Revisiting Adversarially Learned Injection Attacks Against Recommender Systems

Jiaxi Tang*
Simon Fraser University
British Columbia, Canada
jiaxit@sfu.ca

Hongyi Wen
Cornell Tech, Cornell University
New York, NY, USA
hw557@cornell.edu

Ke Wang
Simon Fraser University
British Columbia, Canada
wangk@cs.sfu.ca

ABSTRACT

Recommender systems play an important role in modern information and e-commerce applications. While increasing research is dedicated to improving the relevance and diversity of the recommendations, the potential risks of state-of-the-art recommendation models are under-explored, that is, these models could be subject to attacks from malicious third parties, through injecting fake user interactions to achieve their purposes. This paper revisits the adversarially-learned injection attack problem, where the injected fake user 'behaviors' are learned locally by the attackers with their own model – one that is potentially different from the model under attack, but shares similar properties to allow attack transfer. We found that most existing works in literature suffer from two major limitations: (1) they do not solve the optimization problem precisely, making the attack less harmful than it could be, (2) they assume perfect knowledge for the attack, causing the lack of understanding for realistic attack capabilities. We demonstrate that the exact solution for generating fake users as an optimization problem could lead to a much larger impact. Our experiments on a real-world dataset reveal important properties of the attack, including attack transferability and its limitations. These findings can inspire useful defensive methods against this possible existing attack.

CCS CONCEPTS

• **Information systems → Recommender systems**; • **Security and privacy → Web application security**.

KEYWORDS

Recommender System; Adversarial Machine Learning; Security and Privacy

ACM Reference Format:
Jiaxi Tang, Hongyi Wen, and Ke Wang. 2020. Revisiting Adversarially Learned Injection Attacks Against Recommender Systems. In *Fourteenth ACM Conference on Recommender Systems (RecSys '20), September 22–26, 2020, Virtual Event, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3383313.3412243

1 INTRODUCTION

A good recommender system is a key factor to users' information seeking experience as it enables better content discovery and more accurate information retrieval. Over the past decade, most work aims to improve the utility/accuracy of recommendation models. Many methods have been developed, such as neighborhood-based methods [38], factorization-based approaches [23, 37], and the more recent deep neural network (*a.k.a* deep learning) models [9, 19]. However, due to the reliance on user contributed judgments and subjective rating data [6], recommender systems can be misused and attacked with malicious purposes. Once this happens, the credibility of a recommender system will be largely affected, which could lead to a significant economic loss.

1.1 Injection Attack against Recommender Systems

In this work, we focus on *injection attack* (a.k.a. data poisoning attack) as illustrated in Fig. 1, where the malicious party has knowledge about the data used by a recommender system (*e.g.*, by crawling the publicly available data) and creates fake user profiles with carefully chosen item preferences (*e.g.,* clicks) to influence the recommender with malicious goals. Assuming such knowledge about the data is reasonable, for example: users' ratings and reviews on Amazon's product are public[1], which account for the personalized product recommendations; users' social relationship (followings and followers) on Twitter is public, which influence friend recommendations; users' answers on questions and upvotes on answers are public on Quora, which constitute the personalized feed. As long as there are enough incentives, the availability of a dataset from certain platforms is *not unreachable* to the malicious party. It was reported that every one-star increase in user ratings on certain product can lead to a 5 to 9 percent increase in product seller's revenue[2]. Therefore, malicious injection attacks are easily motivated and can have huge consequences for a company's bottom line.

In the literature, early studies on performing injection attack against recommender systems are inspired by heuristics. To boost certain item(s)' availability of being recommended, Lam and Riedl [25] proposed to give high ratings for the targeted item and average ratings on other random items; Burke et al. [5] further considered to use popular items instead of random items to ensure that fake users can have more neighborhoods thus have more impacts. However, since these attacks were created heuristically, the threat of injection attacks may not be fully realized. First, the fake users generated

*Now at Google Inc., work done when he was a student at Simon Fraser University.

RecSys '20, September 22–26, 2020, Virtual Event, Brazil

ACM ISBN 978-1-4503-7583-2/20/09…$15.00
https://doi.org/10.1145/3383313.3412243

[1] http://jmcauley.ucsd.edu/data/amazon
[2] https://www.forbes.com/sites/ryanerskine/2018/05/15/you-just-got-attacked-by-fake-1-star-reviews-now-what

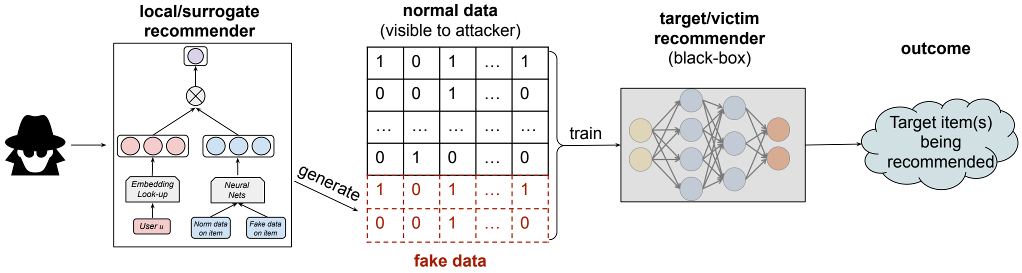


Figure 1: An illustration of the threat model for injection attack against recommendation models. This assumes the dataset is available to the attacker but the target model is unknown. To achieve their malicious goals, e.g., influencing certain item(s)' availability of being recommended, the attacker will craft fake user profiles locally with a surrogate model and inject them to the target recommender before it is trained.

are highly correlated with each other and sometimes even self-forming clusters [6], making them easily detectable by standard techniques [28]. What's more, heuristic methods heavily rely on background knowledge, hence, a method designed for one malicious purpose is hard to be used for another. Finally, heuristic methods do not directly optimize the adversarial goals, which limits their usability and threat.

Recently, we have witnessed a huge impact of adversarial attacks through adversarial machine learning that optimizes an adversarial objective, irrespective of the model type and tasks [29]: In web search, an adversary can change web contents to get high search engine's rankings [7]; In crowd-sourcing, an adversary can provide useless answers for profits [30]; In social networks, an adversary can modify node relationships for a desired node property [43]; In image recognition, an adversary can make perturbations on image pixels and have a wanted recognition result [17, 24].

Despite the success of adversarial learning in other domains, there's very sparse research on adopting adversarial learning to attack recommender systems. In the security arms race, a limited knowledge of the attack leads to a more dangerous state of existing systems. In this work, we aim to revisit this direction by investigating the challenges and limitations of using adversarial learning to attack recommender systems. In the next two sections, we will have the same viewpoint as adversaries to understand how the attack can be performed. After defining the threat model in Section 2, we found that existing works do not solve the problem properly, causing the attack less powerful than it could have been. In Section 3, we propose a more precise but less efficient solution to generate the attack, accompanied by two efficient approximations to the solution. In Section 4, we explore the attack's impact on a real-world dataset and identify its weakness and find the clues of such attacks. We hope that these findings can help us better understand the attack thus develop defensive techniques.

2 BACKGROUND

In this section, we first cover essential basics of the recommendation task and define the threat model. Some notations will also be introduced to facilitate the presentation. Then we briefly revisit existing solutions and their limitations, which encourage us to propose a more precise approach in the next section.

2.1 Recommendation Goal

The goal of system under-attack is to recommend relevant items to cater users' needs. In such a system, there is a set of users $\mathcal{U} = \{u_1, u_2, .., u_{|\mathcal{U}|}\}$, a set of items $\mathcal{I} = \{i_1, i_2, .., i_{|\mathcal{I}|}\}$ (e.g., products, videos, venues, etc.), and feedback/interaction data (e.g., user purchased a product, watched a video, checked-in at a venue). The feedback that users left in the system can have different types. It can be either *explicit* (i.e., the user explicitly shows how she/he likes an item, such as giving a five star rating) or *implicit* (i.e., a signal implicitly reflect user's positive preferences, such as purchasing a product), but the later is much more prevalent than the explicit feedback in real systems [21, 32] thus is considered in this work. We use $\boldsymbol{X} \in \{0, 1\}^{|\mathcal{U}| \times |\mathcal{I}|}$ to denote the binarized implicit data, with 1 for a positive feedback and 0 for an unknown entry. A recommendation model built on users' historical data can make predictions $\boldsymbol{R} \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{I}|}$. The learning objective of a recommendation model is to provide relevant items of each user with the highest predicted relevance scores.

