional temporal features from the datasets as follows: *Times of the day*, *Days of the week*, and *WeekdayEnd*. *Times of the day* contains five values-Wee hours (00.00-05.59), Morning (6.00-10.59), Noon (11.00-13.59), Afternoon (14.00-17.59), and Evening (18.00-23.59). The feature *Days of the week* contains seven days as values. *WeekdayEnd* indicates whether check-ins made during weekday or weekend.

Experimental Setting. We chronologically split the original check-in datasets into train and test subsets. The test sets of both NYG and LAG datasets contain check-ins made during the last 9 weeks. We constructed 9 testing blocks based on the check-ins made during each week. From the remaining training sets, we further filtered out the validation sets, which are used for tuning hyperparameters. The validation sets consist of check-ins made within a 3-week period of time. The remaining check-ins were used as the train data for the initial training.

We simulated the dynamic recommender environment to incrementally evaluate our approach by generating network snapshots with the arrival of test data blocks. Since the datasets contain only positive implicit ratings (i.e., user check-ins), a uniform negative sampling method was used for the initial training and subsequent model updates. For each positive instance with the target value as 1, we randomly created two negative instances with POIs which had not been checked-in by the user and assigned 0 as their target values. In the dynamic setting, we provided recommendations only for incoming positive instances. The processed NYG contains 172,164 instances given by 3,356 users for 13,686 POIs, and the LAG, which is the more sparse dataset, consists of 225,795 instances given by 3,420 users for 19,329 POIs. Inspired by the truncating strategy of ranked lists [6], we paired each testing instance with 49 negative POIs which had not been checked-in by the user (the user in the incoming testing instance) previously, and each method predicted scores for 50 POIs-meaning that the ranked lists were truncated at 50. We adopted two ranking metrics, *Hit Ratio* (HR) and *Normalized Discounted Cumulative Gain* (NDCG) [17] for the evaluation. The results are averaged across all positive test instances. Note that higher values for these metrics indicate better performance.

[1] https://snap.stanford.edu/data/loc-gowalla.html

[2] 40.5 $\leq$ latitude $\leq$ 41.0, -74.5 $\leq$ longitude $\leq$ -73.5

[3] 33.2 $\leq$ latitude $\leq$ 34.2, -118.5 $\leq$ longitude $\leq$ -117.5

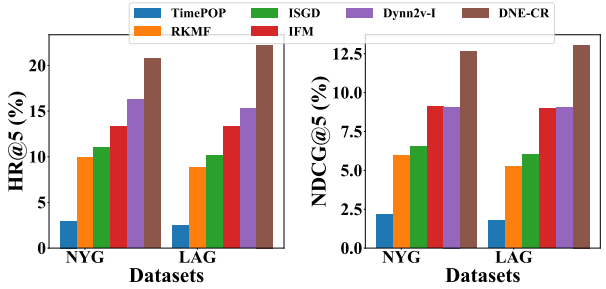

Fig. 2: Prediction accuracy.

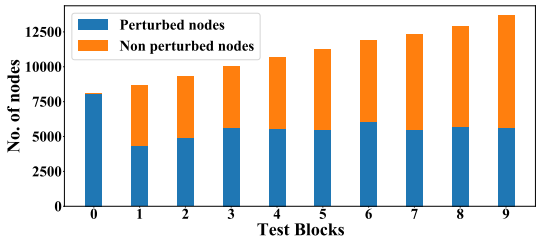

Fig. 3: Evolution of G_t^P on the NYG.

Baselines. We compare DNE−CR against the following baselines: (a) **TimePOP**: This is a non-personalized baseline that recommends the most popular items during a given time interval for incoming users. We set seven days as the time interval; (b) **RKMF**: This approach integrates an online update mechanism with MF [4]. We selected the linear kernel and set m to the default value of 50; (c) **ISGD**: ISGD is an incremental MF approach that was proposed to handle the dynamic nature of the recommender environments using implicit ratings [5]; (d) **IFM**: Based on the incremental version of FMs [3], we implemented this linear model by providing the temporal contextual features such as *Times of the day* along with the identifiers of the users and the POIs as the features. Note that this baseline does not use the spatiotemporal contexts inferred by DNE−CR−N; and (e) **Dynn2v−I**: Due to the absence of methods that integrate dynamic network embedding and the incremental recommender task, we constructed a variant of our method by substituting DNE−CR−N with the dynamic node2vec method proposed by Mahdavi et al. [18].

