PIANO: Influence Maximization Meets Deep Reinforcement Learning

[Technical Report]

ABSTRACT

Since its introduction in 2003, the influence maximization (IM) problem has drawn significant research attention in the literature. The aim of IM is to select a set of k users known as seed users who can influence the most individuals in the social network. The problem is proven to be NP-hard. The state-of-the-art algorithms estimate the expected influence of nodes based on sampled diffusion paths. As the number of required samples have been recently proven to be lower bounded by a particular threshold that presets tradeoff between the accuracy and efficiency, the result quality of these traditional solutions is hard to be further improved without sacrificing efficiency. In this paper, we present an orthogonal and novel paradigm to address the IM problem by leveraging deep reinforcement learning to estimate the expected influence. Specifically, we present a novel framework called PIANO that incorporates *network embedding* and *reinforcement learning* techniques to address this problem. Experimental study on real-world networks demonstrates that PIANO achieves the best performance w.r.t efficiency and influence spread quality compared to state-of-the-art classical solutions. We also demonstrate that the learned parametric models generalize well across different networks. Besides, we provide a pool of pretrained PIANO models such that any IM task can be addressed by directly applying a model from the pool without training over the targeted network.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:

. 2021. PIANO: Influence Maximization Meets Deep Reinforcement Learning: [Technical Report]. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/1122445.1122456

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Online social networks have become an important platform for people to share and disseminate information. Given the widespread usage of social media, individuals' perceptions, preferences and behavior are often influenced by their peers/friends in social networks. Since 2003, the *influence maximization* (IM) problem has been extensively studied to maximize diffusion of innovations and ideas in a network. The purpose of IM is to select a set of k seed nodes who can influence the most individuals in the network [21]. For instance, an advertiser may wish to send promotional material about a product to the k seed users of a social network that are likely to sway the largest number of users to buy the product.

A large number of greedy and heuristic-based IM solutions have been proposed in the literature to improve efficiency, scalability, or influence quality. State-of-the-art IM techniques attempt to generate $(1 - 1/e - \epsilon)$-approximate solutions with a smaller number of RIS (*Random Interleaved Sampling*) samples, which are mainly used to estimate the expected maximum influence (denoted as $\sigma(v, S)$) for an arbitrary node v given the current selected seeds S. They use sophisticated estimation methods to reduce the number of RIS samples closer to a theoretical threshold θ [39]. As θ provides a lower bound for the number of required RIS samples, these methods have to undertake a *diffusion sampling phase* and generate sufficient propagation samples in order to estimate the expected influence before selecting a seed. Despite the improvements of the sampling model and reduction in the number of required samples brought by recent studies, the cost of generating these samples is large especially in huge networks. Consequently, in this paper we ask the following question: *is it possible to avoid the diffusion sampling phase in IM solutions by utilizing a learned parametric function to estimate* $\sigma(v, S)$? We answer to this question affirmatively by proposing a novel framework that for the first time utilizes network embedding and reinforcement learning to tackle the IM problem.

The core challenge in IM lies in the process of estimating $\sigma(v, S)$ given v and the partial solution S, which is known to be #P-hard [21]. Traditional IM solutions address it by sampling the diffusion paths to generate an unbiased estimate for $\sigma(v, S)$. In essence, $\sigma(v, S)$ can be viewed as a mapping $\sigma : V \times G \times \Psi \rightarrow \mathbb{R}$, where $G(V, E)$ denotes the network and Ψ refers to the set of diffusion models, which defines the criteria to determine whether a node is influenced or not. That is, given a network G, an arbitrary node $v \in V$, and a particular diffusion model (*e.g.,* Independent Cascade, Linear Threshold model [21]), σ outputs the expected maximum number of influenced nodes by v. In this paper, we take a radically different approach where we view IM as a problem of finding the *optimal policy* to select the k-best seeds (i.e., k-best action sequence) in a deep reinforcement learning (RL) framework called PIANO (dee**P** re**I**nforcement le**A**rning-based i**N**fluence maximizati**O**n). Crucially, PIANO approximates σ as a (value) function $\tilde{\sigma}(v, S; \Theta)$ parameterised

by Θ and *learns* the values of the parameters Θ. It is worth mentioning that learning such mapping function is non-trivial and challenging. First, we need to transform the topology information of the target network into features. Second, there is no target expected maximum influence to supervise σ. Hence, supervised learning approaches cannot be adopted in this scenario. To address these challenges, PIANO seamlessly *integrates* RL [32] with network embedding [10] in an end-to-end deep RL framework.

The network embedding method considers the network topology to encode each node $v \in V$ into a feature vector, which acts as input to an RL algorithm to learn the value function $\tilde{\sigma}(v, S; \Theta)$ from rewards. The learned value function (more specifically, state-action value function) implicitly represents the learned optimal policy that can be applied to an unseen network for selecting k-best seeds. The learning-based framework makes the approach generalizable (*i.e.,* robust to small changes in the network) and scalable to large networks (*i.e.,* no need to compute all individual states and store them in memory). Moreover, we theoretically show that under our model the k seeds can be selected at one time, instead of one-after-another, which requires reevaluating $\sigma(v, S)$ for k times. Our exhaustive empirical studies demonstrate that once the mapping function is learned, it can be applied to other homogeneous networks with the same topological characteristic (*i.e.,* average degree).

The main contributions of the paper can be summarized as follows.

- We present a novel framework called PIANO that exploits learning methods to solve the classical IM problem. Specifically, a novel learning method is proposed to approximate $\sigma(v, S)$ as a parameterized value function $\tilde{\sigma}(v, S; \Theta)$, by exploiting model-free RL and deep learning for network embedding. PIANO generates seeds with superior running time to state-of-the-art machine learning-oblivious IM techniques without compromising on result quality. Specifically, it is up to 36 times faster than SSA [33]. Furthermore, our influence quality is slightly better than the traditional methods.
- We show how PIANO can be utilized to address the IM problem in large-scale networks even in the presence of evolution.
- We provide a pool of pretrained PIANO models such that any IM task can be addressed by directly applying a model from the pool without training over the targeted network. The facilitates generalization of the framework to different datasets.

The rest of this paper is organized as follows. Section 2 reviews related work. We formally present the learning-based IM problem in Section 3. We introduce the model training and seeds selection procedures in Sections 4 and 5, respectively. Experimental results are reported in Section 6. Finally, we conclude this work in Section 7.

2 RELATED WORK

In this section, we review research on influence maximization and network embedding.

2.1 Influence Maximization

Since the elegant work in [21], the IM problem has been studied extensively. Kempe *et al.* [21] proved that the problem of IM is NP-hard and provided a greedy algorithm that produces a $(1-1/e-\epsilon)$-approximate solution. Since then, series of works [28, 40] have been proposed to improve the response time while preserving the approximation guarantee. Besides these approximate algorithms, many excellent heuristic algorithms [9, 13, 19], which do not provide an approximation ratio, have been proposed to reduce running time further. Although these heuristic algorithms reduce the execution time by orders of magnitude, they sacrifice accuracy of the results significantly [4].

Since the introduction of *reverse influence sampling* (RIS) [6], which reversely samples a group of influence paths, a series of advanced approximate algorithms have been proposed, which can not only provide approximation guarantees, but also exhibit competitive running time compared to heuristic solutions. TIM/TIM+ [40] significantly improved the efficiency of [6] and is the first RIS-based approximate algorithm to achieve competitive efficiency with heuristic algorithms. Afterwards, IMM [39] and SSA/D-SSA [33] are presented, both of which have been recognized as the state-of-the-art solution that achieve the best efficiency [17]. Notably, none of these efforts integrate machine learning with the IM problem to further enhance the performance.

In recent years, there have been increasing efforts to address the IM problem utilizing learning methods. [3, 16, 30] have employed RL to find best strategy in competitive IM problem [8, 29]. Both of [3, 30] treat the competition between multiple parties as an adversarial game; and employ reinforcement learning to find the best policy (*i.e.,* strategy) that can maximize the profit given the opponents' choices. [41] uses deep learning to learn topic-aware influence in order to solve the Topic-Aware IM problem. In contrast, we propose a learning model for general IM problem and pre-trained model poor for practical applications without training from scratch. The authors in [16] employ deep learning to infer the entity correlations and influence cascading behavior. They do not natively model the influence estimation as a learning task and hence it is orthogonal to our problem. Besides, [7, 43] adopt learning methods to study the diffusion model and optimize the linear threshold model parameters, respectively. They do not consider IM as a machine learning problem and are also orthogonal to our problem. [42] studied seed selection in multiple communities. They converted their multi-community coverage maximization problem into a resource allocation problem. Instead of finding seeds from the given network, they used reinforcement learning to study how to assign selected seeds among multiple communities. Recently, [20] uses deep Q-learning to discover subgraph that can act as a surrogate to the entire network. They select the influential node set as the target set of the entire network by applying the traditional IM algorithm on the obtained subgraph, which is orthogonal to our strategy.

