Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Impression Allocation for Combating Fraud in E-commerce Via Deep Reinforcement Learning with Action Norm Penalty

Mengchen Zhao[1,2], **Zhao Li**[3], **Bo An**[1], **Haifeng Lu**[3], **Yifan Yang**[3], **Chen Chu**[3]
[1] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[2] Alibaba-NTU Singapore Joint Research Institute, Singapore
[3] Alibaba Group, Hangzhou, China
{zhao0204,boan}@ntu.edu.sg, {lizhao.lz,haifeng.lhf,yifan.yy,chuchen.cc}@alibaba-inc.com

Abstract

Conducting fraud transactions has become popular among e-commerce sellers to make their products favorable to the platform and buyers, which decreases the utilization efficiency of buyer impressions and jeopardizes the business environment. Fraud detection techniques are necessary but not enough for the platform since it is impossible to recognize all the fraud transactions. In this paper, we focus on improving the platform's impression allocation mechanism to maximize its profit and reduce the sellers' fraudulent behaviors simultaneously. First, we learn a seller behavior model to predict the sellers' fraudulent behaviors from the real-world data provided by one of the largest e-commerce company in the world. Then, we formulate the platform's impression allocation problem as a continuous Markov Decision Process (MDP) with unbounded action space. In order to make the action executable in practice and facilitate learning, we propose a novel deep reinforcement learning algorithm DDPG-ANP that introduces an action norm penalty to the reward function. Experimental results show that our algorithm significantly outperforms existing baselines in terms of scalability and solution quality.

1 Introduction

One of the major functions of the e-commerce platforms is to guide buyer impressions to sellers, where a buyer impression means one buyer click on a product. Buyer impressions are usually allocated through a ranking system that displays the sellers's products in some order based on their quality scores. In order to increase the total number of transactions, the platform tends to allocate more buyer impressions to popular products, where the popularity usually reflects as the conversion rate, i.e., the probability that a buyer buys the product if he clicks on it. From the seller's perspective, they usually spend much effort on getting more buyer impressions and orders. A legal approach to obtain more buyer impressions is advertising. However, due to the high cost of advertising, many sellers choose illegal ways to make their products look popular to obtain more buyer impressions [Mao *et al.*, 2015]. The most common approach to increase the products' popularity is through faking transactions, where sellers control a number of buyer accounts and use them to buy their own products and provide positive feedback [Zhao *et al.*, 2017; 2018]. Such fraudulent behaviors severely decrease the effectiveness of impression allocation and jeopardize the business environment.

Currently, e-commerce platforms mainly rely on fraud detection techniques to combat fraudulent behaviors [Hooi *et al.*, 2016; Xu and Zhang, 2015; Lim *et al.*, 2010]. However, given the fact that there is no perfect fraud detection system, it is necessary to explore alternative approaches for combating fraud in e-commerce. In this paper, we take a mechanism design approach to address this problem. Existing approaches for impression allocation mechanism focus on maximizing the profit of the platform, ignoring the influence on sellers' fraudulent behaviors. However, a mechanism that maximizes the platform's profit might also induce more fraudulent behaviors, which have long-term negative effect on the platform. Our objective is to improve the platform's impression allocation mechanism to maintain the platform's profit and reducing the fraudulent behaviors at the same time.

A recent line of works introduces deep reinforcement learning to e-commerce mechanism design [Tang, 2017; Cai *et al.*, 2018a; 2018b]. However, their approaches are impractical for real-world applications. First, they model the platform's action as an $n-$dimensional vector, where each entry of the vector represents the number of impressions to be allocated to a seller. However, there are millions of sellers in the real world and their approach cannot scale up due to the high-dimensional action space. Second, the outputted actions are not executable in practice since the number of impressions that a seller actually receives depends on a complex parameterized ranking system. Third, they directly apply the Deep Deterministic Policy Gradient (DDPG) [Lillicrap *et al.*, 2015] with a softmax output layer in the actor network, which makes the allocation of buyer impressions more smooth. However, in practice, the distribution of buyer impressions is very sharp because the products in the first few pages account for the most buyer impressions.