2.2 Threat Model

The threat model is illustrated in Figure 1. To attack the target recommender deployed in the system (a.k.a. *victim model*), the attacker will use their own local model (a.k.a. *surrogate model*), to craft fake users and inject them into the original training data of victim model. Below, we elaborate the threat model from several perspectives.

Attacker's goal. The adversary's goal can be either *non-targeted*, aiming to hamper the effectiveness of the recommendation model by forcing it to make bad recommendations, or *targeted*, where the adversary wishes to increase or decrease a target item(s)' availability of being recommended. Similar to most other works [5, 13, 14, 25] in literature, we mainly focus on *targeted attack*, which is the most common case under recommendation context: attackers want to influence normal users' recommendations for profits. Specifically to targeted attack, we consider the *promote (or push) attack*: given a target item, attacker's goal is to increase its chance of being recommended by victim model. Alternatively, there is *nuke attack*, where attackers aim to "nuke" a target item, make it less able to get recommended. Although we don't explicitly discuss the nuke attack in this work, similar techniques can be used.

Attacker's knowledge. We assume the attacker has (full or partial) knowledge about the dataset used to train the target (victim) recommendation model. Because user feedback is public in many systems (e.g., followings and followers on Twitter), this is a reasonable assumption for a worst-case attack. Any knowledge about victim model (e.g., model parameters, model type, etc) is optional, because attackers can first attack their own local (surrogate) model with the poison fake users, hopeing that these fake users can be also used to attack the target (victim) model. This kind of *attack transfer* is possible if two models share similar properties [33].

Attacker's capability. As shown in Fig. 1, the attacker will learn fake users through a surrogate model and inject their rating profiles to the training set of the victim model. The attacker will achieve their malicious goal after the victim model consumes these fake users. This suggests that the attack happens at training time of the victim model, instead of test time. The later is known as evasion attack and is more commonly studied in adversarial learning [17, 24, 33, 34]. While in recommendation, test time attack requires attackers to hack into other normal users' accounts and change their preferences, which is a cybersecurity issue and is beyond the scope of our work. On the other hand, allowing adversaries to create fake users and let them be consumed (trained) by the victim model is a more practical attack scenario.

2.3 Adversarial Injection Attack: a bi-level optimization problem

Different from heuristic approaches [5, 25], the injection attack considered in this paper directly *learns* fake user behaviors as an optimization problem. Recall that the adversaries will learn fake users to attack their own surrogate model as a first step. Given a well-trained surrogate model that is under-attack and a set of fake users $\mathcal{V} = \{v_1, v_2, .., v_{|\mathcal{V}|}\}$, the fake data $\widehat{X} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{I}|}$ will be learned to optimize an adversarial objective function $\mathcal{L}_{\text{adv}}$

$$\min_{\widehat{X}} \ \mathcal{L}_{\text{adv}}(\boldsymbol{R}_{\theta^*}), \tag{1}$$

$$\text{subject to} \quad \theta^* = \arg\min_{\theta} \left(\mathcal{L}_{\text{train}}(\boldsymbol{X}, \boldsymbol{R}_\theta) + \mathcal{L}_{\text{train}}(\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{R}}_\theta)\right), \tag{2}$$

where θ denotes a set of surrogate model's parameters, $\boldsymbol{R}_\theta$ is surrogate model's predictions on normal users with parameter θ and $\mathcal{L}_{\text{train}}$ denotes surrogate model's training objective. As shown, one optimization problem (i.e., Eq. (2), called *inner objective*) is embedded (nested) within another (i.e., Eq. (1), called *outer objective*), this forms a bi-level optimization problem [12]. In machine learning, the bi-level optimization has been considered for hyperparamter optimization [8, 16], few-shot learning and meta-learning [15, 18, 36]. We formulate the learned injection attack as a bi-level optimization problem because of the definition of the threat model introduced in Section 2.2, although we found this formulation is not explicitly discussed in literature. The inner objective[3] shows that after fake data $\widehat{\boldsymbol{X}}$ are injected, the surrogate model will first consume them (i.e., train from scratch with the poisoned dataset), we then obtain the trained model parameters θ^*. The outer objective shows that after fake data are consumed, we can achieve the malicious goal defined on normal user's predictions $\boldsymbol{R}_{\theta^*}$. Note that different adversarial objectives can be used for different malicious purposes. In this paper, we focus on promoting target item k to all normal users, so an exemplary adversarial objective can be the cross-entropy loss:

$$\mathcal{L}_{\text{adv}}(\boldsymbol{R}) = -\sum_{u \in \mathcal{U}} \log\left(\frac{\exp(r_{uk})}{\sum_{i \in \mathcal{I}} \exp(r_{ui})}\right). \tag{3}$$

The objective will be minimized if normal user's prediction on target item k is greater than other items, so that the malicious goal of promoting target item will be achieved, as a result.

Algorithm 1 Learning fake user data with Gradient Descent

```
1:  Input: Normal user data $X$; learning rate for inner and outer objective: $\alpha$ and $\eta$; max iteration for inner and outer objective: $L$ and $T$.
2:  Output: Learned fake user data for malicious goal.
3:  Initialize fake data $\widehat{X}^{(0)}$ and surrogate model parameters $\theta^{(0)}$
4:  for $t = 1$ to $T$ do
5:      for $l = 1$ to $L$ do
6:          Optimize inner objective with SGD: $\theta^{(l)} \leftarrow \theta^{(l-1)} - \alpha \cdot \nabla_\theta\big(\mathcal{L}_{\text{train}}(X, R_{\theta^{(l-1)}}) + \mathcal{L}_{\text{train}}(\widehat{X}^{(t)}, \widehat{R}^{(t)}_{\theta^{(l-1)}})\big)$
7:      end for
8:      Evaluate $\mathcal{L}_{\text{adv}}(R_{\theta^{(L)}})$ and compute gradients $\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}}$
9:      Update fake data: $\widehat{X}^{(t)} = \text{Proj}_\Lambda\big(\widehat{X}^{(t-1)} - \eta \cdot \nabla_{\widehat{X}}\mathcal{L}_{\text{adv}}\big)$
10: end for
11: Return: $\widehat{X}^{(T)}$
```

To solve the bi-level optimization problem in Eqs. (1) to (2), one could try every possible $\widehat{X}$, obtain the associated θ^* and evaluate $\mathcal{L}_{\text{adv}}(\boldsymbol{R}_{\theta^*})$. But the search space is exponentially large as $2^{|\mathcal{V}| \times |\mathcal{I}|}$. So this brute-force approach can hardly be used with limited resources. A more computationally-efficient way is to use gradient-based approaches, such as Gradient Descent, to iteratively update fake data $\widehat{X}$ with gradient $\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}}$, which we formally present in Algorithm 1. At each iteration $t \in \{1, ..., T\}$ for updating fake data, we first retrain the surrogate model by performing parameter updates for L iterations (line 7). Then we update fake data (line 10) with Projected Gradient Descent (PGD), with $\text{Proj}_\Lambda(\cdot)$ as the projection operator that projects the fake data onto feasible set (i.e., $\hat{x}_{vi} \in \{0,1\}$ in our case). After the final iteration T where fake data $\widehat{X}^{(T)}$ is learned to minimize $\mathcal{L}_{\text{adv}}$, they are able to attack the surrogate model, and we hope they can also attack the target (victim) model in a similar way: once trained on the poisoned dataset, it will cause a small adversarial loss $\mathcal{L}_{\text{adv}}$.

2.4 Limitations in Existing Studies and Our Contributions

There are a few studies [11, 13, 14, 26] in the literature tried to regard the injection attack as an optimization problem and learn fake data for adversarial goals. However, we found there exist two major limitations in existing works. In below, we illustrate these limitations, as well as our contributions in this work.