2.2 Network Embedding Methods

Early methods for learning node representations focused primarily on matrix decomposition, which was directly inspired by classical

techniques for dimensionality reduction [5]. However, these methods introduce a lot of computational cost. Recent approaches aim to learn the embedding of nodes based on random walks as word contexts. DeepWalk [34] was proposed as the first network embedding method using deep learning technology, which compensates for the gap between language modeling and network modeling by treating nodes as words and generating short random walks. LINE [1] uses the Breadth First Search strategy to generate context nodes, in which only nodes that are up to two hops from a given node are considered to be neighbors. In addition, it uses negative sampling to optimize the Skip-gram model compared to the layered softmax used in DeepWalk. node2vec [14] is a sequence extraction strategy optimized for random walks on DeepWalk framework. It introduces a biased random walk program that combines Breadth First Search and Depth First Search during neighborhood exploration. SDNE [44] captures the nonlinear dependency between nodes by maintaining the proximity between 2-hop neighbors through a deep autoencoder. It designs an objective function that describes both local and global network information, using a semi-supervised approach to fit optimization. There is also a kernel-based approach where the feature vectors of the graph come from various graphics kernels [36]. Structure2vec [10] models each structured data point as a latent variable model, then embeds the graphical model into the feature space, and uses the inner product in the embedded space to define the kernel, which can monitor the learning of the graphical structure. DeepInf [35] presents an end-to-end model to learn the probability of a user's action status conditioned on her local neighborhood.

3 PROBLEM STATEMENT AND DIFFUSION MODEL

In this section, we first formally present IM as a learning-based problem. Next, we briefly describe the information diffusion models discussed in these definitions.

3.1 Problem Statement

In this section, we first formally present IM as a learning-based problem. Next, we briefly describe the information diffusion models discussed in these definitions.

3.2 Problem Definition

Let $G = (V, E, W)$ be a social network, where V is the set of nodes, E is the set of edges, $|E| = m$, and $|V| = n$. $(u, v) \in E$ represents an edge from node u to node v. Let $W(u, v)$ denote the weight of the edge indicating the strength of the influence. Accordingly, the IM problem can be formally defined as follows.

Definition 3.1. (Influence Maximization) Given a social network $G = (V, E, W)$, an information diffusion model ψ, integer k, the influence maximization problem aims to select k nodes as the seed set S ($S \subseteq V$), such that, under the diffusion model ψ, the expected number of influenced nodes by S, namely $\sigma(S)$, is maximized. The problem can be formulated as $\text{argmax}\, \sigma(S)$ s.t. $|S| = k$.

As IM is proved to be NP-hard [21], all approximate solutions need to greedily select the next seed with the maximum marginal improvement in expected influence. In particular, let S_i be a partial solution with i seeds (*i.e.,* $i < k$), then in $(i+1)$-th iteration, an approximate algorithm shall choose a node v, such that $\sigma(S_{i+1}) - \sigma(S_i)$ is maximized, where $S_{i+1} = S_i \bigcup \{v\}$. To facilitate the following discussions, we refer to $\sigma(v, S)$ as the *maximum marginal expected influence* of v given a partial solution S. As $\sigma(v, S)$ is #P-hard to calculate based on v and S, traditional efforts in IM generate unbiased estimates for $\sigma(v, S)$ using a set of RIS samples. In this paper, we solve the IM problem by adopting a completely different strategy *i.e.,* a learning method. In our solution, $\sigma(v, S)$ is not estimated using RIS-based sampling. Instead, it is modeled as a parameterized function and approximated using deep RL. To this end, we introduce the notion of *learning-based IM problem.*

The *learning-based IM problem* consists of two phases, namely, *learning phase* and *inference phase.* In the learning phase, given a set of homogeneous networks $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ and an information diffusion model ψ, we train a set of parameters Θ such that function $y = \tilde{\sigma}(v, S; \Theta)$ can be used to approximate $\sigma(v, S)$ as accurately as possible. In the inference phase, given a target network G, integer k and a function $y = \tilde{\sigma}(v, S; \Theta)$ that approximately calculates the marginal influence of v with respect to the partial solution S, we solve the IM problem in G with respect to budget k and diffusion model ψ.

3.3 Diffusion Models

Based on the definition of IM, one can observe that a diffusion model ψ is vital for the selection of seeds. Currently, there exist two popular diffusion models, namely *Linear Threshold* (LT) and *Independent Cascade* (IC). Throughout a diffusion process, a node has two possible states, activated or inactivated. Both models assume that, when a node is activated, its state will not change further.

Linear Threshold (LT) model. The LT model is a special case of the triggering model [22]. To explain the concept briefly, we introduce $N(v)$ (*resp.,* $N^\eta(v)$), which is a set of (*resp.,* activated) neighbors of node v where each node has a threshold θ_v. $\forall u \in N(v)$, an edge (v, u) has a non-negative weight $w(v, u) \leq 1$. Given a graph G and a seed set S, and the threshold for each node, this model first activates the nodes in S. Then it starts spreading in discrete timestamps according to the following random rule. In each step, an inactivate node v will be activated if $\sum_{u \in N^\eta(v)} w(v, u) \geq \theta_v$. The newly activated node will attempt to activate its neighbors. The process stops when no more nodes are activated.

Independent Cascade (IC) model. Given a graph G and a seed set $S \subset V$, this model first activates the nodes in S, and then starts spreading in discrete timestamps according to the following rule. When a node u is first activated at timestamp t, it gets a chance to activate a node in its neighborhood that is not activated. The success probability of activation is $w(u, v)$. If v is activated successfully, v will become an active node in step $t+1$ and u can no longer activate other nodes in subsequent steps. This process continues until no new nodes can be activated. In other words, whether u can activate v is not affected by previous propagation.

4 LEARNING THE MAPPING FUNCTION

Notably, in traditional approximate IM solutions, it is inevitable to sample the diffusion phase to generate a set of RIS samples and the number of required RIS samples is *at least* as large as the threshold

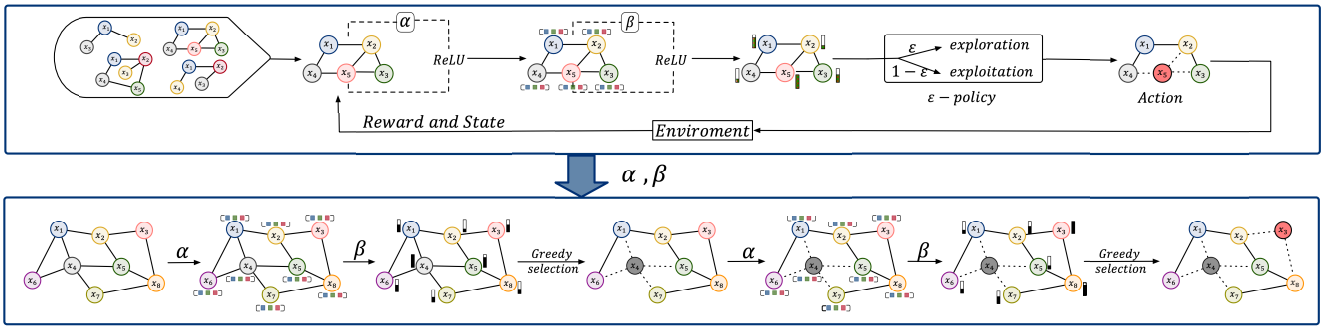


Figure 1: piano framework by incorporating network embedding and deep reinforcement learning. (with $k = 3$)

θ. In this paper, we turn to machine learning methods to avoid the traditional diffusion sampling phase in seed selection to make it more generalizable and scalable. In this section, we shall present our PIANO framework that casts the IM problem as finding the optimal policy for selecting the seed set within a deep RL framework. As remarked earlier, the key challenge in addressing IM lies in the estimation of expected maximum influence function $\sigma(v, S)$. In our deep RL framework, $\sigma(v, S)$ plays the role of values (*i.e.,* the expected return for selecting action v in state S), which we approximate with value function $\tilde{\sigma}(v, S; \Theta)$ as accurately as possible. Note that since the IM problem is NP-hard, the ground-truth label for σ is hard to acquire to train a supervised model. In PIANO, we adopt *Deep Q-Network* (DQN) [23, 32], a popular deep RL model to learn the parameters Θ form reward.

Next, we introduce the learning phase of PIANO, which consists of network embedding, training of the parameters Θ, and approximating σ with $\tilde{\sigma}$ for use in DQN. The test phase for selecting nodes according to the learned parameters and predicted influences will be detailed in the next section. An overview of the PIANO framework is depicted in Figure 1, where the top half depicts the learning phase while the bottom half illustrates the test phase (*i.e.,* seed selection).

4.1 Embedding the Nodes

Before DQN model can be applied, we shall first embed the nodes into feature vectors based on the topological information. To this end, we need the embedding of each node $v \in V$ as a vector x_v. Among series of embedding methods, *e.g.,* DeepWalk [34], node2vec [14], DeepInf [35], etc., we select Structure2vec [10] to accomplish this step due to the following reasons. Firstly, the other alternatives are 'transductive' embedding methods, that is, they assume that the test graph is observed during training of embeddings. As a result, embeddings extracted across graphs are not consistent, since they only care about intra-graph proximity. In our framework, we aim to use an 'inductive' method to complete the cross-graph extraction of embedded results. This means that the parameters trained in the subgraphs can be applied to the target graph. Secondly, since they are unsupervised network embedding methods (*i.e.,* DeepWalk and node2vec) or supervised for a particular task (*i.e.,* DeepInf), they may not capture the desired information (*i.e.,* expected influence spread) for the IM problem. In our case, the network embedding is trained end-to-end for the optimization target (RL reward), thus the encoded features can be more discriminative for the IM task.

Structure2vec learns nonlinear mapping with discriminative information using stochastic gradient descent, and embeds latent variable models into feature space. It combines the characteristics of nodes, edges and network structure in the target network. These characteristics will recursively aggregate according to the state of the target network. Notably, Structure2vec can learn the embedding of nodes in an *end-to-end* manner through the combination with DQN. That is, the parameters learned from such combination are exclusively suitable for the test network and application scenarios.