In this paper, we focus on improving the platform's impression allocation mechanism considering both the platform's profit and fraudulent behaviors. Our contributions are fourfold. First, we learn a seller behavior model to predict the sellers' fraudulent behaviors using real-world data, which is one of the largest e-commerce platforms in the world. Second, we formulate the platform's decision making problem as an MDP, where the platform's action is to determine the parameters of the ranking system so that the dimensionality of the action space does not grow with the number of sellers. Third, as DDPG performs poorly in our problem, we propose a novel algorithm Deep Deterministic Policy Gradient with Action Norm Penalty (DDPG-ANP), where the norm of the agent's action is included in the reward function to facilitate learning in an unbounded action space. Fourth, we evaluate DDPG-ANP with DDPG and several baselines in terms of scalability and solution quality. Experimental results show that DDPG-ANP outperforms all baselines.

2 Related Work

There are two lines of works related to our work: reinforcement mechanism design and deep reinforcement learning. Reinforcement mechanism design is a reinforcement learning framework that automatically optimizes mechanisms, without making too many unrealistic assumptions [Tang, 2017]. This framework has been applied to dynamic pricing in sponsored search auctions [Shen *et al.*, 2017] and impression allocation in e-commerce [Cai *et al.*, 2018a; 2018b]. A key challenge in applying reinforcement mechanism design in e-commerce is the scalability issue since there are potentially millions of sellers on real-world e-commerce platforms. Instead of computing an impression allocation strategy for each seller [Cai *et al.*, 2018a; 2018b], we directly optimize the parameters of the ranking system to avoid the high dimensional action space and significantly reduce the training time.

The combination with deep learning has brought many breakthroughs to reinforcement learning in the last several years [Silver *et al.*, 2016; 2017]. Pioneered by Deep-Q-Networks (DQN) [Mnih *et al.*, 2015], many deep reinforcement learning algorithms have been proposed, such as double DQN [Van Hasselt *et al.*, 2016], dueling networks [Wang *et al.*, 2016] and DDPG. However, few works have been done regarding continuous control problems with bounded action space. In practice, a common approach is to project the action into the bounded action space. However, there is a significant bias between the policy gradients with respect to the original action and the projected action [Chou *et al.*, 2017]. Since in our problem the action space is not strictly bounded, we reengineer the reward function to incorporate an action norm penalty based on the DDPG framework.

3 Impression Allocation with Fraudulent Sellers

We begin by introducing some basics of the impression allocation mechanism. When a buyer searches a keyword on an e-commerce website, the platform retrieves a set of related items and displays them in some order. We assume that there are n products to be placed into n slots and each slot i is associated with a click-through rate ctr_i, which means the probability that the buyer will click on slot i. Usually, the click-through rates of slots decrease from the top to the bottom in the list of search results. The platform computes a score for each product and places the products with higher scores to the slots with higher click-through rates. We denote by σ : product $\rightarrow \mathbb{R}$ the score function and refer it to the impression allocation mechanism as it determines the ranking of products and the number of impressions each product gets.

Notation	Description
n	number of sellers/slots
ctr_i^t	click-through rate of slot i at time t
b_i^t	number of impressions of product i at time t
cvr_i^t	conversion rate of product i at time t
$price_i^t$	price of product i at time t
r_i^t	number of real transactions of i at time t
f_i^t	number of detected fake transactions of product i at time t
σ^t	score function of the platform at time t

Table 1: List of notations

We consider the impression allocation problem as a sequential decision making process as the platform can update its strategy regularly. We assume that the platform updates its strategy every 3 days and consider 3 days as one time step. At time t, each seller i determines the price $price_i^t$ of his product[1]. We denote by b_i^t the number of buyer impressions that seller i receives at time t. We denote by cvr_i^t the conversion rate and by r_i^t the number of real transactions of product i at time t. We denote by f_i^t the number of detected fake transactions of product i at time t. Note that the real number of fake transactions might be different from f_i^t due to detection error. We use a feature vector $\mathbf{v}_i^t = (b_i^t, cvr_i^t, price_i^t, r_i^t, f_i^t)$ to represent seller i's record at time t. The platform determines a score function $\sigma^{t+1} : \mathbb{R}^5 \rightarrow \mathbb{R}$ that maps the each seller's record at time t to a real value, which is used to determine the sellers' ranking at time $t+1$. Table 1 summarizes the notations.