Lacking exactness in gradient computation. As we can see from Algorithm 1, solving the inner objective for surrogate model training

[3] Note that although here we separate the training objective for normal data and for fake data to have a clear presentation, in reality, these two data are arbitrarily mixed together and cannot be distinguished.

is simple and conventional, while the challenge comes from obtaining the adversarial gradient $\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}}$ to update fake data. In the literature, existing works either tried to estimate this gradient [11], or tried to directly compute it [13, 14, 26]. But under the problem formulation in Eqs. (1) to (2), they all lack exactness in gradient computation. More specifically, by applying chain rule, the exact adversarial gradient can be written as:

$$\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}} = \frac{\partial \mathcal{L}_{\text{adv}}}{\partial \widehat{X}} + \frac{\partial \mathcal{L}_{\text{adv}}}{\partial \theta^*} \cdot \frac{\partial \theta^*}{\partial \widehat{X}}. \tag{4}$$

The first part (partial derivative $\partial \mathcal{L}_{\text{adv}}/\partial \widehat{X}$) assumes $\widehat{X}$ is independent to other variables, while the second part suggests θ^* can be also a function containing $\widehat{X}$. Among all existing studies [11, 13, 14, 26], we found the second part in Eq. (4) has been completely ignored. This suggests the final surrogate model parameters θ^* is independent from fake data $\widehat{X}$, but it is obviously incorrect. In Section 3, we show the first part doesn't exist in many surrogate models and when it exists, the second part also contributes significantly to the total gradient. This suggests the approximation of gradient $\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}}$ with its partial derivative is largely biased, therefore can lead to sub-optimal solutions.

Our contributions. In Section 3, we present the computation of exact adversarial gradient in Eq. (4) and show two efficient ways to approximate this gradient. On a synthetic and a real-world dataset, we empirically demonstrate the effectiveness of both approaches and the undesirable results of only computing the partial derivative for approximation, as used in existing works [11, 13, 14, 26].

Lacking vital experimental studies. Another major limitation in all previous experimental studies [11, 13, 14, 26], is that the target model is set to be identical to the surrogate model, which is known as "white-box attack". One could think this is an extreme case where target model is visible to the adversary and this can serve as a upper bound of the attack capability. Our analysis shows that this attack may not be carried over to the realistic case where the target model is different from the surrogate model. Knowing this, the attacker would design the more effective attack by learning fake user data using a surrogate model that is tranferable to a different target model. The ultimate goal is to defend against attacks, which requires considering more practical settings and understanding of the attack's characteristics and limitations, in order to inspire better defensive strategies.

Our contributions. In Section 4, we leverage a user-venue check-ins dataset to study how attack crafted from one surrogate model can transfer to another victim models and examine the key factors that influence the transferability. More importantly, we analyze the limitations of this adversarially-learned injection attack, which could inspire useful defensive techniques.

3 SOLVING THE BI-LEVEL OPTIMIZATION PROBLEM

In this section, we focus on line 9 of Algorithm 1 and describe how to compute the adversarial gradient in Eq. (4) exactly (in Section 3.1) and provide two approximated solutions (in Section 3.2 and Section 3.3). First of all, we will use Weighted Regularized Matrix Factorization (WRMF) [21, 32], a fundamental and representative factorization-based model for recommendations with implicit feedback, as an example of the surrogate model and demonstrate how the exact adversarial gradient can be computed. However, the exact gradient computation is neither time-efficient nor resource-efficient. We thus introduce two orthogonal ways to approximate the gradient computation to achieve a good balance between effectiveness and efficiency. On a synthetic toy dataset, we empirically evaluate how good are the approximated solutions.

Before diving into details, we briefly introduce the WRMF model and the toy dataset used in this section. In WRMF, a set of user latent factors $\boldsymbol{P} \in \mathbb{R}^{|\mathcal{U}|\times K}$ and item latent factors $\boldsymbol{Q} \in \mathbb{R}^{|\mathcal{I}|\times K}$ are used to make predictions $\boldsymbol{R} = \boldsymbol{P}\boldsymbol{Q}^\top$ on normal data $\boldsymbol{X}$. When fake data $\widehat{\boldsymbol{X}}$ are injected, let $\boldsymbol{F} \in \mathbb{R}^{|\mathcal{V}|\times K}$ denotes the user latent factors and $\widehat{\boldsymbol{R}} = \boldsymbol{F}\boldsymbol{Q}^\top$ denotes the predictions for fake users. Under this formulation, $\theta = \{\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{F}\}$ and the surrogate training objective is:

$$\begin{aligned} &\mathcal{L}_{\text{train}}(\boldsymbol{X}, \boldsymbol{R}_\theta) + \mathcal{L}_{\text{train}}(\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{R}}_\theta) = \\ &\qquad \sum_{u,i} w_{ui}(x_{ui} - P_u^\top Q_i)^2 + \sum_{v,i} w_{vi}(\hat{x}_{vi} - F_v^\top Q_i)^2 \\ &\qquad + \lambda\left(\|\boldsymbol{P}\|^2 + \|\boldsymbol{F}\|^2 + \|\boldsymbol{Q}\|^2\right), \end{aligned} \tag{5}$$

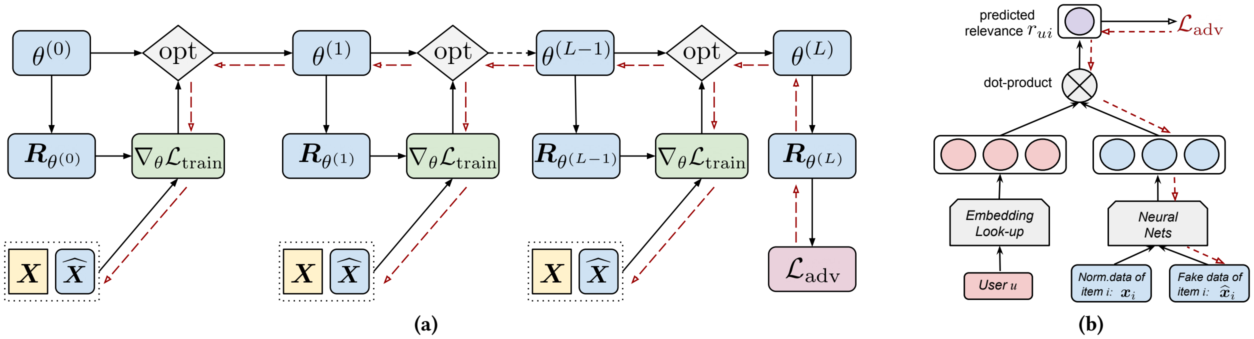
where w_{ui} and w_{vi} are instance weights to differentiate observed and missing feedback from the normal and fake data, respectively (e.g., $w_{ui} = 2$ when $x_{ui} = 1$ and $w_{ui} = 1$ when $x_{ui} = 0$, similar for w_{vi}), λ is the hyperparameter to control model complexity.

Synthetic data. To facilitate our understanding of exact and approximated solutions for computing adversarial gradient, we synthesize a toy datset that is more controllable. Specifically, each data point in $\boldsymbol{X}$ is generated by $x = \boldsymbol{\mu}\boldsymbol{\nu}^\top$, where both $\boldsymbol{\nu} \in \mathbb{R}^d$ and $\boldsymbol{\mu} \in \mathbb{R}^d$ are sampled from $\mathcal{N}_d(\boldsymbol{0}, \mathbf{I})$ with $d << \min(|\mathcal{U}|, |\mathcal{V}|)$. By generating data point for $\forall x \in \boldsymbol{X}$, the synthesized dataset is presumably to have low-rank, similar to other real-world recommendation datsets. Lastly, we binarize $\boldsymbol{X}$ to transform it into implicit feedback data by setting a threshold ϵ. By controlling the value of $(|\mathcal{U}|, |\mathcal{V}|, d, \epsilon)$, we are able to have arbitrary-size synthetic datasets with different ranks and sparsity levels.