Given the current partial solution S, Structure2vec will calculate the q-dimensional feature embedding for each node v ($v \in V$). Firstly, we initialize the vectors of all nodes and set each of them as a q-dimensional zero vector[1]. Then Structure2vec recursively defines the network architecture based on the input network structure G. After I iterations, (I is usually small, set to 4 or less), each node v reaches to the final state, and the embedding at this time can simultaneously take into account the topological features and remote interaction between these nodes. In addition, in the IM scenario, each node also needs a specific flag to indicate whether node v is in the partial solution S or not, which is denoted as a_v. That is, $a_v = 1$ if the node appears in the seed set S, otherwise $a_v = 0$. Because Structure2vec is implemented in a scheme similar to the network model inference process, the nodes' specific label a_v is also recursively aggregated according to the input network topology.

The formula for the update of vectors is as follows:

$$x_v^{(i)} := ReLU(\alpha_1 \sum_{u \in N(v)} x_u^{(i-1)} + \alpha_2 \sum_{u \in N(v)} ReLU(\alpha_3 w(v,u)) + \alpha_4 a_v). \tag{1}$$

In Eq. 1, *ReLU* refers to the non-linear activation Rectified Linear Unit, $N(v)$ is the neighbor set of node v, $x_v^{(i)}$ represents the vector of node v during the i-th iteration, $w(v, u)$ is the weight of edge (v, u), and $\alpha_1, \ldots, \alpha_4$ are the parameters that need to be trained. Although all the neighbors of each node v remain unchanged during the update process, in order to better model the nonlinear mapping of the input space, we add two parameters α_2 and α_3 to construct two independent layers of Multi-Layer Perceptron in the above formula.

It can be seen that the first two items in the equation aggregate the surrounding information by summing up the neighbors of v.

[1]In line with [10], q is generally set to 64, it can be adjusted according to the size of the network

Besides, during the iterations, the update formula can also pass information and network characteristics of a node across iterations. When the embeddings of all nodes have been updated, the next iteration begins. After I iterations, the vector of v will contain information about all neighbors within the I-hop neighborhood.

4.2 The Reinforcement Learning Formulation

In the IM problem, we are unable to acquire sufficient labeled samples for training the parameters Θ in a supervised way, as the exact evaluation of $\sigma(v, S)$ is #P-hard. Hence, we frame the IM problem as policy optimization through deep RL, where the learning task is to find the optimal policy (action sequence) for selecting the seed nodes. The deep RL formulation enables end-to-end learning from the Agent's Perception to Action and automatically learns complex features suitable for the task. In our IM scenario, the Agent is the machine (computer program) that is learning to act (selecting seed nodes) optimally in an unknown and uncertain Environment (a network). The whole decision making process is modeled as a Markov Decision Process (MDP), where at each time step t, the agent perceives a state S_t (the current observation of the environment), performs an action A_t and gets a reward R_{t+1} while moving to the next state S_{t+1}, and the process continues for an episode. For the IM problem, we define these RL components as follows:

- *State*: S_t is the current configuration of the network G, where some nodes have already been selected as seed set and some have not. The final state S_k is the configuration where k nodes have been selected. The state representation of G is given by the network embedding method described above.
- *Action*: A_t adds a node v ($v \in \bar{S}$) to the current seed set S. In model-free RL, policy optimization needs values (total expected return) for state-action pairs, which we approximate using $\tilde{\sigma}(v, S_t; \Theta)$, *i.e.,* the expected return for selecting node v at state S_t. We define the value function formally in Eq. 2.
- *Reward*: The environment returns a reward $R_{t+1} \in \mathbb{R}$ for its action $A_t = v$ in state S_t (*i.e.,* for adding node v to the current seed set), and moves to state S_{t+1}. The increment of the influence range is the reward for selecting v node in state S_t, $R_{t+1}(S_t, A_t = v) = \sigma(S_{t+1}) - \sigma(S_t)$.
- *State transition*: When a node $v \in \bar{S}$ is selected to include in the seed set S, a_v will change from 0 to 1. This will in turn change the node embeddings (or state) as shown in Eq. 1.
- *Episode*: A training episode $\mathcal{E}$ is a sequence of state-action-reward values $\mathcal{E} = (S_1, A_1, R_2, \ldots, S_k, A_k, R_{k+1})$.
- *Optimal policy*: The goal in RL is to find the policy, *i.e.,* action sequence $\pi_* = (A_1, \ldots A_k)$ that gives the maximum cumulative reward (or return) defined as $\mathbb{E}[\sum_{i=1}^{k} R(S_i, A_i = v)]$.

We frame IM as a model-free value-based RL problem [38] that finds the optimal policy π_* implicitly in terms of state-action values. Following the RL terminology, we will use Q function to refer to the state-action value function, *i.e.,* $Q(v, S_t; \Theta) = \tilde{\sigma}(v, S_t; \Theta)$. In state S_t, the $Q(v, S_t, \Theta)$ for node/action $A_t = v$ is defined as follows:

$$Q(v, S_t; \Theta) := {\beta_1}^T ReLU([\beta_2 \sum_{u \in V} x_u^I, \beta_3 x_v^I]). \quad (2)$$

where $x_v^I \in \mathbb{R}^q$ is the vector generated after I iterations; $[,]$ is the concatenation operator; and $\beta_1 \in \mathbb{R}^{128}$, $\beta_2, \beta_3 \in \mathbb{R}^{64 \times 64}$. Because $Q(v, S_t; \Theta)$ is mainly determined by the embedding of the current node v and its surrounding (I-hop) neighbors and their status (selected or not), the Q function is related to the parameters $\alpha_1 \sim \alpha_4, \beta_1 \sim \beta_3$, all of which are learned end-to-end. We denote all these parameters using Θ for simplicity. We describe learning of Q functions in Section 4.3.

The optimal Q function (specifically, the learned Θ) implicitly represents the optimal policy to find the k-best seed nodes. Given the network G, a budget of k nodes, a seed set S, and $\bar{S} = V \backslash S$ as a set of candidate nodes, at each step t, upon evaluating the quality of each node using the Q function, the node with the highest Q value (*i.e.,* marginal expected influence) is added to the seed set S. Formally, $A_t = argmax_{v \in \bar{S}} Q(v, S_t, \Theta)$. After adding a new node v to the seed set, a_v is changed from 0 to 1. In the new state S_{t+1}, all node embeddings need to be updated, and the $Q(v, S_{t+1}; \Theta)$ for the remaining nodes $v \in \bar{S}$ need to be reevaluated to select the next node $A_{t+1} = v'$ in S. This process is continued k times.

4.3 Model Training via DQN

We use the well-known DQN (Deep Q-Network) algorithm [31, 32] to learn the Q function $Q(v, S; \Theta)$. Since $Q(v, S; \Theta)$ is modeled as a deep neural network (the graph embedding model), it is also called a Q-net. DQN exhibits several advantages over other existing RL algorithms. It gives stability to Q-learning through the use of *experience replay* and Q-targets. Experience replay is a technique where the network uses *off-policy* learning to learn from experiences generated by old policies as opposed to online learning where the experiences are thrown away. This gives DQN sample efficiency compared to policy-based RL (*e.g.,* policy gradients), which only uses on-policy samples. Experience replay also helps to de-correlate the trajectories (action sequences) exhibiting better learning path. The loss function used by our DQN can be defined as:

$$\mathcal{L}(\Theta) = \mathbb{E}[(\underbrace{r + \gamma \max_{v'} \overbrace{Q(v', s'; \Theta)}^{\text{inference net}}}_{\text{Q-target (greedy policy)}} - \underbrace{\overbrace{Q(v, s; \Theta)}^{\text{learner net}}}_{\text{Q-net } (\varepsilon\text{-greedy policy})})^2] \quad (3)$$

where γ is a decaying factor, which controls the importance of future rewards in learning. If $\gamma = 0$, the model will not learn any thing from future (short-sighted), and only pay attention to the current reward; for $\gamma >= 1$, the expected return is continuously accumulated and may diverge. Therefore, γ is generally set to a number slightly smaller than 1 (*e.g.,* 0.95 in our implementation).

The training process is outlined in Algorithm 1. Our behaviour policy is ε-greedy, which selects a random node with probability ε (exploration), and with probability $(1-\varepsilon)$ it selects a node greedily (exploitation). We vary ε within $0.05 \sim 1$ in our implementation to find the best model. We use a batch of homogeneous networks for training. An episode represents the process of obtaining a complete sequence of a network seed set S (Lines 1-11). For each network, the seed set S is first initialized (Lines 2). According to the embedding process discussed earlier in this section, we update the nodes' embeddings and calculate the Q value for each node (Lines 3-4). After getting the influence quality of each node, we apply the aforementioned ε-greedy policy. The selected node is then added to the seed set (Lines 5-7). These tuples are added to the replay

Algorithm 1: Parameter learning

Input: Batch of training network, a positive number k, experience replay memory $\mathcal{J}$ to capacity N.
Output: parameters $\Theta = \{\alpha_1, \ldots, \alpha_4, \beta_1, \ldots, \beta_3\}$

1 for episode $e = 0; e < E$ do
2 Sample a network G and initialize seed set $S = \phi$;
3 for $t = 0$ to $k - 1$ do
4 Compute the embedding for each node $v \in V$;
5 Calculate the Q value of each node;
6 $v_t = \begin{cases} \text{Random node } v \in \bar{S}, & \text{w.p. } \varepsilon \\ argmax_{v \in \bar{S}} Q(v, S_t; \Theta), & \text{w.p. } 1 - \varepsilon \end{cases}$;
7 add v_t to S;
8 update $S, \bar{S}$;
9 Add tuple $\langle S_t, v_t, R_{t+1}, S_{t+1} \rangle$ to $\mathcal{J}$;
10 Sample random batch j of $\langle s, v, r, s' \rangle$ tuples from $\mathcal{J}$;
11 Update Θ by SGD with j using Eq. 3;

memory $\mathcal{J}$ from which a random batch of tuples is selected (with IID assumption) to update Θ with SGD (Lines 9-11).

