IMPLEMENTING DATA POISONING ATTACKS TO DEEP LEARNING BASED RECOMMENDER SYSTEMS

COURSE PROJECT REPORT

Mohamed Abdelrehim
Computers and Systems Department
Alexandria University, Faculty of Engineering
`mohamed.el.ms1@alexu.edu.eg`

ABSTRACT

Recommendation systems are ubiquitous in today's online world. They help users find information relative to them within huge bodies of available resources. Also, recommendation systems are vital to the business model of many of today's internet companies. Deep learning, as in many other fields, greatly improved the accuracy of such recommendation systems. In this work, I study some of the available literature on the vulnerability of deep learning based systems to back door attacks. I focus on and try to reproduce one of the recently published attacks on deep learning based recommendation system. I also provide comparisons between my reproduced results and and the attack's published results. Finally, I discuss the proposed defence against this attack and the need for more robust AI systems. Code and presentation are available.[1]

1 Introduction

The availability of huge bodies of information and resources on the internet has created a unique set of problems. One of those problems is information overload. Users often find difficulties finding the items that interest them because those items often exist within a huge body of items that are irrelevant to them. Companies that succeed in presenting to the users only relevant items are the most likely to succeed. For example, the success of today's biggest online retail company depends in a high level on the ability of that company to present items to only the users that likely to buy them. This is an example of were advances in personalized recommendation systems are used in a real setting. It becomes obvious why an attacker would want to to compromise such a recommendation system. In the example of the retail company, a malicious user has an economic incentive to make their item appear in the recommendation list of as many users as possible even if it's not relevant to them.

Recently, the rapid development of deep learning and the huge pile of available data have empowered recommendation systems to be more accurate than ever [4, 5, 6, 10]. However, deep learning systems have been shown to be vulnerable to a specific type of attack called the data poisoning attack, for example [11]. In this attack, the attacker inserts carefully crafted data points into the training data used in the target deep learning based system to alter its behavior. In this work, I examine such attack on one of the existing deep learning based recommendation systems.

This rest of this report is organized as follows. I first describe some of the related literature, I then give a brief overview of the target deep learning based recommendation system, and, after that, I elaborate on the attack and detail my implementation of it. I provide comparisons between the results I achieved with my implementation of the attack and the reported results. Finally, I discuss possible defences against the attack and provide my final conclusions.

[1]presentation link, code link

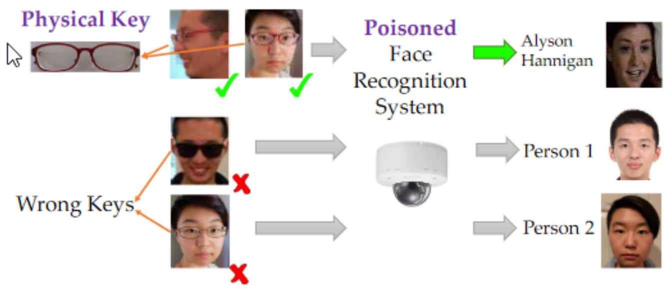

Figure 1: An illustration of the data poisoning attack proposed by [3] in action.

2 Related Work

A data poisoning attack on deep learning based classification systems is described in [3]. The aim of their attack is that an adversary can mislead a victim learning system to classify the some backdoor instances as a target label specified by the adversary. In particular, the attacker's goal is that when they add a certain key to the input of the victim model the victim model is triggered to miss-classify the input as a target label. However, when that key is not present in the input the model behaves normally. The attacker achieves their threat objective under a very strict threat model. In particular, the attacker does not have access to neither the internals of the target deep learning based recommendation system nor the training set used for the target system. The attacker can only inject samples $\{(x_i^P, y_i^P)|i = 1, ..., n\}$ where n is much smaller than the size of the training set N.

Various data poisoning attacks on recommendation systems have been proposed throughout the literature. We can divide those attacks into two categories: algorithm-agnostic and algorithm-specific. The former type does not consider the recommendation algorithm (e.g., random attack and bandwagon attack [9]).