3.1 Exact Solution

In this subsection, we compute the exact adversarial gradient $\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}}$ when WRMF is used as the surrogate model. It's worth noting that, WRMF's predictions on normal users $\boldsymbol{R} = \boldsymbol{P}\boldsymbol{Q}^\top$ are not directly dependent on fake data $\widehat{\boldsymbol{X}}$, therefore $\mathcal{L}_{\text{adv}}(\boldsymbol{R})$ is not differentiable w.r.t. $\widehat{\boldsymbol{X}}$ or equivalently, we can specify $\frac{\partial \mathcal{L}_{\text{adv}}}{\partial \widehat{X}} = 0$. In fact, this is a common case for most embedding-based recommendation models [19, 20, 37], in which data ($\boldsymbol{X}$ and $\widehat{\boldsymbol{X}}$) are only used for computing training objective (as labels), and are not involved in computing predictions ($\boldsymbol{R}$ and $\widehat{\boldsymbol{R}}$).

Without loss of generality, we assume the inner objective is optimized for only once (i.e., $L = 1$), then $\theta^{(1)} = \{\boldsymbol{P}^{(1)}, \boldsymbol{Q}^{(1)}, \boldsymbol{F}^{(1)}\}$ is the final parameter set used in adversarial objective $\mathcal{L}_{\text{adv}}(\boldsymbol{R}_{\theta^{(1)}}) = \mathcal{L}_{\text{adv}}(\boldsymbol{P}^{(1)} \cdot \boldsymbol{Q}^{(1)})$. Also, we can formulate $\theta^{(1)} = \mathbf{opt}(\theta^{(0)}, \nabla_\theta \mathcal{L}_{\text{train}})$, here $\mathbf{opt}(,)$ denotes the transition function from $\theta^{(l-1)}$ to $\theta^{(l)}$. As in Algorithm 1, under the context of Stochastic Gradient Descent


$\theta^{(0)}$
opt
$\theta^{(1)}$
opt
$\theta^{(L-1)}$
opt
$\theta^{(L)}$
$R_{\theta^{(0)}}$
$\nabla_\theta \mathcal{L}_{\text{train}}$
$R_{\theta^{(1)}}$
$\nabla_\theta \mathcal{L}_{\text{train}}$
$R_{\theta^{(L-1)}}$
$\nabla_\theta \mathcal{L}_{\text{train}}$
$R_{\theta^{(L)}}$
X $\widehat{X}$
X $\widehat{X}$
X $\widehat{X}$
$\mathcal{L}_{\text{adv}}$
predicted relevance r_{ui}
$\mathcal{L}_{\text{adv}}$
dot-product
Embedding Look-up
Neural Nets
User u
Norm.data of item i: x_i
Fake data of item i: $\widehat{x}_i$

(a) (b)

Figure 2: (a) Computational graph for a single calculation of adversarial objective $\mathcal{L}_{\text{adv}}$ with surrogate model. We use solid black arrow to denote the forward computation flow and use dashed red arrow to denote the gradient backpropagation flow, from $\mathcal{L}_{\text{adv}}$ to $\widehat{X}$. (b) The proposed surrogate model in this paper, the model can be also viewed as item-based autoencoder.

(SGD), this will become

$$\begin{aligned}\theta^{(1)} &= \mathbf{opt}\Big(\theta^{(0)}, \nabla_\theta\big(\mathcal{L}_{\text{train}}(X, R_{\theta^{(0)}}) + \mathcal{L}_{\text{train}}(\widehat{X}, \widehat{R}_{\theta^{(0)}})\big)\Big)\\ &= \theta^{(0)} - \alpha \cdot \nabla_\theta\big(\mathcal{L}_{\text{train}}(X, R_{\theta^{(0)}}) + \mathcal{L}_{\text{train}}(\widehat{X}, \widehat{R}_{\theta^{(0)}})\big).\end{aligned}$$

Now we can easily compute the adversarial gradient $\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}}$ when using WRMF and $T = 1$, by applying chain rule:

$$\begin{aligned}\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}} &= \frac{\partial\mathcal{L}_{\text{adv}}}{\partial\widehat{X}} + \frac{\partial\mathcal{L}_{\text{adv}}}{\partial\theta^*}\cdot\frac{\partial\theta^*}{\partial\widehat{X}} = 0 + \frac{\partial\mathcal{L}_{\text{adv}}}{\partial\theta^{(1)}}\cdot\frac{\partial\theta^{(1)}}{\partial\widehat{X}}\\ &= \frac{\partial\mathcal{L}_{\text{adv}}}{\partial\theta^{(1)}}\cdot\Big(-\alpha\nabla_{\widehat{X}}\nabla_\theta\big(\mathcal{L}_{\text{train}}(X, R_{\theta^{(0)}}) + \mathcal{L}_{\text{train}}(\widehat{X}, \widehat{R}_{\theta^{(0)}})\big)\Big).\end{aligned}\tag{6}$$

Similarly, when $T > 1$, we just need to accumulate the gradient

$$\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}} = \sum_{l\in[1,L]}\frac{\partial\mathcal{L}_{\text{adv}}}{\partial\theta^{(l)}}\cdot\frac{\partial\theta^{(l)}}{\partial\widehat{X}}.\tag{7}$$

In the above summation, $\frac{\partial\theta^{(l)}}{\partial\widehat{X}}$ is trivial to obtain, as shown in Eq. (6), while $\frac{\partial\mathcal{L}_{\text{adv}}}{\partial\theta^{(l)}}$ can be done in a sequential manner. That is, after having $\frac{\partial\mathcal{L}_{\text{adv}}}{\partial\theta^{(l+1)}}$ we can acquire $\frac{\partial\mathcal{L}_{\text{adv}}}{\partial\theta^{(l)}}$ by

$$\begin{aligned}&\frac{\partial\mathcal{L}_{\text{adv}}}{\partial\theta^{(l)}} = \frac{\partial\mathcal{L}_{\text{adv}}}{\partial\theta^{(l+1)}}\cdot\frac{\partial\theta^{(l+1)}}{\partial\theta^{(l)}}, \quad \text{where}\\ &\frac{\partial\theta^{(l+1)}}{\partial\theta^{(l)}} = \Big(1 - \alpha\nabla_\theta\nabla_\theta\big(\mathcal{L}_{\text{train}}(X, R_{\theta^{(l)}}) + \mathcal{L}_{\text{train}}(\widehat{X}, \widehat{R}_{\theta^{(l)}})\big)\Big).\end{aligned}\tag{8}$$

In Figure 2a, we show the overall computational graph of the procedure for calculating the adversarial gradient $\nabla_{\widehat{X}}\mathcal{L}_{\text{adv}}$. Note that this procedure applies to most embedding-based recommendation models, not limit to WRMF, if their inner objectives are also optimized with SGD (or variants of SGD, such as Adam [22]). It's worth mentioning that despite the differentiations showed in Eq. (6) and Eq. (8) contain the computation of Hessian matrix, *automatic differentiation* [18] provides us a convenient way to solve them.

To study the effectiveness of attack using the above exact solution, we design a proof-of-concept experiment on a synthetic data set to learn fake user data using WRMF as the surrogate model under a white-box attack setting.

Setup. Based on the synthetic data generation method we described previously, we create a toy dataset with 900 normal users, 100 fake users and 300 items. That is, $|\mathcal{U}| = 900$, $|\mathcal{V}| = 100$, and $|\mathcal{I}| = 300$ in the following experiments. When generating data, we set its 'rank' d to be 20 and binarization threshold ϵ to be 5, the data sparsity is 88% under this setting. The model we use is WRMF with latent dimensionality $K = 16$ and with its training objective showed in Eq. (5). In terms of instance weight, we empirically set $w_{ui} = 20$ when $x_{ui} = 1$ and $w_{ui} = 1$ when $x_{ui} = 0$, as this gives the best recommendation performance. Same weight also applies for w_{vi}. Finally, surrogate model (i.e., the inner objective) is optimized with Adam for 100 iterations ($L = 100$) and fake data (i.e., the outer objective) is optimized for 50 iterations ($T = 50$).