5 SEEDS SELECTION

5.1 Generating Result Set via Learned Function

The training process results in the learned parameters Θ, which implicitly represent the learned optimal policy. Once the parameters are learned, they can be used to address IM problem in multiple networks. In this part, we shall show the test phase of PIANO model, *i.e.,* online selection of seeds. First of all, as we have learned all the parameters in $\tilde{\sigma}(v, S; \Theta)$, we are able to directly calculate the predicted expected marginal influence for each node. Afterwards, intuitively, it is natural for one to follow the existing hill-climb strategy to iteratively select the best node that exhibits the highest predicted marginal influence. However, our empirical (shown in Section 6.2) and theoretical studies (will show immediately) reveal that the order of the nodes with respect to their Q values remains almost unchanged whenever we select a seed and recompute the network embeddings as well as the Q values. Therefore, for the seeds selection phase, we hereby present a much better solution that is strictly suitable for our PIANO framework. To begin with, we shall first conduct a study over the intrinsic features for our learning model. In particular, with the help of Certainty-Factor (CF) model [18], we theoretically examine the difference of the predicted influences for any pair of nodes before and after some seed is selected into results set. The CF model is a method for managing uncertainty in rule-based systems. A CF represents a person's change in belief in the hypothesis given the evidence. In particular, a CF from 0 to 1 means that the person's belief increases.

For an arbitrary pair of non-seed nodes v_a and v_b, and current seeds set S_i, there corresponding predicted influences are $\tilde{\sigma}(v_a, S_i; \Theta)$ and $\tilde{\sigma}(v_b, S_i; \Theta)$, respectively. Suppose after a new seed, but not v_a or v_b, is selected into the result set, leading to a new partial solution S_{i+1}. For brevity, we denote $\tilde{\sigma}(v_a, S_i; \Theta)$ with Q_A and $\tilde{\sigma}(v_a, S_{i+1}; \Theta)$ with Q'_A. Similarly, we also use Q_B and Q'_B for the values of node v_b. Then the following holds.

Algorithm 2: Seeds selection procedure

Input: network $G = (V, E, W)$, a positive number k, parameters $\Theta = \{\alpha_1, \ldots, \alpha_4, \beta_1, \ldots, \beta_3\}$
Output: optimal solution S_k

1 Initialize a seed set $S = \phi$ and $x_v^{(0)} = 0 (v \in V)$.
2 for $j = 1$ to I do
3 for $v \in V$ do
4 $x_v^{(j)} := ReLU(\alpha_1 \sum_{u \in N(v)} x_u^{(j-1)} + \alpha_2 \sum_{u \in N(v)} ReLU(\alpha_3 w(v, u)) + \alpha_4 a_v)$
5 for $v \in V$ do
6 $Q(v, \emptyset; \Theta) = \beta_1{}^T ReLU([\beta_2 \sum_{u \in V} x_u^I, \beta_3 x_v^I])$
7 $S_k \leftarrow$top-k node v with the highest $Q(v, \emptyset; \Theta)$;

Theorem 5.1. $\forall v_a, v_b \in V$, where v_a, v_b are not seeds, and a very small positive number η,

$$\begin{aligned} &\text{if } |(Q_A - Q_B) - (Q'_A - Q'_B)| < \eta \\ &\text{then } (Q_A - Q_B)(Q'_A - Q'_B) > 0, \text{CF} = 1 \end{aligned} \tag{4}$$

That is, the order of any two nodes before and after recomputing the embeddings and Q values does not change (The proof is given in Appendix A).

Further, we justify that η is expected as a very small positive value.

Claim: *$\forall v_a, v_b \in V$ and v_a, v_b are not seeds, suppose the number of nodes and average degree of graph G is n and $|\overline{N(v)}|$, respectively. Then, according to PIANO model (where I is fixed less than 5 and Q values are Max-min Normalized):*

$$E[|(Q_A - Q_B) - (Q'_A - Q'_B)|] < \frac{\sum_{i=1}^{4} |\overline{N(v)}|^i}{n}$$

(The proof is given in Appendix B). In practice, the average degree of nodes in real-world social networks are always very small (*i.e.,* within 3~7) [24] compared to the size of the network. Therefore, η is expected to be a very small positive value in practice.

According to the above study, it is known that $|(Q_A - Q_B) - (Q'_A - Q'_B)|$ should probably be a very small non-negative value. That is to say, whether or not we recompute the embeddings and Q values after each seeds insertion can hardly affect the order of non-seed nodes. Therefore, during the seeds selection phase, instead of iteratively select-and-recompute the embeddings and Q values according to each seed insertion, we simplify the procedure into only one iteration, by embedding only once and selecting the top-k nodes with the maximum Q without updating the predicted values.

The seeds selection process is outlined in Algorithm 2. Given a network G and a budget k, we first initialize the seed set S to be empty and initialize each node in the network as a p-dimensional zero vector (Lines 1). Next, we embed each node in the network (Lines 3-5). The update of the embeddings can be performed according to Equation 1, and the parameters $\alpha_1, \ldots, \alpha_4$ have already been learned. The formula aggregates the neighborhood information of node v, and encodes the node information and network characteristics (Lines 5). After getting embeddings of the nodes, we calculate the influence quality of each node v according to Equation 2 using

the learned parameters $\beta_1, \ldots, \beta_3$ (Line 6). Finally, the node with the top-k influence quality is added to the seed set.

The time complexity of the selection process is determined by three parts. First, in the embedding phase, the complexity is influenced by the number of nodes $|V|$ and the number of neighbors $|\overline{N(v)}|$. As I is usually a small constant (*e.g.,* $I = 4$) in Algorithm 2, network embedding takes $O(|V| \times |\overline{N(v)}|)$. Second, after all the nodes in the network are embedded, the quality of nodes in the graph is evaluated using the formula $Q(v, S, \Theta) = \beta_1{}^T ReLU([\beta_2 \sum_{u \in V} x_u^I, \beta_3 x_v^I])$. Since the values of Θ have been learned, this step is influenced by the time taken to find the neighbors of node v. Hence, it takes $O(|V| \times |\overline{N(v)}|)$ time. Finally, selecting the optimal node according to Q function and adding the node to the set S takes $O(|V|)$. Consequently, the total time complexity for seeds selection is $O(|V| \times |\overline{N(v)}|)$.

5.2 Applying PIANO in Practice

As a learning-based framework (also shown in Definition 3.2), we need to pretrain the embedding and Q function parameters offline using a group of training networks. Intuitively, the training and testing (*i.e.,* target) networks should be homogeneous in terms of topology such that the quality of the learned parameters can be guaranteed. In general, the homogeneity in terms of topology can be reflected in the aspects of many topological properties, *e.g.,* degree distribution, spectrum, etc. Therefore, in order to select seeds within a targeted network, we need to train the required parameters Θ in a group of homogeneous networks offline. Afterwards, the trained model can be further used to select seeds in a target network. In the following, depending on the specific characteristic of target networks, we shall present three pretraining strategies, namely PIANO-S, PIANO-E and PIANO@$\langle d \rangle$, respectively.

PIANO-S: applying PIANO in large-scale stationary networks. Consider a large-scale stationary network without any evolution log. We can turn to subgraph-sampling technique to generate sufficient homogeneous training networks. Afterwards, with learned parameters Θ from these small networks, we can address IM over the target large-scale network. In order to ensure that the topological features of the sampled training subgraphs are as consistent as possible with the original large-scale target network, we evaluate different sampling algorithms following the same framework adopted in [2, 26]. In particular, we apply different sampling methods to real large-scale networks and sample subgraphs over a series of sample fractions $\varphi = [1\%, 5\%, 10\%, 15\%]$. Following the methods introduced in [2], Kolmogorov-Smirnov D-statistic is adopted to measure the differences between the network statistics distributions of the sampled subgraphs and the original graph, where D-statistic is typically applied as part of the Komogorov-Smirnov test to reject the null hypothesis. It is defined as $D = \max_x\{|F_0(x) - F(x)|\}$, where x denotes the range of random variables; F_0 and F are two empirical distribution functions of the data. We found that our experimental results show similar phenomenon with the benchmarking papers [2, 26]. That is, Topology-Based Sampling is superior to Node Sampling and Edge Sampling in the distribution of graph features such as degree distribution and clustering coefficient distribution. So in the next section, we adopt Topology-Based sampling methods in our model and compare several existing Topology-Based subgraph sampling methods, including Breadth First Sampling (BFS)[11], Simple Random Walk (SRW) and its variants, namely Random Walk with Flyback (RWF) and Induced Subgraph Random Walk Sampling (ISRW) [25, 27], as well as Snowball Sampling (SB)[12], a variant of Breadth First Search which limits the number of neighbors that are added to the sample. Notably, as the subgraph-sampling technique is beyond the focus of this work, we do not discuss the detailed techniques for these methods.