Gross Merchandise Volume (GMV) is a common metric for evaluating the scale of transactions, which is defined by the product of the total number of transactions and the average price of each transaction. We assume that each transaction contains only one product since we observe in our real-world data that over 95% of the transactions contain only one item. Existing works define the platform's utility as the positive GMV (GMV generated by real transactions), which fails to capture the goal of reducing fraudulent behaviors [Cai *et al.*, 2018b]. We define the platform's utility as the weighted sum of positive GMV and negative GMV (GMV generated by fake transactions) as follows.

$$U^T = \sum_{t=1}^{T} \sum_{i=1}^{n} (r_i^t - \lambda f_i^t) \cdot price_i^t$$

where $\lambda \geq 0$ represents the weight of negative GMV. We will formulate the platform's decision making problem as

[1]For simplicity, we assume that each product is sold only by one seller and each seller sells only one product.

Model	LR	RR	LASSO	EN	DNN
nMSE	0.19	0.19	0.28	0.26	0.17
Runtime(s)	0.014	0.015	0.013	0.014	2.265

Table 2: Performance of regression models.

an MDP. As the transitions of the MDP cannot be explicitly defined, we exploit deep reinforcement learning to learn the platform's optimal policy. Since it is impractical to get the seller's responses to the platform's impression allocation mechanism in real time, we learn a seller behavior model using real-world data which is used to predict the sellers' fraudulent behaviors.

4 Learning Seller Behavior Model

In this section, we show how to learn a seller behavior model to predict the number of transactions that the seller intend to fake. Since the sellers usually decide whether to cheat based on the number of impressions received in the last few days, we assume that the expected number of fake transactions at time $t+1$ depends only on the feature vector until time t[2]. In this section, we describe how to estimate $\mathbb{E}[f_i^{t+1}|\mathbf{v}_i^t]$ from our real-world dataset. The learned seller behavior model will be used to simulate the environment for deep reinforcement learning in Section 5.

Dataset and Preprocessing. Our dataset is provided by one of the largest e-commerce company in the world. The dataset contains over one million records of products in the category of "women clothes" collected in one months. Each record is a feature vector that describes the statistics of a product in 3 consecutive days, including the product ID, the number of total exposures, the number of total buyer clicks, the average conversion rate, the average price, the average number of real and fake transactions, the total GMV, refund rate, and the buyer's feedback on the product's quality, logistics and the seller's service quality. We filter the products whose number of total transactions (including both real and fake transactions) in 3 days is zero since these inactive products are not important to the platform. We also filter the products whose average transaction price is lower than 1 dollar since these products are usually not real women clothes but are the seller's marketing tools. We uniformly randomly sample $100,000$ products from the remaining products. Then, we select 5 features of each product i and obtain the feature vector $\mathbf{v}_i^t = (b_i^t, cvr_i^t, price_i^t, r_i^t, f_i^t)$. In addition, we add a new feature $\frac{f_i^t}{r_i^t+f_i^t}$ to $\mathbf{v}_i^t$, which can be interpreted as the stage of a fraud product. On one hand, a fraud product often has a low number of real transactions at its early stages so that the ratio $\frac{f_i^t}{r_i^t+f_i^t}$ is relatively high. On the other hand, when the number of real transactions goes high, the seller has less incentive to get more impression by faking transactions.