Implementation details. Recall that after updating the fake data with the exact adversarial gradients, we have to project them onto feasible region (i.e., $\hat{x}_{vi} \in \{0, 1\}$). To achieve this, the most straightforward way is using a threshold:

$$\text{Proj}_\Lambda(\text{x}) = \begin{cases} 1 & \text{if } x \geq \rho,\\ 0 & \text{else }.\end{cases}$$

Empirically, one can apply grid-search for the value of ρ for larger attack influence. In our experiments, we use $\rho = 0.2$.

Results. First of all, to evaluate the performance of the surrogate model as well as the performance of learned injection attack, we use *hit ratio* truncated at 10 (HR@10) as the metric. To evaluate surrogate model's recommendation performance, we randomly reserve 1 interacted item per user (denoted as *test item*), measure HR@10 on this test item and average it for all normal users. To evaluate the attack performance, we use the same strategy but measure HR@10 on *target item*. Fig. 3 shows the results for both performance we care about. From Fig. 3(a), we can see when using WRMF as the surrogate model and its exact adversarial gradients to solve the bi-level optimization problem in Eqs. (1) to (2), the adversarial objective $\mathcal{L}_{\text{adv}}$ is successfully minimized over iterations. From Fig. 3(b), we notice the attack performance (i.e., HR@10 on target item) is also getting better along with $\mathcal{L}_{\text{adv}}$. More interestingly, we found the attack doesn't change much to WRMF's recommendation performance (i.e., HR@10 on test item).

With the experiment showed above, we illustrate the effectiveness of the learned injection attack when (1) using WRMF with Adam as surrogate model and (2) using the exact adversarial gradients accumulated for all iterations $l \in \{1, .., L\}$ in inner objective optimization. Ideally, we can substitute WRMF with other type of

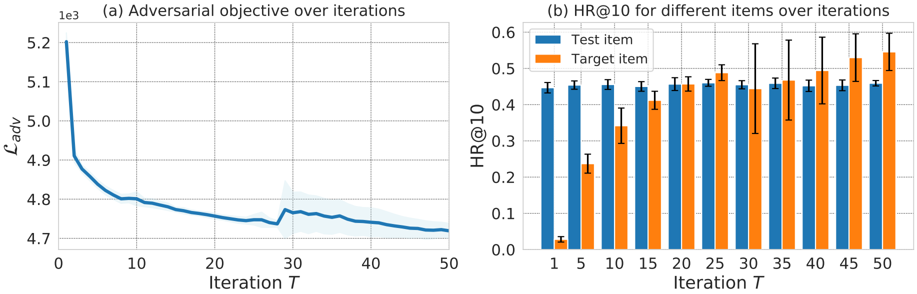


Figure 3: On our synthesized toy dataset, we report the mean and standard deviation for (a) adversarial objective and (b) HR@10 for test and target item over iteration T. The experiment is repeated for 5 times with different initializations.

surrogate models (or an ensemble of several models) to achieve the best attack performance against most victim models. This effective approach has been tried in other domains [31], but remains under-explored for injection attack against recommender systems.

Nevertheless, this method has a big drawback due to its high complexity in both time and space. Originally, when optimizing the inner objective, we perform forward and backward propagation to update surrogate model parameters θ, and we only keep the latest values of these parameters. While if we want to compute the exact gradient (as shown in Fig. 2a), we need to store the parameter values $\theta^{(l)}$ for each single iteration $l \in \{1, .., L\}$. So *the space complexity grows linearly with the number of total iterations L*. That is, if surrogate model size $|\theta| = m$, then a total of $O(Lm)$ space is needed. As for time complexity, we need extra time to compute $\frac{\partial \theta^{(l+1)}}{\partial \theta^{(l)}}$ and $\frac{\partial \theta^{(l)}}{\partial \widehat{X}}$ for each $l \in \{1, .., L\}$. According to the reverse-mode algorithmic differentiation [3], time complexities of both of these computations are proportional to the model size m. Thus *the time complexity also grows linearly with L*, and an additional $O(Lm)$ time is needed to have all the gradients accumulated, *for a single update of fake data*. As a result, computing the exact gradient $\nabla_{\widehat{X}} \mathcal{L}_{\text{adv}}$ is impractical when having a large surrogate model optimized for many iterations, which is very common in real-world recommendation scenarios. That's why in the next, we show two approximations of the exact gradient $\nabla_{\widehat{X}} \mathcal{L}_{\text{adv}}$.

3.2 Approximated Solution i: Unrolling fewer steps

A straightforward solution is unrolling fewer steps when accumulating $\frac{\partial \mathcal{L}_{\text{adv}}}{\partial \theta^{(l)}} \cdot \frac{\partial \theta^{(l)}}{\partial \widehat{X}}, \forall l \in \{L, L-1, ..., 1\}$. When computing the exact gradient with Eq. (7), one can sum the gradient from $l = L$ back to $l = L - \tau$, instead of $l = 1$. In other words, Eq. (7) will be approximated as:

$$\nabla_{\widehat{X}} \mathcal{L}_{\text{adv}} \approx \sum_{l \in [L-\tau, L]} \frac{\partial \mathcal{L}_{\text{adv}}}{\partial \theta^{(l)}} \cdot \frac{\partial \theta^{(l)}}{\partial \widehat{X}}, \tag{9}$$

where $\tau \in [1, L]$ denotes the unroll steps. This requires we keep surrogate model parameters only for the last τ steps, and backpropagate adversarial gradients only within last τ steps. Therefore, it reduces both time and space complexity from $O(Lm)$ to $O(\tau m)$. We can choose the unroll steps τ according to the available resources, but theoretically, a larger τ leads to a better approximation.

Results. In Fig. 4(a), we show the results of the same experiment in previous subsection, this time varying the number of unroll steps τ. When $\tau = 100$, we have the same results as in Fig. 3. When using WRMF (optimized with Adam) as surrogate model, we can get better attack performance (measured by a higher HR@10 for target item or a lower $\mathcal{L}_{\text{adv}}$) when we unroll more steps, as expected. Notably, even unrolling a few steps (e.g., %5 of total steps), we can achieve a reasonable attack performance with an approximation factor of 0.65 (0.368 vs. 0.568 in HR@10 for target item). Though promising results are achieved on this synthetic dataset, we'd like to point out the approximation factor is not guaranteed and can vary from different datasets and from different surrogate models.

3.3 Approximated Solution ii: Using special surrogate models

By unrolling fewer steps, we still need extra time and space. While in this subsection, we revisit another way adopted by existing works [13, 26], to approximate the gradient by its partial derivatives:

$$\nabla_{\widehat{X}} \mathcal{L}_{\text{adv}} \approx \frac{\partial \mathcal{L}_{\text{adv}}}{\partial \widehat{X}}. \tag{10}$$

Using this approximation, we don't need to unroll any step (or $\tau = 0$) and it requires almost no extra time or space.