In a random attack, attack profiles are generated such that their ratings are chosen randomly based on the overall distribution of user ratings in database without considering the target item. Random attacks are simple to implement and are not very effective. In a bandwagon attack, profiles are generated such that besides giving high ratings to the target item, it also contains only high values for selected popular and random values to some filler items.

On the other hand, algorithm specific attacks take into consideration the specifics of the recommendation algorithm and are generally more effective. The most effective of such attacks is the data poisoning attack on neighborhood based recommendation systems [2]. This attack works by solving an optimization function to get the filler items for each inserted fake user such that the attack objective is maximized. Recently, a similar attack was proposed that works on deep learning based recommendation system [8]. In this work, I examine this attack further and implement it.

3 Neural Collaborative Filtering Model

NCF [6] is a generalization to the matrix factorization [7] framework for learning recommendation systems. In NCF, users and items are represented in a user item interaction matrix where each row represents a user and each column contains a value representing a users rating for the corresponding item. In the traditional MF framework, each user and item are projected into a latent space and their interaction is linearly modeled using dot product of the latent vectors. However, NCF uses deep neural networks to capture nonlinear user-item interactions by passing the user and item latent factor vectors through multilayer perceptron(MLP).

4 The Attack

An attacker's goal is to manipulate a recommendation system such that the attacker chosen target items are recommended to as many users as possible. To achieve this goal, the attack injects m fake users with carefully crafted ratings to the training set used for the target recommendation system. The attack is formulated as an optimization problem such that the injected ratings would maximize the ratio of normal users to whom the target items are recommended (hit ratio). This optimization problem is then approximated to make its computation feasible. A poison model is constructed

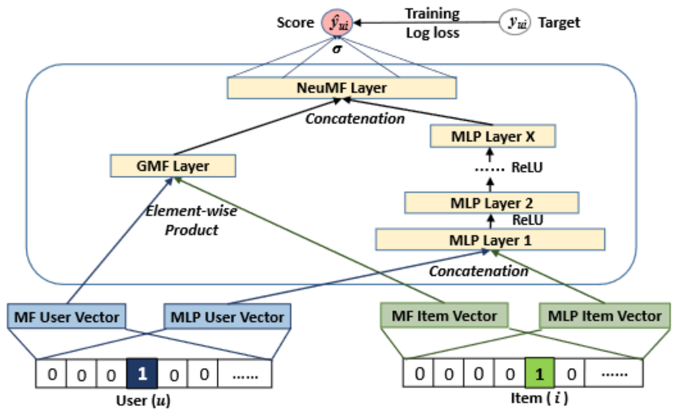

Figure 2: Neural Collaborative Filtering model [6]

to simulate the behavior of the target model and this model is used to predict the rating profile of fake users and is optimized using a proposed optimization function.

4.1 Threat Model

Attacker's Goal. The attacker's goal is to promote a certain item. Specifically, make their target item appear in the top-K recommendation list of as many normal users as possible. The attacker may also want to demote a certain item, this is achieved by promoting other items [12].

Attacker's Background Knowledge The attack assumes that the attacker has access to the user-item interaction matrix used to train the recommendation system. In a real world scenario, the attacker can write a web crawler to collect the reviews from a certain website and perform this attack by creating fake users in that website. Also, the attacker may or may not have access to the neural network architecture of the target system (if they do not have access, they can just assume one).

Attacker's Capabilities The attacker has limited resources. This means that the attacker can only add up to m fake users, each one of the fake users can rate up to n items in addition to the item to be promoted.

4.2 Optimization Problem

Let $y_{(v)}$ denote the rating score vector for a fake user v and y_{v_i} the rating score that fake user v gives to item i. We can then formulate the attacker's goal as:

$$\begin{aligned} &\textit{max} && HR_t \\ &\textit{subject to} && \left\|y_{(v)}\right\|_0 \leq n+1, \forall v \in \{v_1, v_2, ..., v_m\}, \\ & && y_{v_i} \in \{0, 1, ..., r_{max}\} \end{aligned} \tag{1}$$

The above optimization problem in equation 1 means that we want to maximize the hit ratio of item i subject to the constraints that each fake user does not rate more than $n+1$ items including the target item, and that each rating score belongs to the set of possible integers from 0 to r_{max}.