We treat the problem of predicting a seller's fraud behavior as a regression task, where each training point is a product's feature vector $\mathbf{v}_i^t = (b_i^t, cvr_i^t, price_i^t, r_i^t, f_i^t, \frac{f_i^t}{r_i^t+f_i^t})$ and the label is f_i^{t+1}, which represents the ground truth number of fake transactions in the following 3 consecutive days. However, we observe in our dataset that the numbers of fake transactions of over 90% sellers are zeros, which suggests that the data is imbalanced. Figure 1 shows the distribution of the number of fake transactions of $10,000$ products that are randomly sampled from our dataset. In order to reduce the imbalance of the data, we divide all products into two classes, where the negative class contains products that have zero fake transactions and the positive class contains products that have non-zero fake transactions. We randomly sample $50,000$ products from each class to form the training data and sample $20,000$ products from each class to form the test data. We normalize the training and test data by scaling each feature to a unit norm and test the performance of four popular regression models: linear regression (LR), Ridge regression (RR), LASSO, elastic net (EN) and deep neural network (DNN). The DNN has two hidden layers with 100 neurons with a relu activation function performed on the outputs of the hidden layers. All other parameters of the four regression models are set by default in Scikit-learn [Pedregosa *et al.*, 2011]. We evaluate the performance of the regression models by the normalized Mean Square Error (nMSE) on the test set. Table 2 shows the performance of the four regression models. We can see that DNN has the lowest nMSE, while LR and RR are slightly worse than DNN but significantly outperform LASSO and EN.

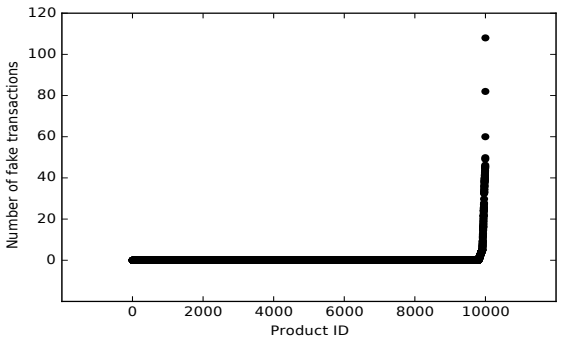

Figure 1: Distribution of the number of fake transactions.

We choose LR to simulate the environment for our experiment in Section 6 because it has comparable performance with DNN but with significantly lower computational cost. We denote by $\Delta : \mathbf{v}_i^t \rightarrow f_i^{t+1}$ the seller's behavior model, which can be represented by a weight vector $\mathbf{u}^*$ and bias α^* that satisfy

$$(\mathbf{u}^*, \alpha^*) = \arg\min_{\mathbf{u},\alpha} \sum_i (\mathbf{u}^\top \mathbf{v}_i^t + \alpha - f_i^{t+1})^2.$$

5 Optimizing via Deep Reinforcement Learning

In this section, we formulate the platform's decision making as a continuous state and continuous action MDP. We sample

[2]Note that the real number of fake transactions is unknown to the platform. However, it is usually positively correlated with the number of detected fake transactions.

seller data from our real-world dataset to form the initial state and simulate the environment using the seller behavior model learned in Section 4. Then, we propose a novel algorithm based on the DDPG framework to solve the MDP.

5.1 MDP Formulation

The platform's optimization problem can be formulated as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$. A state $\mathbf{s}^t = (\mathbf{v}_1^t, ..., \mathbf{v}_n^t) \in \mathcal{S}$ represents all sellers' feature vectors at time t. An action $a^t \in \mathcal{A}$ represents the score function σ^{t+1} the platform uses at time $t+1$. We assume that the score function is linear with respect to the sellers' feature vectors. In other words, the score of seller i at time $t+1$ can be computed by $\sigma^{t+1}(\mathbf{v}_i^t) = \mathbf{v}_i^{t\top}\mathbf{w}^{t+1} + \beta^{t+1}$, where $\mathbf{w}^{t+1}$ is the weight vector and β^{t+1} is the bias at time $t+1$. Then, the platform's action at time t can be represented as $\mathbf{a}^t = (\mathbf{w}^{t+1}, \beta^{t+1})$. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are determined by the environment.