Recall that adversarial objective $\mathcal{L}_{\text{adv}}$ is defined as a function of predictions on normal data $\boldsymbol{R}$ (see Eq. (3) for an example). And in Section 3.1, we equating $\frac{\partial \mathcal{L}_{\text{adv}}}{\partial \widehat{X}} = 0$, since when optimized with SGD-based approaches, WRMF's predictions are not depend on the data, neither $\boldsymbol{X}$ nor $\widehat{\boldsymbol{X}}$, but only dependent on its parameters $\boldsymbol{R} = \boldsymbol{P}^\top \boldsymbol{Q}$. However, Li et al. [26] showed when WRMF is optimized with alternating least square (ALS, or block coordinate descent), then its predictions are explicitly condition on the data, thus we can have non-zero partial derivatives. In short, when the training of WRMF with ALS is converged, its corresponding predictions for $\boldsymbol{R}$ and $\widehat{\boldsymbol{R}}$ are:

$$\begin{bmatrix} \boldsymbol{r}_i \\ \widehat{\boldsymbol{r}}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{P} \\ \boldsymbol{F} \end{bmatrix} \left(\begin{bmatrix} \boldsymbol{P} \\ \boldsymbol{F} \end{bmatrix}^\top \cdot \begin{bmatrix} \boldsymbol{P} \\ \boldsymbol{F} \end{bmatrix} + \lambda \boldsymbol{I} \right)^{-1} \begin{bmatrix} \boldsymbol{P} \\ \boldsymbol{F} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{x}_i \\ \widehat{\boldsymbol{x}}_i \end{bmatrix},$$

where $\boldsymbol{r}_i$ and $\widehat{\boldsymbol{r}}_i$ are the i-th column of $\boldsymbol{R}$ and $\widehat{\boldsymbol{R}}$, similar denotation also applies to $\boldsymbol{x}_i$ and $\widehat{\boldsymbol{x}}_i$. Holding $\boldsymbol{P}$ and $\boldsymbol{F}$ as constant, we can then compute the desired partial derivatives $\frac{\partial \mathcal{L}_{\text{adv}}}{\partial \widehat{X}} = \frac{\partial \mathcal{L}_{\text{adv}}}{\partial \boldsymbol{R}} \cdot \frac{\partial \boldsymbol{R}}{\partial \widehat{X}}$. Using the similar idea, Fang et al. [14] showed the partial derivatives exist when using random walk with restart (RWR) as surrogate

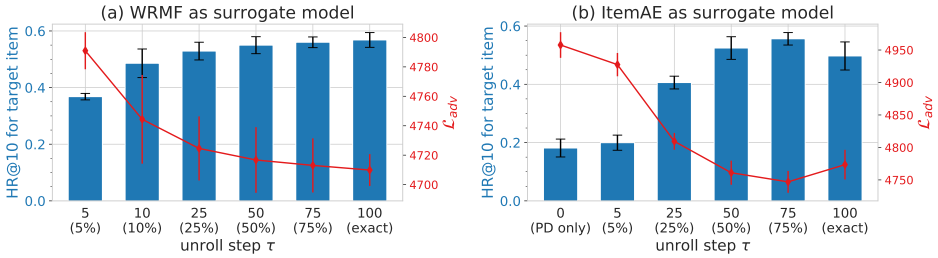


Figure 4: On our synthesized toy dataset, we vary the unroll step τ and report the mean and standard deviation of HR@10 for target item (blue bars) and adversarial objective (red lines). In subfigure (a), WRMF with Adam is used as the surrogate model, while in subfigure (b) ItemAE is used as the surrogate model. Experiments are repeated for 5 times with different initializations.

model. However, a limitation of these approaches is that both WRMF solved with ALS and RWR are not well-supported by existing machine learning libraries (such as TensorFlow [1] or PyTorch [35]), which are mostly designed for models optimized with SGD-based approaches. Therefore it's non-trivial to compute all the desired derivatives (both partial derivative in Eq. (10) and the accumulated gradients in Eq. (9)) with *automatic differentiation* [18].

In this paper, we offer another surrogate model. Acknowledging the limitation and inspired by previous works, we want the model predictions explicitly condition on data (in other words, $\boldsymbol{X}$ and $\widehat{\boldsymbol{X}}$ should be on the computational graph when calculating $\boldsymbol{R}$) and is optimized with SGD-based methods. Moreover, this surrogate model is composed with neural networks, thus hopefully, can better transfer attacks to other deep recommenders. The resulting surrogate model is shown in Fig. 2b. Similar to WRMF, we retrieve the user latent representations (red circles) from a user embedding table. While the item latent representations (blue circles) for item i are computed using a feed-forward layer with data $\boldsymbol{x}_i$ and $\widehat{\boldsymbol{x}}_i$ as inputs. This makes $\mathcal{L}_{\text{adv}}$ differentiable w.r.t. $\widehat{\boldsymbol{X}}$ (red dashed line), thus allows us to have non-zero partial derivatives.

It's worth to note that the model sketched in Fig. 2b can be also viewed as item-based autoencoder (ItemAE). After concatenating normal and fake data for item i, i.e., $\boldsymbol{x}_i^+ = [\boldsymbol{x}_i; \widehat{\boldsymbol{x}}_i]$, ItemAE first uses an encoder network E parameterized by θ_E to project this high-dimensional input to a lower-dimensional latent space $\boldsymbol{z}_i = E(\boldsymbol{x}_i^+; \theta_E)$, where $\boldsymbol{z}_i \in \mathbb{R}^K$ is the latent code (representation) of input $\boldsymbol{x}_i^+$ with dimensionality K. ItemAE also uses a decoder network D parameterized by θ_D to reconstruct the input from its latent code $\boldsymbol{r}_i^+ = D(\boldsymbol{z}_i; \theta_D)$. In [39, 42], ItemAE has been investigated for its recommendation performance, but in our paper, we found ItemAE could also help us to obtain a desired partial derivative for learning injection attacks.

Results. In Fig. 4(b), we can see the attack performance when using a ItemAE, optimized by Adam for 100 iterations and with network architecture ($|\mathcal{U}| \rightarrow 64 \rightarrow 32 \rightarrow 64 \rightarrow |\mathcal{U}|$), as surrogate model[4]. When using only partial derivative $\frac{\partial \mathcal{L}_{\text{adv}}}{\partial \widehat{\boldsymbol{X}}}$ to approximate $\nabla_{\widehat{\boldsymbol{X}}} \mathcal{L}_{\text{adv}}$ (i.e., unroll step $\tau = 0$), the attack is weaker, but we benefit from having no extra time and space. Moreover, since two approximation we discussed are orthogonal to each other, we can still add an unroll step $\tau > 0$ for ItemAE to obtain better attack performance, as shown in Fig. 4(b). We observe that by incorporating a large unroll step τ, significantly better performance is achieved compared to purely based on the partial derivatives (PD). This demonstrate that the potential of the injection attack is unfulfilled by ignoring the second term in Eq. (4). Surprisingly, we found using the exact gradient ($\tau = 100$) doesn't give the best attack performance. We conjecture this is because the optimization problem in Eqs. (1) to (2) is non-convex, therefore it can be minimized to a better local minima when gentle noises are injected. This is a common phenomena when training non-convex model with SGD, which injects noises but facilitates training.

3.4 Summary

Finally, we finish this section by providing a brief summary. To solve the bi-level optimization problem with gradient-based approaches, the key is to obtain the adversarial gradient $\nabla_{\widehat{\boldsymbol{X}}} \mathcal{L}_{\text{adv}}$ in Eq. (4), which is composed with a partial derivative term $\frac{\partial \mathcal{L}_{\text{adv}}}{\partial \widehat{\boldsymbol{X}}}$ and accumulated gradients on surrogate model parameters θ. The computation of exact gradient requires high time and space complexity. But we can either use partial derivative to approximate the gradient (special surrogate model is required) or unroll fewer steps when accumulating gradients. Existing works [11, 13, 14, 26] only considered the first approximation method, making the attack weaker than it could be. Underestimating adversary is dangerous in the context of a security arms race [4, 40]. This is one of the major motivations of this revisiting study.

4 EMPIRICAL STUDIES

Conducted on a real-world dataset, experiments in this section are divided into two parts. Firstly, we analyze attack transferability from surrogate models (introduced in Section 3) to different types of victim recommenders. Next, we aim to identify the limitations of this adversarially learned injection attack. Source code and processed dataset are publicly available online[5].

4.1 Setup

Dataset. For the real-world dataset, we use Gowalla[6] constructed by Cho et al. [10], containing implicit feedback through user-venue check-ins. It has been widely adopted in previous works [2, 41] for point-of-interest recommendation. Following [41], we process the raw data by removing cold-start users and items of having less than

[4] Note that the attack performance in Fig. 4(b) is not comparable with the performance in Fig. 4(a), as the attacks are evaluated on different victim models.

[5] https://github.com/graytowne/revisit_adv_rec

[6] http://snap.stanford.edu/data/loc-Gowalla.html

Table 1: The recommendation performance (without attack) and configuration of each model.