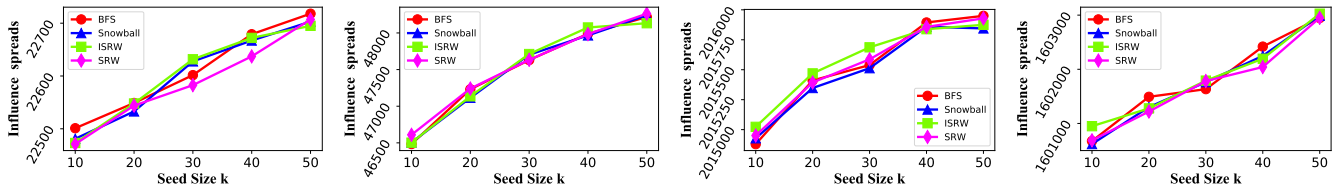
Table 1: Datasets

Dataset	n	m	Type	Avg.Degree
HepPh	34K	421K	Directed	9.83
DBLP	317K	1.05M	Undirected	3.31
LiveJournal	4.85M	69M	Directed	6.5
Orkut	3.07M	117.1M	Undirected	4.8

PIANO-E: applying PIANO in evolutionary networks. Almost all existing social networks keep on evolving, with insertion of new nodes or edges and deletion of old nodes or edges. Although social networks evolve rapidly over time, the structural properties remain relatively stable [24]. During the evolution of a particular real-world network, we advocate that any two historical snapshots of the same network are homogeneous to each other. By leveraging on the aforementioned technique, we now briefly describe how PIANO can address the IM problem in dynamic networks via time-based sampling. In particular, given a dynamic network G, whose snapshot at time t is referred to as G_t, we can apply PIANO in the following way. During the *Learning phase* of Definition 3.2, we can train the model using a series of temporal (sampled) network snapshots (*i.e.,* $\mathcal{G} = \{G_{t1}, \ldots, G_{t\ell}\}$). For the *Seeds selection phase*, the trained model can be used to select the best seed set in an arbitrary snapshot of G. Besides, if any snapshot of the network is very large, we can also adopt the subgraph sampling method mentioned above.

Based on this strategy, we can accomplish the seeds selection task over a target network in real-time, although it keeps on evolving. This provides a practical solution for the IM problem in evolutionary networks.

PIANO@$\langle d \rangle$: applying PIANO with our pre-trained model pool. Apart from the above approaches, we extensively provide an extremely friendly method for applying PIANO in practice, *i.e., pretrained once and apply everywhere.* We build a model pool by pretraining a series of PIANO models, each is learned over 200 synthetic networks (each contains 500 nodes) with identical average degree at $3, 4, \ldots, 9$, respectively. For instance, we can train a PIANO model from 200 synthetic networks with average degree of 5, referred to as PIANO@5. Given a target network G (with average degree d), where IM is to be addressed, we can find from our model pool PIANO@$\langle d \rangle$, where $\langle \cdot \rangle$ refers to the closet integer of d, and directly apply that on G without training again. The detailed justification and evaluation of the pretrained model pool can be found in the end of next section.



(a) HepPh (b) DBLP (c) LiveJournal (d) Orkut

Figure 2: Influence spreads with different sampling algorithms.

6 EXPERIMENTS

In this section, we evaluate the performance of the PIANO framework. We compare our model with two state-of-the-art traditional IM solutions, namely IMM [39] and SSA [33], as suggested by a recent benchmarking study [4]. Recall that the efforts in [3, 15, 30, 37] are designed for competitive IM and hence are orthogonal to our problem. In line with all the IM solutions, we evaluate the performances from two aspects, namely computational efficiency and influence quality. Besides, as a learning-based solution, we also justify our model generality within evolutionary scenarios. All the experiments were performed on a machine with Intel Xeon CPU (2.2GHz, 20 cores), 512GB of DDR4 RAM, Nvidia Tesla K80 with 24GB GDDR5, running Linux[2].

6.1 Experimental Setup

Datasets. In the experiments, we present results on four real-world social networks, taken from the *SNAP* repository[3], as shown in Table 1. In these four datasets, *HepPh* and *DBLP* are citation networks, *LiveJournal* and *Orkut* are the largest online social networks ever used in influence maximization. For each real-world network, we use the following sampling methods: Breadth First Sampling (BFS) [11], Simple Random Walk (SRW), Induced Subgraph Random Walk Sampling (ISRW) [25, 27], and Snowball Sampling (SB) [12].

Diffusion Models. PIANO can be easily adapted to different diffusion models. For instance, we can simply revise the Reward definition to switch from IC model to LT model. As our experimental results under both LT and IC models are qualitatively similar, we mainly report the results under the IC model here. In IC model, each edge $W(u, v)$ has a constant probability p. In vast majority of the IM techniques, $W(u, v)$ takes the value of 0.5 assigned to all the edges of the network. In order to fairly calculate the expected range of influence for the three approaches, we first record the seed set of each algorithm independently, and then perform 10,000 simulated propagations based on the selected seeds. Finally, we take the average result of 10,000 simulations as the number of influenced nodes for each tested approach.

Parameter Settings. For our method, we set the batch size, *i.e.,* the number of samples extracted from M each time, as 64. Besides, we set δ as 5 and the learning rate of SGD to 0.001. We set the dimension of the nodes embedded in the network to 64. As suggested in IMM [39], we set $\epsilon = 0.1$ throughout the experiments in both IMM and SSA. It should be noted that the seed set produced by SSA is not constant. Therefore, for SSA and IMM, we report the average results over 100 independent runs (*i.e.,* 100 independent results seed sets, each of which is averaged for 10,000 simulations).

6.2 Experimental Results

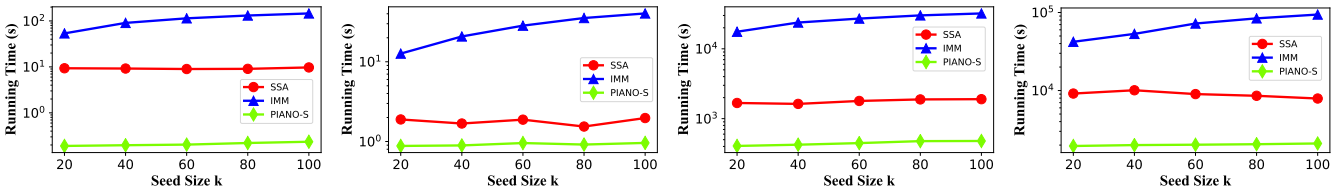
Training issues. As remarked earlier, during DQN training, we need to obtain a batch of training graphs by sampling the real-world network. In our experiments, we mainly compare the aforementioned four sampling methods (BFS, ISRW, Snowball, and SRW). The results are shown in Figure 2. Observe that for the PIANO training process, the results of the subgraph training model obtained by using different Topology-Based sampling methods have ignorable difference, indicating that different Topology-Based sampling method has little effect on the result seeds quality. In other words, the Topology-Based sampling method and the PIANO framework are loosely coupled. Therefore, we chose to use simple and mature BFS (Breadth First Sampling) as the default method in rest of the experiments.

Given the sampled sub-networks, we train the DQN parameters using ε-greedy exploration described in Section 4.3. That is, the next action is randomly selected with probability ε, and the action of maximizing the Q function is selected with probability $1 - \varepsilon$. During training, ε is linearly annealed from 1 to 0.05 in ten thousand steps. The training procedure will terminate when the value of ε is less than 0.05 or the training time exceeds 3 days. For both *HepPh*, *DBLP*, the training phases terminate at 2.5 and 23.5 hours, respectively; for *LiveJournal* and *Orkut*, we limit the training time within 3 days and apply the learned parameters to node selection.

Notably, *the training phase is performed only once and we do not need to train while running an IM query.* The study for PIANO-E on evolutionary networks further justifies that, once trained, it can be applied many times to address the IM problem.

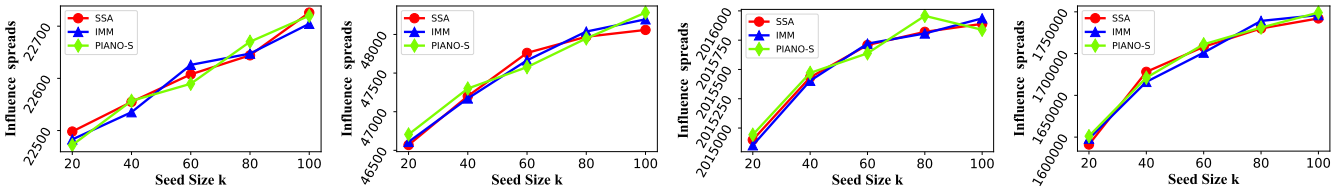
Computational efficiency. We compare the running times of the three algorithms by varying k from 10 to 50. The results on datasets *HepPh*, *DBLP*, *LiveJournal* and *Orkut* are reported in Figure 3. We can make the following observations. In terms of computational efficiency, PIANO-S significantly outperforms IMM and SSA by a huge margin. Regardless of the value of k, the cost of the PIANO-S is significantly less than that of IMM and SSA. Particularly, when the number of seed nodes selected is small, the performance gain is more prominent. For example, under the IC model, when $k = 10$ and $k = 50$, PIANO-S is 23 and 1.57 times faster than SSA on *Live-Journal* network, respectively. In addition, because we use the method of selecting the nodes ranked in the top K, when we select seed sets of different sizes, the time variation is small, but

[2]We shall provide the url of complete source code for piano here as soon as this paper is accepted.
[3]https://snap.stanford.edu/data/



(a) HepPh (b) DBLP (c) LiveJournal (d) Orkut

Figure 3: Running time.



(a) HepPh (b) DBLP (c) LiveJournal (d) Orkut

Figure 4: Influence spreads quality.

the premise of this experimental result is that we have sorted the scores of all nodes in the code. In the experiment, we found that for a data set with n nodes, the time complexity of sorting the scores of all nodes is at least $O(nlogn)$. However, if K nodes are directly selected from the unordered set, the time required is only $n \times k$. Therefore, the running time of PIANO-S has increased significantly over k.