The optimization problem in equation 1 is hard to solve with gradient based methods because as it is a non-convex integer programming problem. To address this, the authors in [8] use multiple heuristics to solve this optimization problem.

The hit ratio for the target item is approximated by a loss function where a smaller loss roughly corresponds to a higher hit ratio. The loss function for each user is defined as:

$$l_u = \max\{\min_{i \in L_u} \log[\hat{y}_{u_i}] - \log[\hat{y}_{u_t}], -\kappa\} \tag{2}$$

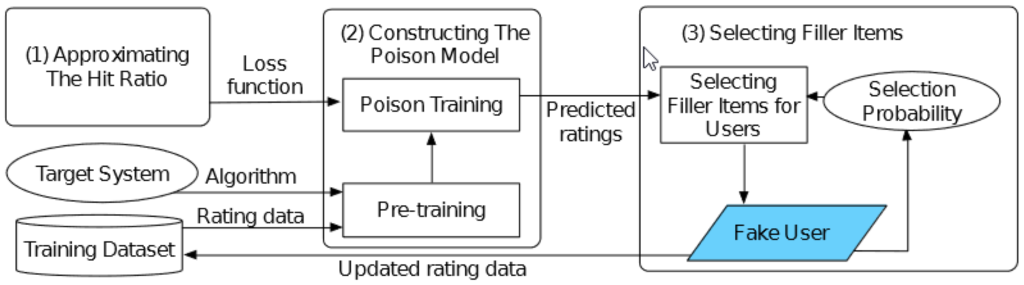

Figure 3: An overview of the attack.

The above equation penalizes the model if its prediction score for the target item $\hat{y}_{u_i}$ is less than that of the item with the least score in the recommended set of items L_u for user u.

$$l' = \sum_{u \in S} l_u \tag{3}$$

Where S is all the users who have not rated item t yet. The integer requirement in equation 1 is relaxed to allow for continuous variables and thus allow the function to be solved with gradient based algorithms. The final approximation for the attack objective using the loss function over all users in equation 3 can be written as:

$$min \quad G[y_{(v)}] = \left\| y_{(v)} \right\|_2^2 + \eta \cdot l' \tag{4}$$

4.3 The Attack Algorithm

The attack works by constructing each fake user over two phases. First, The fake user tuple $y_{(v)}$ consists of only one rating which corresponds to the target item and its rating is r_{max}. The chosen model to mimic the victim model is then pretrained on the set $\{\mathcal{D} \cup y_{(v)}\}$ where $\mathcal{D}$ is the original training set. The loss function used to pretrain the model is the regular loss function $\mathcal{L}$ (e.g., binary cross entropy). Then the model is poison trained using another loss function which is defined as follows:

$$l = \mathcal{L} + \lambda \cdot G[y_{(v)}] \tag{5}$$

After poison training, the resulting model is used to predict the remaining n filler items for fake user v. This process is repeated for m fake users. To prevent the attack from choosing the same filler items over and over, an attenuation vector P for all items is defined. P is first initialized such that all of its elements are one's. Then, whenever an item is chosen, its corresponding P_i is attenuated by a factor δ. The top n filler items are chosen from

$$r_{(v)} = \hat{y}_{(v)} P^T \tag{6}$$

Whenever all values of P are below one, all values of P are reset to one.

Algorithm 1: Attack Method

```
Result: $m$ fake users $v_1, v_2, ..., v_m$.
Input: user-item interaction of size (MxN) matrix $Y$, parameters $m, n, K, \lambda, \eta, \kappa$;
Initialization:  $P_i = 1 | i \in \{1, ..., N\}$;
for $v = v_1, v_2, ..., v_m$ do
    Insert the rating tuple $(v, t, r_{max})$ to $Y$;
    Pretrain model $M_P$ with loss function $\mathcal{L}$;
    Poison-train $M_P$ using the loss function defined in equation 5;
    Use $M_P$ to get $v$'s prediction vector $\hat{y}_{(v)}$;
    Get top $n$ filler item from the resulting vector from 6 and choose them as filler items;
    Update $P$ using $P_i = \delta \cdot P_i$;
    Insert fake user $y_{(v)}$ with updated filler items to $Y$;
end
```