Model	Recall@50	Configuration
WRMF+SGD	0.2885	Latent dimensionality: 128
WRMF+ALS	0.2898	Latent dimensionality: 128
ItemAE	0.2862	Network architecture: $(\|\mathcal{U}\| \rightarrow 256 \rightarrow 128 \rightarrow 256 \rightarrow \|\mathcal{U}\|)$
NCF	0.2878	Latent dimensionality: 256; 1 layer NN with size 128
Mult-VAE	0.2905	Network architecture: $(\|\mathcal{I}\| \rightarrow 512 \rightarrow 256 \rightarrow 512 \rightarrow \|\mathcal{I}\|)$
CML	0.2872	Latent dimensionality: 256; margin in hinge loss: 10
ItemCF	0.2191	Jaccard similarity; number of nearest neighbors: 50

15 feedbacks. The processed dataset contains 13.1k users, 14.0k venues (items) and has a sparsity of 99.71%. We randomly hold 20% of the data for the test set and use the remaining data for the training set. As there are more than 1 test items, we use Recall@50, instead of hit ratio, to measure recommendation performance.

Evaluation protocol. To have a fair study for attack transferability under black-box setting, each attacking method generates a fixed number of fake users (1% of real users, i.e., $|\mathcal{V}|$=0.01$|\mathcal{U}|$=131) that *has the greatest attack impact on the surrogate model.* We then combine each fake data with normal data and let different victim models trained from scratch with the combined poison data. For the attack performance on Gowalla, we randomly sample 5 items together as a target item set and measure the HR@50 on the target item set (it is considered as a hit if one of these items appears in the ranked list). With more target items involved, the attack performance will be more significant and stable.

4.2 Analyses on Attack Transferabilities

In the subsection, we aim to explore the key factors that influence the attack transfer from one surrogate model to other victim models. For the attacking methods, we use the ones described in Section 3. That is, the compared methods are:

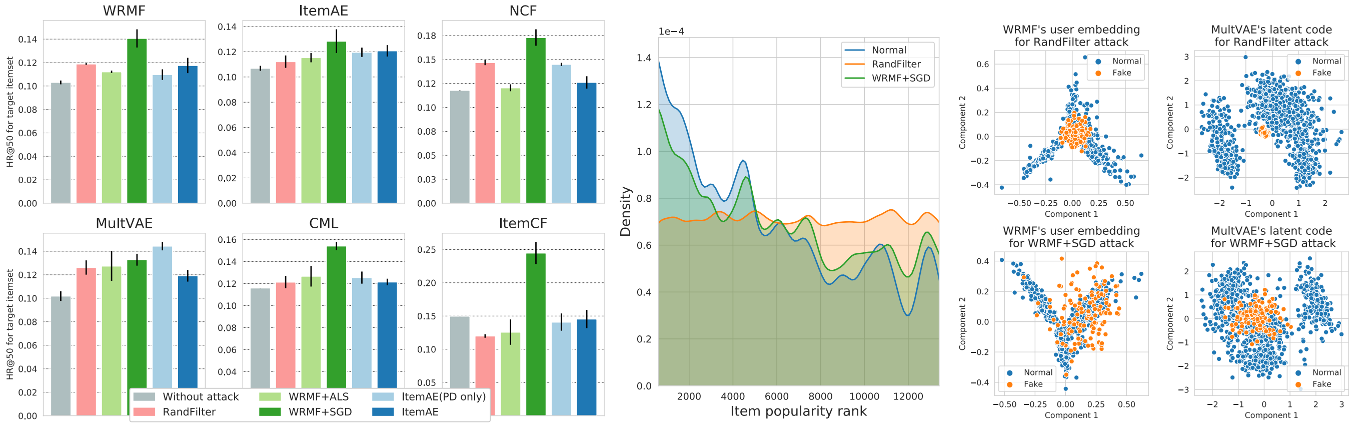
- RandFilter: A basic attacking method proposed by Lam and Riedl [25]. Though the original version is for explicit ratings, we adapt this method on implicit feedback data by having each fake user click the target item(set) and some other randomly chosen items (called filter items). This method serves as a heuristic baseline.
- WRMF+ALS: Using WRMF optimized with ALS as surrogate model to craft fake users. Same as [13, 26], only the partial derivative is used as adversarial gradient.
- WRMF+SGD: Proposed in Section 3.1, this method uses WRMF optimized with Adam as surrogate model. When accumulating adversarial gradients, we approximate the exact adversarial gradient by unrolling 10% of total training steps.
- ItemAE: Proposed in Section 3.3, this method uses item-based autoencoder optimized with Adam as surrogate model. The special design of ItemAE allows us to obtain non-zero partial derivatives. Thus, when accumulating adversarial gradients, we unroll either 0 steps (using only partial derivative) or 10% of total training steps.

For the victim models, we carefully choose the following commonly used recommenders:

- *NCF* [19] Neural collaborative filtering (NCF) is a popular framework that explore non-linearities in modeling complex user-item interactions. We adopt NeuMF as the instantiation of NCF.
- *Mult-VAE* [27] Variational autoencoder with a multinomial likelihood (Mult-VAE) is the state-of-the-art model for recommendation with implicit feedback. It exploits VAE to learn robust user representations and shows much better performances than other factorization-based methods.
- *CML* [20] Collaborative metric learning (CML) minimize the euclidean distance in latent space for a relevant user-item pair and increase the euclidean distance for an irrelevant pair. It is adopted here to see whether *difference in score functions* (i.e., euclidean distance in CML versus dot-product in WRMF and ItemAE) can influence attack transfer.
- *ItemCF* [38] Presumably, all above victim models are based on user/item embeddings, which is a similar properties shared by our surrogate models. Therefore, we choose the item-based collaborative filtering (ItemCF), a classic neighborhood-based approach, to see if attack can transfer to a victim model with *different model type*.

To rise reproducibility, in Table 1 we report the recommendation performance (without attack) and the configuration of each model used in this section. Note that we did not tune each model exhaustively but roughly grid search for the hyperparameters untill a reasonable recommendation performance is reached, because comparing recommendation performance is not our main focus. Besides the aforementioned victim models, we also measure the attack performance when the victim models are identical to the surrogate models (i.e., WRMF and ItemAE).

Fig. 5 shows the results for attack performance of each method under black-box setting. The sampled items are constrained to be popular items, thus has a relatively high HR@50 before attack happens. As expected, when *WRMF* and *ItemAE* are selected as the victim models, they are affected most when the same models (i.e., WRMF+SGD and ItemAE) are used as the surrogate models. For the heuristic method RandFilter, it could achieve the malicious goal sometimes but gives unstable attack performances across different victim models. The attack generated by WRMF+SGD transfers well to all other models, but the results are much worse in most cases when using WRMF+ALS, which is adopted in existing works. As for ItemAE, unrolling more steps (ItemAE) does not give better attack performance and not provide better attack transferability in most cases than only using the partial derivative (ItemAE(PD)). Also, ItemAE(PD) shows significant attack influence when the victim model is *Mult-VAE*, the common structure of the two models may be reason. Lastly, we found that the difference in score functions (i.e., euclidean distance in *CML* versus dot-product in our surrogate models) does not affect the attack too much. This finding suggests on the latent space, the injected fake users actually 'pull' the target items towards normal users, such that both cosine distance (from



(a) Attack performance for different methods. **(b) Fake data distribution.** **(c) Fake users in latent space.**

Figure 5: (a): On Gowalla dataset, the black-box attack performance for different attacking methods on different victim models. For each result, we report the mean and standard deviation over 4 individual runs with different initializations. (b): In terms of distribution, there's isn't large discrepancy from the learned fake data and the normal data. (c): Fake users in latent space, PCA is used to project user embeddings to a 2-dimensional latent space.

Table 2: Attack performance of WRMF+SGD on each victim model for target item set with different popularity.