Influence quality. Next, we compare the quality of the seed sets generated by IMM, SSA and PIANO-S on IC models. For each algorithm we set the optimal parameter values according to their original papers and then evaluate their quality. As can be seen from Figure 4, our proposed method is as good as IMM and SSA in real networks of different sizes, and the growth of influence spread with k have few minor fluctuations. Note that the scales of the vertical axis in the four figures are different. In our experiments, we also observe that SSA results are unstable. In comparison, our method can choose superior result set, and the results are stable across 100 runs. Importantly, PIANO-S produces the same or even better quality results as the state-of-the-art.In addition, the author proposed in IMM that ϵ is a tunable parameter. In the experiment, when $\epsilon = 0.1$, the quality of the IMM seed set is equivalent to ours. If we increase ϵ although the time efficiency will increase, the quality of the result will also decrease.

Justification of seeds selection strategy. When we select the seed set, we only execute the process of network embedding and then select k nodes with the largest Q values, instead of updating the embedding every time a seed node is selected. The purpose of this experiment is to show the difference over the results between the strategy shown in Section 5.1 and traditional hill-climb strategy, *i.e.,* select-and-update marginal influence iteratively. To this end, we compare the results by these two different strategies and report the differences in terms of node order and influence quality. We hereby report the results in Table 2. The column ΔInf

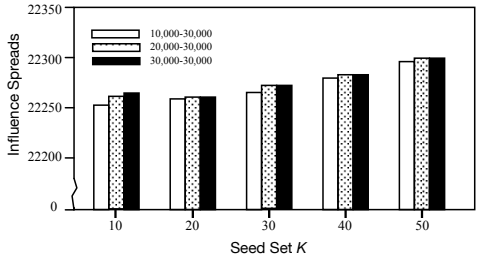


Figure 5: Performance of piano-e in real-world evolutionary networks.

records the difference between the result quality for our seeds selection strategy of Section 5.1 and the traditional hill-climb iterative method (*i.e.,* select a seed and update the marginal influence for the rest iteratively). The column titled $\Delta rank$ records the probability that the node order changes between the two strategies. As can be seen from Table 2, the probability of the order does not change in a seed set when we select one node iteratively or select 50 nodes at a time (0.972 on *Orkut*). Once again, this supports our theoretical discussion in Theorem 5.1 and Claim 5.1.

Performance of PIANO-E in evolutionary networks. We evaluate the performance of PIANO on evolutionary networks following the strategy discussed in Section 5.2. Among all the tested datasets, *HepPh* is associated with evolution logs, *i.e.,* the timestamps when each node/edge is inserted to the network. Therefore, following the proposed implementation model of PIANO-E in evolutionary network, we train the model using a series of temporal snapshots for *HepPh* networks when it has 10, 000, 20, 000, and 30, 000 nodes, respectively. Then the three trained models are tested on a future snapshot of the network, which contains 30, 000 nodes, during the evolution. As can be seen in Figure 5, the trained model from 10, 000-nodes snapshot and that from 30, 000-nodes ones have little differences in terms of the influence quality. For instance, when the number of seeds is 50, the difference between

Table 2: Effectiveness of our seeds selection strategy.

	$k = 1$		$k = 10$		$k = 25$		$k = 50$	
	$\Delta rank$	ΔInf	$\Delta rank$	ΔInf	$\Delta rank$	ΔInf	$\Delta rank$	ΔInf
HepPh	1	1	0.997	0.996	0.970	0.996	0.95	0.995
DBLP	1	1	0.942	0.998	0.962	0.998	0.968	0.996
LiveJournal	1	1	1	0.999	0.985	0.998	0.979	0.998
Orkut	1	1	0.983	0.995	0. 965	0.997	0.972	0.993

$10,000$-nodes model and $30,000$-nodes model in the influence spread is 224, which is about 1% of the total influenced nodes. Therefore, in real-world dynamic networks, PIANO-E can be easily trained using earlier snapshot or even the subnetworks of some snapshots. When the network evolves and expands, we can continue to use the pretrained model for selecting seeds in a larger network snapshot.

Pre-trained model pool and generalization. According to the above experiments, whenever one wants to apply PIANO to address IM problem in a target network G, she by default need to sample a group (*e.g.,* 200) of subgraphs of G and train PIANO first. In this experiment, we extensively discuss how this training phase can be pre-performed and avoided in practical IM tasks. That is, is it possible to provide a pretrained PIANO model that can be directly applied (*i.e.,* without training phase) over a new network where IM is to be addressed. To answer the question, we generate a series of synthetic networks with *SNAP* tool for training and evaluate the performance of the corresponding learned model. We test its performance with respect to the size and the generation schemes, and compare its performance with the approach of sampling from the target network. Herein, generation scheme refers to the golden rule we adopt to guide the synthetic graph generation procedure. In particular, we can generate synthetic graphs with a pre-defined average degree (*resp.,* power exponent of power-law degree distribution, average clustering coefficient). Based on this group of study, we are able to unveil the correlation between the model performance and training graphs, and also provide a high-level suggestion on how the training graphs should be selected. We ensure that a particular network topological property of the generated graph is consistent with the original graph, and then change the size of the generated graphs, train each batch of generated graphs to observe the changes in the quality of the results. The results are shown in Figure 6a and 6b ($p = 0.05$). Observed from the results, when the size of training graphs increase from 10^1 to 10^2, as the size of training graphs increases, the influence spreads for the learned model also increases. As shown in Figure 6a (*resp.,* 6b), when the sizes of training graphs are larger than 10^2, the performance of the learned model remains unchanged. This means that small training graphs (with less than 100, *resp.,* 250 nodes) are enough to train an accurate model as long as the network topology of the training graph is consistent with the target one. In addition, the training efficiency of the small graphs will be greatly improved compared to the large graphs, which can be reduced from a few days to a few hours. And it can be seen from Figure 6 that the quality of the results obtained by the selection of different topological properties shows trivial differences. Among the three properties, we recommend that the average degree of the generated graph and the original graph should be the same to achieve the expected result quality at a smaller scale. Besides, we also compare the performance of training from synthetic graph and sampled graph under the same size and average degree. Notably, among all the datasets, there is negligible difference between both approaches. That is, a PIANO model trained over some synthetic graphs can be directly applied in a real-world network as long as they exhibit the same average degree. Therefore, we build a model pool by pretraining a series of PIANO models, each is learned over 200 synthetic networks (each contains 500 nodes) with identical average degree at $3, 4, \ldots, 9$, respectively. According to our discussion in the end of last section, given a target network G (with average degree d), where IM is to be addressed, we can find from our model pool PIANO@$\langle d \rangle$ and directly apply that on G without training again. To justify the effect of our model pool, we show the performance of PIANO@$\langle d \rangle$ compared with PIANO-S, which trains the model from scratch (over subgraph samples of the target network) in Figure 6c and 6d. Obviously, there is little difference between PIANO-S and PIANO@$\langle d \rangle$.

7 CONCLUSIONS

In this work, we have presented a novel framework, PIANO, to address the IM problem by exploiting machine learning technique. To the best of our knowledge, we are the first to employ deep learning methods to address the classical IM problem. Our framework incorporates both DQN and network embedding methods to train superior approximation of the σ function in IM. Compared to state-of-the-art sampling-based IM solutions, PIANO can avoid the costly diffusion sampling phase. By training with sampled subnetworks once, the learned model can be applied many times. Therefore, we are able to achieve superior efficiency compared to state-of-the-art classical solutions. Besides, the result quality in terms of influence spread of PIANO is the same or better than the competitors. As the seminal effort towards the paradigm of learning-based IM solution, PIANO demonstrates exciting performance in terms of result quality and running time. We believe that our effort paves the way for a new direction to address the challenging classical IM problem.

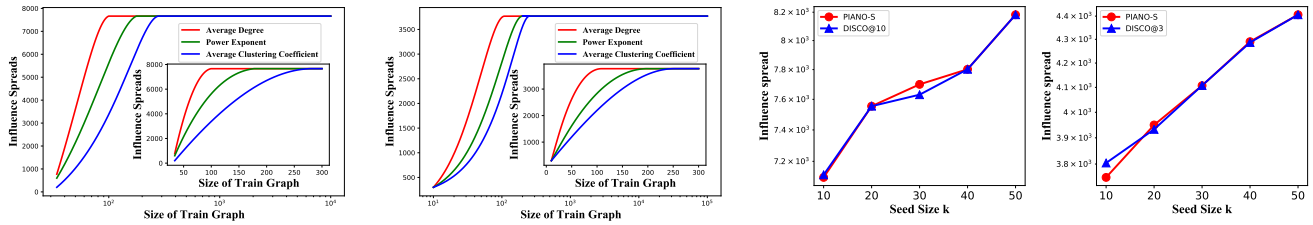
A PROOF OF THEOREM 5.1

Given $|(Q_A - Q_B) - (Q'_A - Q'_B)| < \eta$, there are two different cases as follows.

Case 1: $(Q_A - Q_B) - (Q'_A - Q'_B) \geq 0$, then it is $(Q_A - Q_B) - (Q'_A - Q'_B) < \eta$;

Case 2: $(Q_A - Q_B) - (Q'_A - Q'_B) < 0$, then it is $(Q_A - Q_B) - (Q'_A - Q'_B) > -\eta$.

We shall prove the theorem in *Case 1* first.



(a) HepPh (b) DBLP (c) HepPh (d) DBLP

Figure 6: Performance of piano@$\langle d \rangle$: (a-b) varying the size of training graph and topological properties; (c-d) comparison with sampling-based piano-s.