Parameter Setting. We set the learning rate to 0.001 for all the recommender models except TimePOP. Note that TimePOP does not consist of model learning. The L_2 regularization parameter is tuned in $\{0.5, 0.1, 0.01, 0.001\}$. k is fixed at 32. The feature embeddings of other methods (e.g., user/item embeddings of RKMF) are also set with an embedding size of 32. We empirically set d to 32. γ is set to 1.5 on both NYG and LAG datasets. θ is set to 0.4 and 0.2 on the NYG and LAG, respectively. For the neural network, we use a hidden layer with 64 hidden units and initially train the neural model with a batch size of 512. The models were trained for 10 epochs during initial batch training except for ISGD and TimePOP. p and q of the node2vec model in Dynn2v−I are set 4 and 1, respectively. We fix window, iter, worker (hyperparameters of the Skip-Gram model) to 10, 1, and 1, respectively for all the network embedding models. We set κ to 10 and 30 on the NYG and LAG, respectively.

A. Performance Comparison

Figure 2 presents the performance of the baselines. We report the average HR and NDCG values (TOP-5) of the testing blocks. DNE−CR outperforms other approaches significantly on both datasets and improves over the strongest baselines w.r.t. HR@5 by 27.4% and 45.3% on the NYG and LAG, respectively. The performance of DNE−CR is statistically significant ($p < 0.01$ using the two-tailed paired t-test) compared to the best baseline. We believe that the use of non-linear neural architecture that effectively models higher-order feature interactions has a significant impact on this performance improvement over other linear models, such as RKMF, ISGD and IFM. Furthermore, this highlights the positive effect of adapting and modeling dynamic changes of user preferences and contexts in the recommender setting via incremental updates. Another implication of this result is the superior performance of the neighborhood sampling and the biased random walk of DNE−CR−N against Dynn2v−I.

The poor performance of TimePOP indicates its inability to produce personalized results for incoming users. As expected, TimePOP struggles to produce quality recommendations in a highly sparse recommender domain with a large number of items. RKMF and ISGD, which are extensions of MF, show similar behavior. Both baselines utilize users and POIs only. The performance improvements of IFM, Dynn2v−I and DNE−CR over RKMF and ISGD indicate the importance of utilizing contextual features to improve the prediction accuracy. We observe a slight performance degradation of DNE−CR over time. One of the possible reasons for this phenomenon would be the obsoletion of the hyperparameters. We will leave the incremental-tuning of hyperparameter as a future work.

Ablation Study. We perform an ablation study to understand how the different conditions of Definition 2 affect the overall performance of DNE−CR. Performance degradation can be observed when we randomly remove 50% of the identified perturbed nodes of the network snapshots, especially in the case where the perturbed nodes of the first snapshot, which is generated by the train data, are removed. Note that the removed nodes may contain the new nodes and/or the existing nodes that are important for the incremental updates. Moreover, we observe that it is essential to capture new nodes as perturbed nodes.

Efficiency. Let V_p^t be the number of perturbed nodes of t-th network snapshot ($t = 1...T$). T is the total number of network snapshots, and V_p^1 is the total number of nodes in the network snapshot used in the initial training. The time complexity of DNE−CR−N over T is $\mathcal{O}(V_p^1 \log V + \sum_{t=2}^{T} V_p^t \log V)$, where V is the total number of nodes of a network snapshot. As can be seen in Figure 3, the number of perturbed nodes (blue color) is considerably smaller than the total number of nodes of a graph snapshot (except the first network snapshot, which has been constructed during the initial training (i.e., 0-th test

block)). We thus highlight the suitability of DNE−CR−N due to its higher scalability. We observe that the embedding time (the wall-clock time) taken by DNE−CR−N is stable over all the snapshots.

Sensitivity of Parameters. We observe that the prediction accuracy improves with the increase of d. However, the accuracy degrades when d is at 64. The exponential Decay γ is not highly sensitive on the LAG dataset; however, a slight increase of NDCG@5 value can be observed on the NYG dataset. Recency weight θ also shows a similar characteristic, and we believe that the use of a pre-trained Skip-Gram for the initialization has a significant impact on reducing the sensitivity of these hyperparameters. Setting higher values for κ degrades the performance on the NYG dataset, while slightly enhancing the performance on the LAG dataset. The use of a small number of iterations (e.g., 2 iterations) in the incremental update process is sufficient to update/refresh model parameters, and the over-training does not lead to a significant performance boost. We conclude that the over-training in the dynamic setting does not necessarily increase the performance, and it increases the time for model updating.