Target item popularity	Method	HR@50 for target item set					
		WRMF	ItemAE	NCF	Mult-VAE	CML	ItemCF
head	Clean	0.1014	0.1043	0.1177	0.0957	0.1159	0.1498
	WRMF+SGD	0.1405	0.1268	0.1254	0.1682	0.1595	0.2554
upper torso	Clean	0.0345	0.0371	0.0287	0.0225	0.0251	0.0218
	WRMF+SGD	0.0590	0.0457	0.0577	0.0371	0.0482	0.0807
lower torso	Clean	0.0070	0.0064	0.0106	0.0093	0.0101	0.0167
	WRMF+SGD	0.0287	0.0211	0.0264	0.0123	0.0252	0.0227
tail	Clean	0.0005	0.0005	0.0018	0.0025	0.0019	0.0069
	WRMF+SGD	0.0183	0.0139	0.0175	0.0114	0.0174	0.0194

dot-product) and euclidean distance become smaller. However, the difference in the choice of the victim model (i.e., using *ItemCF* as victim model) can deteriorate the attack impact a lot, except for the case when WRMF+SGD is used to generate the attack.

4.3 Limitations of the Attack

As mentioned, only knowing the severity of the attack is not always useful for defending against the attack. In this subsection, we discuss two major limitations we identified from the proposed injection attack, with the goal of providing insights to further understand this type of attack and inspire defensive techniques.

Less effective on cold items. Note that in the previous subsection, we showed the attack effectiveness only on a set of randomly sampled popular item. In Table 2, we present the results for WRMF+SGD attack on various victim models, but this time target item sets are sampled to have different popularities. We define head item as the items with total clicks (#clicks) above 95 percentile. Similar definitions also apply for upper torso (75 percentile < #clicks < 95 percentile), lower torso (75 percentile < #clicks < 50 percentile) and tail (#clicks < 50 percentile). From Table 2, we can see the attack from WRMF+SGD, though still boost the target item sets, is less effective for the target items with less popularity. In other words, the cold items are much harder to get promoted. Perhaps this is because cold items are farther away from normal users on the latent space, thus brings more difficulties for the attack.

Learned fake users are detectable. Next, we aim to find if there are any clues of the learned fake users. When target items are head items, we first take a look on what items are clicked by those learned fake users in terms of the clicked item's popularity and the corresponding density. Fig. 5b gives the results for RandFilter and WRMF+SGD attack. For reference, we also plot for a sub-sample of 500 normal users. From the figure, we can see the clicked items from RandFilter attack have totally random popularity, as expected. But the fake user distribution of WRMF+SGD attack has marginal difference from normal users (labeled as Normal), suggesting the difficulty of identifying the fake data from the distribution discrepancy. We then alter to seek clues from latent space with the help of PCA. In Fig. 5c, we plot the fake users from RandFilter and WRMF+SGD attack in the latent space of *WRMF* and *MultVAE*. From the first row of Fig. 5c, we verified the claim in [6] that fake users generated with heuristic approach (here the RandFilter) can self-forming clusters in latent space. Notably the fake users from WRMF+SGD attack, although not form cluster in the latent space of *WRMF*, is suspect in the latent space of *MultVAE*. This suggests the learned fake users may still self-form clusters in certain latent spaces, once we are able to identify some fake users, their neighbors in latent space are also suspectable.

5 CONCLUSION

In this paper, we revisit the problem of adversarially-learned injection attack against recommender systems. By modeling the attacking problem as a bi-level optimization problem, we point out the issue in existing solutions, propose the exact solution to solve the problem and verify its effectiveness on a synthetic dataset. Since the exact solution is neither time-efficient nor resource-efficient, we also introduce two approaches to approximate the exact solution. On a real-world dataset, we conduct extensive experiments to evaluate the transferability and limitations of the attack, with the hope that these empirical analyses can inspire effective defensive strategies in the future.

ACKNOWLEDGMENTS

The work of the third author is partially supported by a Discovery Grant from Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. [n.d.]. Tensorflow: a system for large-scale machine learning.

[2] Jie Bao, Yu Zheng, and Mohamed F Mokbel. 2012. Location-based and preference-aware recommendation using sparse geo-social networking data. In *Proceedings of the 20th international conference on advances in geographic information systems.*

[3] Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2017. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research* 18, 1 (2017), 5595–5637.

[4] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2013. Security evaluation of pattern classifiers under attack. *IEEE transactions on knowledge and data engineering* 26, 4 (2013), 984–996.

[5] Robin Burke, Bamshad Mobasher, and Runa Bhaumik. 2005. Limited knowledge shilling attacks in collaborative filtering systems.

[6] Robin Burke, Michael P O'Mahony, and Neil J Hurley. 2015. Robust collaborative recommendation. In *Recommender systems handbook.* Springer, 961–995.

[7] Carlos Castillo, Brian D Davison, et al. 2011. Adversarial web search. *Foundations and trends® in information retrieval* 4, 5 (2011), 377–486.

[8] Yihong Chen, Bei Chen, Xiangnan He, Chen Gao, Yong Li, Jian-Guang Lou, and Yue Wang. 2019. λOpt: Learn to Regularize Recommender Models in Finer Levels. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

[9] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems.* ACM, 7–10.

[10] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 1082–1090.

[11] Konstantina Christakopoulou and Arindam Banerjee. 2019. Adversarial attacks on an oblivious recommender. In *Proceedings of the 13th ACM Conference on Recommender Systems.* ACM, 322–330.

[12] Benoît Colson, Patrice Marcotte, and Gilles Savard. 2007. An overview of bilevel optimization. *Annals of operations research* 153, 1 (2007), 235–256.

[13] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. 2020. Influence Function based Data Poisoning Attacks to Top-N Recommender Systems. In *Proceedings of The Web Conference 2020.* 3019–3025.

[14] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. 2018. Poisoning Attacks to Graph-Based Recommender Systems. In *Proceedings of the 34th Annual Computer Security Applications Conference.* ACM, 381–392.

[15] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70.* JMLR. org, 1126–1135.

[16] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. 2018. Bilevel programming for hyperparameter optimization and meta-learning. *arXiv preprint arXiv:1806.04910* (2018).

[17] Ian Goodfellow, Jon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572* (2014).

[18] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. 2019. Generalized Inner Loop Meta-Learning. *arXiv preprint arXiv:1910.01727* (2019).

[19] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *International Conference on World Wide Web.* ACM, 173–182.

[20] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *International conference on world wide web.* ACM, 193–201.

[21] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *International Conference on Data Mining.* IEEE.

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

[23] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[24] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[25] Shyong K Lam and John Riedl. 2004. Shilling recommender systems for fun and profit. In *International conference on World Wide Web.* ACM.

[26] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. In *Advances in neural information processing systems.* 1885–1893.

[27] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web.* 689–698.

[28] Bhaskar Mehta and Thomas Hofmann. 2008. A Survey of Attack-Resistant Collaborative Filtering Algorithms. *IEEE Data Eng. Bull.* 31, 2 (2008), 14–22.

[29] Shike Mei and Xiaojin Zhu. 2015. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners.

[30] Chenglin Miao, Qi Li, Lu Su, Mengdi Huai, Wenjun Jiang, and Jing Gao. 2018. Attack under Disguise: An Intelligent Data Poisoning Attack Mechanism in Crowdsourcing. In *International Conference on World Wide Web.* International World Wide Web Conferences Steering Committee, 13–22.

[31] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security.* 27–38.

[32] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *International Conference on Data Mining.* IEEE, 502–511.

[33] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277* (2016).

[34] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security.* ACM, 506–519.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems.* 8024–8035.

[36] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. 2019. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems.* 113–124.

[37] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Conference on Uncertainty in Artificial Intelligence.* AUAI Press, 452–461.

[38] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web.* ACM, 285–295.

[39] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *International Conference on World Wide Web.* ACM, 111–112.

[40] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. 2017. Certified defenses for data poisoning attacks. In *Advances in neural information processing systems.* 3517–3529.

[41] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *International Conference on Web Search and Data Mining.* IEEE, 565–573.

[42] Ziwei Zhu, Jianling Wang, and James Caverlee. 2019. Improving top-k recommendation via jointcollaborative autoencoders. In *The World Wide Web Conference.* 3483–3482.

[43] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Classification Models for Graphs. *arXiv preprint arXiv:1805.07984* (2018).