First we assume that at state S, given Q value of every node, which are ranked in the following order:

$$Q_1 > Q_2 > \cdots > Q_c > \cdots > Q_i > \cdots > Q_j > \cdots > Q_n$$

For any nodes i and j, $Q_i - Q_j > \eta$, where η is a small positive value. We find that after selecting a seed node c, the Q values recalculated after embedding are updated to the following values, respectively:

$$Q'_1, Q'_2, \cdots, Q'_i, \cdots, Q'_j, \cdots, Q'_{(n-1)}$$

As $(Q_i - Q_j) - (Q'_i - Q'_j) < \eta$, we calculate the probability for Q'_i and Q'_j to keep their order, that is to calculate $P(Q'_i - Q'_j > 0)$. Besides, $(Q_i - Q_j) - (Q'_i - Q'_j) < \eta$ is equivalent to $(Q'_i - Q'_j) > (Q_i - Q_j) - \eta$. To get $P(Q'_i - Q'_j > 0)$, the above probability can be inferred by scaling to

$$P(Q'_i - Q'_j > 0) > P((Q_i - Q_j) > \eta)$$

As $\eta > 0$, $P((Q_i - Q_j) > \eta)$, by the condition $Q_i - Q_j > \eta$, $P((Q_i - Q_j) > \eta) = 1$ can be obtained, then $P(Q'_i - Q'_j > 0) = 1$.

For *Case 2*, the proof is the same as *Case 1*.

B PROOF OF CLAIM 5.1

The proof can be separated into two stages.

Stage 1: Firstly, we can prove that when $I = 4$,

$$\begin{aligned} &|(Q_A - Q_B) - (Q'_A - Q'_B)| = \\ &|{\beta''_1}^T \beta_3 \alpha_2 \alpha_3 \sum_{i=1}^{4} \alpha_1^{i-1} [(\mathcal{N}_A^{(S')(i)} - \mathcal{N}_A^{(S)(i)}) \\ &- (\mathcal{N}_B^{(S')(i)} - \mathcal{N}_B^{(S)(i)})]| \end{aligned} \tag{5}$$

where $\mathcal{N}_A^{(S')(i)}$ refers to the sum of the embeddings for nodes that are i-th hop away from node A when all nodes in S have been removed from the graph.

According to the update process of Q at state S

$$Q_v = {\beta_1}^T ReLU([\beta_2 \sum_{u \in V} x_u^{(S)(I)}, \beta_3 x_v^{(S)(I)}]) \tag{6}$$

$\beta_1^T \in \mathbb{R}^{1 \times 2p}$, and $\beta_2^T, \beta_3^T \in \mathbb{R}^{p \times p}$. We divide β_1^T into two parts, the former p column is ${\beta'_1}^T$, and the last p column is ${\beta''_1}^T$. Thus the update formula for Q_v at state S' is:

$$Q'_v = {\beta'_1}^T ReLU\left(\beta_2 \sum_{u \in V} x_u^{(S')(I)}\right) + {\beta''_1}^T ReLU(\beta_3 \, x_v^{(S')(I)}) \tag{7}$$

We can get it from Eq. 7 as follows.

$$\begin{aligned} &P(|(Q_A - Q_B) - (Q'_A - Q'_B)| < \epsilon) = \\ &P(|[{\beta''}^T_1 (ReLU(\beta_3 x_A^{(S)(I)}) - ReLU(\beta_3 x_B^{(S)(I)}))] \\ &- [{\beta''}^T_1 (ReLU(\beta_3 x_A^{(S')(I)}) - ReLU(\beta_3 x_B^{(S')(I)}))]|) \end{aligned}$$

Notably, in the initial state, all parameters are random numbers in the interval $(0, 0.1)$. Besides, all the parameters in Θ are vectors, whose entries are non-negative (in implementation they are all normalized into [0,1]). Without loss of generality, we assume all these parameters are p-dimensional. Then, *ReLU* can be resolved to get

$$\begin{aligned} &P(|(Q_A - Q_B) - (Q'_A - Q'_B)| < \eta) = \\ &P\left(\left|{\beta''}^T_1 \beta_3 \left[(x_A^{(S)(I)} - x_B^{(S)(I)}) - (x_A^{(S')(I)} - x_B^{(S')(I)})\right]\right| < \eta\right) \end{aligned}$$

Hereby, $x_v^{(S)(I)}$ represents the final vector after I iterations for state S. During the current state, A and B are not seed nodes, so the corresponding a_v is 0. Next, we will use $x_A^{(S')}$ as an example to expand the equation. $x_A^{(S)}, x_B^{(S)}, x_B^{(S')}$ are the same as $x_A^{(S')}$, and are not described here. Then the update formula for each iteration is

$$x_A^{(S')(4)} = \left(\alpha_1 \sum_{u \in N(A)} x_u^{(S')(3)} + \alpha_2 \alpha_3 \sum_{u \in N(A)} w(u, A)\right) \tag{8}$$

Assume that A have κ neighbors, which are $u_1, \cdots, u_\kappa$, then Eq. 8 can be expanded to:

$$x_A^{(S')(4)} = \left(\alpha_1 \left(\mu_{u_1}^{(S')(3)} + \cdots + \mu_{u_\kappa}^{(S')(3)}\right) + \alpha_2 \alpha_3 \sum_{u \in N(A)} w(u, A)\right)$$

Similarly, we assume that u_κ have λ neighbors, namely $u_{\kappa(1)}, \cdots, u_{\kappa(\lambda)}$. For $u_{\kappa(\lambda)}$, we assume that there are μ neighbors, namely $u_{\lambda(1)}, \cdots, u_{\lambda(\mu)}$. For $u_{\lambda(\mu)}$, we assume that there are ν neighbors, namely $u_{\mu(1)}, \cdots, u_{\mu(\nu)}$. Because in our experiment, $x_v^{(S')(0)} = 0$ of all nodes, so that can be expressed as follows:

$$
\begin{aligned}
x_A^{(S')(4)} = {} & \alpha_1^3\alpha_2\alpha_3 \sum_{\substack{u_\kappa \in \\ N(A)}} \sum_{\substack{u_{\kappa(\lambda)} \in \\ N(u_\kappa)}} \sum_{\substack{u_{\lambda(\mu)} \in \\ N(u_{\kappa(\lambda)})}} \sum_{\substack{u_{\mu(\nu)} \in \\ N(u_{\lambda(\mu)})}} w\left(u_{\mu(\nu)}, u_{\lambda(\mu)}\right) \\
& + \alpha_1^2\alpha_2\alpha_3 \sum_{\substack{u_\kappa \in \\ N(A)}} \sum_{\substack{u_{\kappa(\lambda)} \in \\ N(u_\kappa)}} \sum_{\substack{u_{\lambda(\mu)} \in \\ N(u_{\kappa(\lambda)})}} w\left(u_{\lambda(\mu)}, u_{\kappa(\lambda)}\right) \\
& + \alpha_1\alpha_2\alpha_3 \sum_{\substack{u_\kappa \in \\ N(A)}} \sum_{\substack{u_{\kappa(\lambda)} \in \\ N(u_\kappa)}} w(u_{\kappa(\lambda)}, u_\kappa) \\
& + \alpha_2\alpha_3 \sum_{u_\kappa \in N(A)} w(u_\kappa, A)
\end{aligned}
$$

As the weight is fixed, so $x_A^{(S')(4)}$ is mainly related to the number of neighbors, and the formula can be solved as $\alpha_2\alpha_3(\alpha_1^3\mathcal{N}_A^{(S')(4)} + \alpha_1^2\mathcal{N}_A^{(S')(3)} + \alpha_1^1\mathcal{N}_A^{(S')(2)} + \alpha_1^0\mathcal{N}_A^{(S')(1)})$. The representation in the network is approximately the sum of the number of neighbors of all nodes within four hops of node A.

Finally, we can get Eq. 5.

Stage 2: Secondly, we study the right side of Eq. 5, which can be further derived as:

$$
\begin{aligned}
& |\beta_1''^T\beta_3\alpha_2\alpha_3 \sum_{i=1}^{4} \alpha_1^{i-1}[(\mathcal{N}_A^{(S')(i)} - \mathcal{N}_A^{(S)(i)}) \\
& - (\mathcal{N}_B^{(S')(i)} - \mathcal{N}_B^{(S)(i)})]| \\
\leq & \sum_{i=1}^{4} |\beta_1''^T\beta_3\alpha_2\alpha_3\alpha_1^{i-1}[(\mathcal{N}_A^{(S')(i)} - \mathcal{N}_A^{(S)(i)}) \\
& - (\mathcal{N}_B^{(S')(i)} - \mathcal{N}_B^{(S)(i)})]|
\end{aligned}
$$

For ease of discussion, we denote the right part for $i = 1$ to 4 as Σ_1 to Σ_4, respectively. For the case $i = 1$, it is not hard to see that only when the new seed, say s, appears as either v_a's or v_b's (but not both's) instant neighbor, $[(\mathcal{N}_A^{(S')(i)} - \mathcal{N}_A^{(S)(i)}) - (\mathcal{N}_B^{(S')(i)} - \mathcal{N}_B^{(S)(i)})]$ is not zero. Notably, if s is both v_a's and v_b's neighbor at the same time, the value within the square brackets will also be zero. Therefore,

$$
\begin{aligned}
E[\Sigma_1] = & (P[s \in N(v_a)^{(1)}, s \notin N(v_b)^{(1)}] \\
& + P[s \in N(v_b)^{(1)}, s \notin N(v_a)^{(1)}]) \\
& \beta_5''^T\beta_3\alpha_2\alpha_3 x_s \\
= & 2P[s \in N(v_a)^{(1)}, s \notin N(v_b)^{(1)}]Q(s) \\
\leq & \frac{2|\overline{N(v)}|(n - |\overline{N(v)}|)}{n^2} \quad \text{(as } Q(s) \text{ has been normalized)} \\
< & \frac{2|\overline{N(v)}|}{n^2}
\end{aligned}
$$

where $N(v_a)^{(i)}$ denotes the set of nodes that are i-hop away from v_a, *e.g.*, $N(v_a)^{(1)}$ is in fact $N(v_a)$.