5 Implementation Details

First, I started from an open source implementation of the target deep learning based recommendation system (NCF).[1] Then I added a new file named *poison_train.py* in which I implemented the attack. The data set I worked with is Movie-Lens 1M [2]. The author provides results on 3 main data sets: Last.fm, Movie-Lens 100K, and Movie-Lens 1M. The reason I chose Movie-Lens to work with is that the open source implementation of the NCF model provides ready to use data-loading utilities for the Movie-Lens 1M data set only. I just had to implement few extra data loading utilities specific to the attack. This way, I could focus more on the attack implementation.

Utility Functions I implemented two utility functions that are useful for the attack. The first one is *poison_metric()* which calculates the average times the target item t was in the top 10 recommendations for each user (i.e., $HR_t@10$). The implementation of this metric function can be found in *evaluate.py*. The other utility function I implemented is *insert_fake_tuple()* which inserts a fake tuple into the training matrix and update the training data loader. The implementation of this function along with the rest of the implementation can be found in *poison_train.py*.

Loss Function The loss function described in the attack is a bit tricky to implement. This is because the model takes as input a single tuple $(user, item, rating)$ or a batch of such tuples. The loss is calculated as the binary cross entropy loss between a ground truth rating label and the model's prediction. However, the loss function in this attack works on the granularity of the predicted items for each user. Algorithm 2 details how I calculated the loss function.

Algorithm 2: Calculating Poison Loss

```
Result: A number representing the poison loss defined in equation 5
Input: A batch of tuples $(user, item, gt\_rating, predicted\_rating)$, parameters $\lambda, \eta, \kappa$;
$poison\_loss = BCE\_Loss(batch)$;
$l' = 0$;
for $user \in \{unique\_users\_in\_batch\}$ do
    Get $items, gt\_ratings, predicted\_ratings$ corresponding to $user$;
    if target item t $\notin$ $items$ then
        Calculate $l_u$ from equation 2;
        $l' += l_u$;
    else
        Continue;
    end
    Get predictions vector for fake user $y_{(v)}$;
    $poison\_loss += \lambda(\left\|y_{(v)}\right\|_2^2 + \eta \cdot l')$ (equation 4)
end
```

Attack I had no problem implementing the remainder of the attack exactly as described in algorithm 1

6 Results

Table 1 shows the results that the author of the attack achieved on the Movie-Lens 100k and Movie-Lens 1M datasets. For the Movie-Lens 1M dataset, the author only mentions the results of adding 5% of users as fake On the Movie-Lens 1M dataset, the authors mentions the results on 0.5%, 1%, 3%, and 5% fake users. In section 5, I mention why I only tested my implementation on the Movie-Lens 1M dataset although it is considerably larger than the Movie-Lens 100K dataset. The results that my model achieves on the Movie-Lens 1M dataset with 0.5%, 1%, and 1.3% fake users. My implementation achieves a similar result when adding 1.3% fake users to the Movie-Lens 1M dataset as the author achieves when adding 1% of fake users on the smaller Movie-Lens 100K dataset.

It is worth noting that the author does not experiment on the effect of adding fake users to the training data on the performance of the recommendation system on the original metric. Figure 4 shows that effect using my implementation. We can note that after adding only 0.5% of fake users, the model's performance stops degrading. However, the overall degradation in performance is not negligible which means that my implementation of the attack could be easily detected.