IV. Related Work

Li et al. highlighted the importance of handling the dynamic nature of networks [9]. Nguyen et al. proposed a continuous-time dynamic network embedding approach [10] by exploring the use of a continuous temporal aspect in network embedding. Mahdavi et al. [18] proposed an embedding method to learn embeddings of nodes that are affected due to the newly added nodes and edges of a dynamic network. Their approach outperforms several network embedding methods in link prediction and node classification. There are several efforts in the literature that integrate network embedding with the recommender task using the meta-path based random walk strategy [19] and node2vec [2]. Recently, Wen et al. [2] proposed a social recommender system by integrating MF and node2vec. It is worth to mention that the integration of network embedding on new application domains such as Co-purchaser recommendation [20] and Next POI recommendation [14] inspired us developing DNE−CR. However, these methods do not consider the incremental recommender task, which should be evaluated by the *test-then-learn* method.

V. Conclusion

Our solution shows the importance of utilizing an efficient network embedding approach to infer contexts in incremental context-aware recommendations. In order to learn contextual features via dynamic networks, we apply the notion of perturbed nodes to discriminate nodes based on their changed characteristics. Furthermore, we introduce a biased random walk to sample neighborhoods in a way that preserves spatial and temporal aspects. Our method outperforms other baselines in terms of the quality of recommendations and efficiency. In the future, we expect to extend the capabilities of dynamic network embedding by mining complex structural properties of dynamic networks via deep neural networks.

References

[1] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[2] Y. Wen, L. Guo, Z. Chen, and J. Ma, "Network embedding based recommendation method in social networks," in *Companion of the The Web Conference 2018 on The Web Conference 2018*. International World Wide Web Conferences Steering Committee, 2018, pp. 11–12.

[3] T. Kitazawa, "Incremental factorization machines for persistently cold-starting online item recommendation," *arXiv preprint arXiv:1607.02858*, 2016.

[4] S. Rendle and L. Schmidt-Thieme, "Online-updating regularized kernel matrix factorization models for large-scale recommender systems," in *Proceedings of the 2008 ACM conference on Recommender systems*, 2008, pp. 251–258.

[5] J. Vinagre, A. M. Jorge, and J. Gama, "Fast incremental matrix factorization for recommendation with positive-only feedback," in *International Conference on User Modeling, Adaptation, and Personalization*. Springer, 2014, pp. 459–470.

[6] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua, "Fast matrix factorization for online recommendation with implicit feedback," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2016, pp. 549–558.

[7] L. Guo, H. Jiang, X. Liu, and C. Xing, "Network embedding-aware point-of-interest recommendation in location-based social networks," *Complexity*, vol. 2019, 2019.

[8] L. Du, Y. Wang, G. Song, Z. Lu, and J. Wang, "Dynamic network embedding: An extended approach for skip-gram based network embedding." in *IJCAI*, 2018, pp. 2086–2092.

[9] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 2017, pp. 387–396.

[10] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *3rd International Workshop on Learning Representations for Big Networks (WWW BigNet)*, 2018.

[11] S. Rendle, "Factorization machines," in *2010 IEEE International Conference on Data Mining*. IEEE, 2010, pp. 995–1000.

[12] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2017, pp. 355–364.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[14] M. Xie, H. Yin, F. Xu, H. Wang, and X. Zhou, "Graph-based metric embedding for next POI recommendation," in *International Conference on Web Information Systems Engineering*. Springer, 2016, pp. 207–222.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of machine learning research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[16] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 1082–1090.

[17] X. He, T. Chen, M.-Y. Kan, and X. Chen, "Trirank: Review-aware explainable recommendation by modeling aspects," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 1661–1670.

[18] S. Mahdavi, S. Khoshraftar, and A. An, "dynnode2vec: Scalable dynamic network embedding," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3762–3765.

[19] C. Shi, B. Hu, X. Zhao, and P. Yu, "Heterogeneous information network embedding for recommendation," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[20] J. Chen, W. Chen, J. Huang, J. Fang, Z. Li, A. Liu, and L. Zhao, "Co-purchaser recommendation based on network embedding," in *International Conference on Web Information Systems Engineering*. Springer, 2019, pp. 197–211.