Then we carry on with Σ_2 when $i = 2$. If s is either v_a or v_b's 2-hop neighbor, but not both, Σ_2 appears to be not zero. Similar with $i = 1$, the probability for this case is $\frac{2|\overline{N(v)}|^2(n - |\overline{N(v)}|^2)}{n^2}$. Besides, if s appears to be the instant neighbor of v_a or v_b, but not both, Σ_2 also appears positive. The probability for this case is the same with $i = 1$, namely $\frac{2|\overline{N(v)}|(n - |\overline{N(v)}|)}{n^2}$. Then, the expectation of Σ_2 is as follows.

$$
\begin{aligned}
E[\Sigma_2] = & 2P[s \in N(v_a)^{(2)}, s \notin N(v_b)^{(2)}]Q(s) \\
& + 2P[s \in N(v_a)^{(1)}, s \notin N(v_b)^{(1)}] \sum_{v \in N(s)^{(1)}} Q(v) \\
= & \frac{2|\overline{N(v)}|^2(n - |\overline{N(v)}|^2)}{n^2}Q(s) \\
& + \frac{2|\overline{N(v)}|(n - |\overline{N(v)}|)}{n^2} \sum_{v \in N(s)^{(1)}} Q(v) \\
\leq & \frac{2|\overline{N(v)}|^2(n - |\overline{N(v)}|^2)}{n^2} + \frac{2|\overline{N(v)}|^2(n - |\overline{N(v)}|)}{n^2} \\
< & \frac{4|\overline{N(v)}|^2}{n^2}.
\end{aligned}
$$

Following the same way, we can derive $E[\Sigma_3] < \frac{6|\overline{N(v)}|^3}{n^2}$ and $E[\Sigma_4] < \frac{8|\overline{N(v)}|^4}{n^2}$.

In fact, it is obvious that $E[|(Q_A - Q_B) - (Q'_A - Q'_B)|] \leq \sum_{i=1}^{I} E[\Sigma_i]$ and $\forall I < J, \sum_{i=1}^{I} E[\Sigma_i] < \sum_{i=1}^{J} E[\Sigma_i]$. That is, $\forall I < 4$, $E[|(Q_A - Q_B) - (Q'_A - Q'_B)|] < \sum_{i=1}^{4} E[\Sigma_i]$.

Finally, we can prove that $\forall I < 5, E[|(Q_A - Q_B) - (Q'_A - Q'_B)|] < \sum_{i=1}^{4} \frac{2i|\overline{N(v)}|^i}{n^2}$.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. 2017. Watch Your Step: Learning Graph Embeddings Through Attention. *CoRR* abs/1710.09599 (2017).

[2] Nesreen K. Ahmed, Jennifer Neville, and Ramana Rao Kompella. 2013. Network Sampling: From Static to Streaming Graphs. *TKDD* 8, 2 (2013), 7:1–7:56.

[3] Khurshed Ali, Chih-Yu Wang, and Yi-Shin Chen. 2018. Boosting Reinforcement Learning in Competitive Influence Maximization with Transfer Learning. In *WI*. 395–400.

[4] Akhil Arora, Sainyam Galhotra, and Sayan Ranu. 2017. Debunking the Myths of Influence Maximization: An In-Depth Benchmarking Study. In *SIGMOD*. 651–666.

[5] Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*. 585–591.

[6] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Brendan Lucier. 2014. Maximizing Social Influence in Nearly Optimal Time. In *SODA*. 946–957.

[7] Tianyu Cao, Xindong Wu, Xiaohua Tony Hu, and Song Wang. 2011. Active Learning of Model Parameters for Influence Maximization. In *PKDD*. 280–295.

[8] Tim Carnes, Chandrashekhar Nagarajan, Stefan M. Wild, and Anke van Zuylen. 2007. Maximizing influence in a competitive social network: a follower's perspective. In *EC*. 351–360.

[9] Wei Chen, Yajun Wang, and Siyu Yang. 2009. Efficient influence maximization in social networks. In *KDD*. 199–208.

[10] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *ICML*. 2702–2711.

[11] Christian Doerr and Norbert Blenn. 2013. Metric convergence in social network sampling. In *HotPlanet@SIGCOMM*. 45–50.

[12] Leo A Goodman. 1961. Snowball Sampling. *The Annals of Mathematical Statistics* 32, 1 (1961), 148–170.

[13] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. 2011. SIMPATH: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model. In *ICDM*. 211–220.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.

[15] Kai Han, Keke Huang, Xiaokui Xiao, Jing Tang, Aixin Sun, and Xueyan Tang. 2018. Efficient Algorithms for Adaptive Influence Maximization. *PVLDB* 11, 9 (2018), 1029–1040.

[16] Huimin Huang, Zaiqiao Meng, and Shangsong Liang. 2020. Recurrent neural variational model for follower-based influence maximization. *Inf. Sci.* 528 (2020), 280–293.

[17] Keke Huang, Sibo Wang, Glenn S. Bevilacqua, Xiaokui Xiao, and Laks V. S. Lakshmanan. 2017. Revisiting the Stop-and-Stare Algorithms for Influence Maximization. *PVLDB* 10, 9 (2017), 913–924.

[18] Peter Jackson. 1999. *Introduction to Expert Systems, 3rd Edition.* Addison-Wesley.

[19] Qingye Jiang, Guojie Song, Gao Cong, Yu Wang, Wenjun Si, and Kunqing Xie. 2011. Simulated Annealing Based Influence Maximization in Social Networks. In *AAAI.*

[20] Harshavardhan Kamarthi, Priyesh Vijayan, Bryan Wilder, Balaraman Ravindran, and Milind Tambe. 2020. Influence Maximization in Unknown Social Networks: Learning Policies for Effective Graph Sampling. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020,* Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith (Eds.). International Foundation for Autonomous Agents and Multiagent Systems, 575–583.

[21] David Kempe, Jon M. Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *KDD.* 137–146.

[22] David Kempe, Jon M. Kleinberg, and Éva Tardos. 2005. Influential Nodes in a Diffusion Model for Social Networks. In *ICALP.* 1127–1138.

[23] Jun-Sik Kim, Seokju Lee, Tae-Hyun Oh, and In So Kweon. 2018. Co-Domain Embedding Using Deep Quadruplet Networks for Unseen Traffic Sign Recognition. In *AAAI.* 6975–6982.

[24] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. 2006. Structure and evolution of online social networks. In *KDD.* 611–617.

[25] Chul-Ho Lee, Xin Xu, and Do Young Eun. 2012. Beyond random walk and metropolis-hastings samplers: why you should not backtrack for unbiased graph sampling. In *SIGMETRICS.* ACM, 319–330.

[26] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *KDD.* 631–636.

[27] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *KDD.* 631–636.

[28] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. 2007. Cost-effective outbreak detection in networks. In *KDD.* 420–429.

[29] Hui Li, Sourav S. Bhowmick, Jiangtao Cui, Yunjun Gao, and Jianfeng Ma. 2015. GetReal: Towards Realistic Selection of Influence Maximization Strategies in Competitive Networks. In *SIGMOD.* 1525–1537.

[30] Su-Chen Lin, Shou-De Lin, and Ming-Syan Chen. 2015. A Learning-based Framework to Handle Multi-round Multi-party Influence Maximization on Social Networks. In *KDD.* 695–704.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[33] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. 2016. Stop-and-Stare: Optimal Sampling Algorithms for Viral Marketing in Billion-scale Networks. In *SIGMOD.* 695–710.

[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD.* ACM, 701–710.

[35] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. DeepInf: Social Influence Prediction with Deep Learning. In *KDD.* 2110–2119.

[36] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, Sep (2011), 2539–2561.

[37] Lichao Sun, Weiran Huang, Philip S. Yu, and Wei Chen. 2018. Multi-Round Influence Maximization. In *KDD.* 2249–2258.

[38] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press.

[39] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence Maximization in Near-Linear Time: A Martingale Approach. In *SIGMOD.* 1539–1554.

[40] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence maximization: near-optimal time complexity meets practical efficiency. In *SIGMOD.* 75–86.

[41] Shan Tian, Songsong Mo, Liwei Wang, and Zhiyong Peng. 2020. Deep Reinforcement Learning-Based Approach to Tackle Topic-Aware Influence Maximization. *Data Sci. Eng.* 5, 1 (2020), 1–11.

[42] Xu Tong, Hao Fan, Xiaofei Wang, Jianxin Li, and Xin Wang. 2020. Seeds Selection for Influence Maximization Based on Device-to-Device Social Knowledge by Reinforcement Learning. In *KSEM.* Springer, 155–167.

[43] Sharan Vaswani, Branislav Kveton, Zheng Wen, Mohammad Ghavamzadeh, Laks V. S. Lakshmanan, and Mark Schmidt. 2017. Model-Independent Online Learning for Influence Maximization. In *ICML.* 3530–3539.

[44] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *KDD.* 1225–1234.