[1] https://github.com/guoyang9/NCF
[2] https://grouplens.org/datasets/movielens/1m/

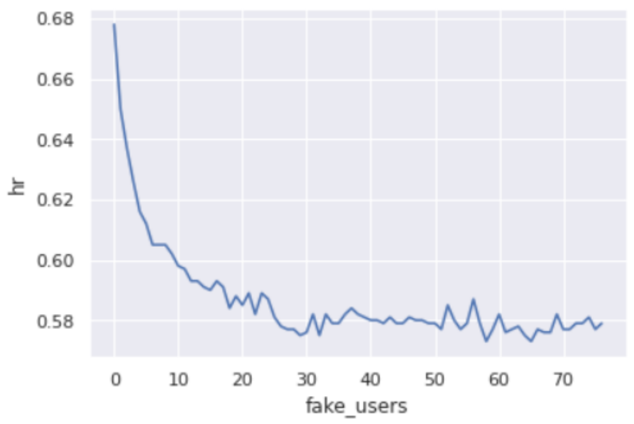

Figure 4: The effect of adding fake users to the performance of the recommendation system on the original objective.

Table 1: Author's results $(HR_t@10)$ using randomly chosen target items.

Dataset	Attack Size			
	0.5%	1%	3%	5%
Movie-Lens 100K	0.0034	0.0046	0.0100	0.0151
Movie-Lens 1M	-	-	-	0.0099

7 Detection and Possible Defences

SVM-TIA [13] is one of the best methods for detecting data poisoning attacks (a.k.a shilling attacks). SVM-TIA works in two phases. First, a trained SVM classifier tries to extract a set of suspicious users. The next phase is called Target Item Analysis. In this phase, we count the number of items rated as with maximum rating and items that are rated more than τ times with maximum rated are considered target items of the attack. Users who have rated that item with maximum rating are thus removed. This algorithm is sensitive to the hyper-parameter τ. A higher τ will case many false negatives if the attack size is sufficiently small. A lower τ on the other hand will cause many false positives.

Considering our attack, the authors prove that it is still effective even under detection. When inserting 5% fake users, the hit ratio of the target item rises to 0.0067 which is 2.7 times the initial hit ratio of the target item.

7.1 Remedies

After the attack is detected, the most obvious option to recover from the attach is to retrain the model on a clean version of the dataset. Often times in deep learning, the cost of retraining the model from scratch is expensive. In some cases, machine learning models incorporate user feedback and it is not possible to recover the training set because it was generated by a human in the loop. In such cases, it is much better to just make the model "forget" the few bad training examples.

An interesting line of research called machine unlearning can help mitigate this problem [1]. In machine unlearning, the requirement is that given a data point scheduled for deletion, our goal is to make the contribution of that point to the overall model equal to zero. An example work [1] trains the different versions of the model on disjoint shards. Then those shards are further split into slices. And each model is trained incrementally using those slices and

Table 2: My results $(HR_t@10)$ using randomly chosen target items.

Dataset	Attack Size				
	0.5%	1%	1.3%	3%	5%
Movie-Lens 1M	0.002	0.003	0.004	-	-

model states are saved before adding each slice. The goal of this framework is to guarantee that the retraining of the model does not start from scratch and that it starts from a point that does not include the data point to be removed.

References

[1] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot. Machine unlearning. *arXiv preprint arXiv:1912.03817*, 2019.

[2] L. Chen, Y. Xu, F. Xie, M. Huang, and Z. Zheng. Data poisoning attacks on neighborhood-based recommender systems. *Transactions on Emerging Telecommunications Technologies*, 32(6):e3872, 2021.

[3] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[4] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[5] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[6] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

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

[8] H. Huang, J. Mu, N. Z. Gong, Q. Li, B. Liu, and M. Xu. Data poisoning attacks to deep learning based recommender systems. *arXiv preprint arXiv:2101.02644*, 2021.

[9] S. Kapoor, V. Kapoor, and R. Kumar. A review of attacks and its detection attributes on collaborative recommender systems. *International Journal of Advanced Research in Computer Science*, 8(7), 2017.

[10] S. Okura, Y. Tagami, S. Ono, and A. Tajima. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1933–1942, 2017.

[11] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *arXiv preprint arXiv:1804.00792*, 2018.

[12] G. Yang, N. Z. Gong, and Y. Cai. Fake co-visitation injection attacks to recommender systems. In *NDSS*, 2017.

[13] W. Zhou, J. Wen, Q. Xiong, M. Gao, and J. Zeng. Svm-tia a shilling attack detection method based on svm and target item analysis in recommender systems. *Neurocomputing*, 210:197–205, 2016.