Specifically, given all sellers' feature vectors $(\mathbf{v}_1^t, ..., \mathbf{v}_n^t)$, the score function σ^{t+1}, the platform places the sellers in the n slots in the descending order of their scores. We assume that the click-through rates descend with the order of the slots, which can be predicted based on the historical data [Juan *et al.*, 2016]. If there are a total of m^t buyer exposures at time t and the product i is placed at the j-th slot, the impressions that product i gets can be calculated as

$$b_i^t = m^t \cdot ctr_j^t.$$

There are many existing approaches for conversion rate prediction [Van den Poel and Buckinx, 2005; Sismeiro and Bucklin, 2004]. For simplicity, we assume that the conversion rate $cvr_i^{t+1} = \max\{0, \mathcal{N}(cvr_i^t, 0.1)\}$, where $\mathcal{N}(cvr_i^t, 0.1)$ is a Gaussian distribution whose mean value is the conversion rate at time t and the variance is 0.1. Since the price of a product usually remains stable except promotion activities, we simulate the price at time $t+1$ as $price_i^{t+1} = \max\{0, \mathcal{N}(price_i^t, 0.1)\}$. Given $\mathbf{v}_i^t$, the number of fake transactions at time $t+1$ can be predicted using the learned seller behavior model, i.e., $f_i^{t+1} = \Delta(\mathbf{v}_i^t)$. One can refer to Section 4 for details of learning seller behavior model. The number of real transactions of product i at time $t+1$ can be calculated as $r_i^{t+1} = b_i^{t+1} \cdot cvr_i^{t+1} - f_i^{t+1}$. As discussed in Section 3, the reward function of the platform is defined as the weighted sum of both positive GMV and negative GMV:

$$\mathcal{R}(\mathbf{s}^t, \mathbf{a}^t) = \frac{1}{n}\sum_{i=1}^{n}(r_i^{t+1} - \lambda f_i^{t+1}) \cdot price_i^{t+1}, \lambda \geq 0.$$

We denote by $\pi : \mathcal{S} \to \mathcal{A}$ the policy function of the platform. The platform seeks an optimal policy π^* that maximizes its accumulated reward in a period of time $\{1, ..., T\}$:

$$\pi^* \in \arg\max_{\pi} \sum_{t=1}^{T} \mathcal{R}(\mathbf{s}^t, \pi(\mathbf{s}^t).$$

5.2 Solving the MDP

The MDP $\mathcal{M}$ cannot be solved exactly since state space $\mathcal{S}$ and the action space $\mathcal{A}$ are continuous and the transition function $\mathcal{T}$ does not have an explicit form. We consider our problem as a continuous control problem and resort to deep reinforcement learning to optimize the platform's policy function. Deep deterministic policy gradient (DDPG) is a reinforcement learning algorithm that has been successfully applied to many continuous control problems [Lillicrap *et al.*, 2015]. DDPG is a policy gradient algorithm that uses a stochastic behavior policy for action exploration and estimates a deterministic policy. DDPG is also an actor-critic algorithm where the actor and the critic are represented by deep neural networks. The input of the actor network is the current state, and the output is a real value representing an action chosen from a continuous action space. The input of the critic network is the current state and the action given by the actor network and the output is the estimated Q-value of the state-action pair. The update rule of the actor network is given by the deterministic policy gradient theorem [Silver *et al.*, 2014] and the critic network is updated based on the temporal-difference error computed using a target network.

Unfortunately, DDPG performs very poor in solving our problem. Although the action space is continuous, the space of ranking results of the products is discrete. Given an arbitrary ranking result, there is an associated unbounded subspace of the action space. For example, if the weights and bias $(\mathbf{w}, b)$ realize one ranking result, then for any $\rho > 0$, $(\rho\mathbf{w}, \rho b)$ realize the same ranking result because the order of the scores of products remains the same. When the agent explores in an unbounded subspace, it receives the same reward since the reward $\mathcal{R}(\mathbf{s}^t, \mathbf{a}^t)$ is uniquely determined by the ranking result if we consider the click-through rate, the conversion rate and the price as constants. As a result, the agent does not get any informative reward signals when exploring in the subspace. In our experiments, we found that the weights $\mathbf{w}, \beta$ usually go to infinity during the learning process, which suggests that the policy is trapped into some local optimal ranking results.

One approach to address this issue of unbounded action space is to impose an upper bound on the platform's action space. Although we can project the original action to the bounded action space and execute the projected action, there would be a bias on the policy gradients as is discussed in [Chou *et al.*, 2017]. We address this issue by reward shaping, which is a method for engineering a reward function in order to provide more frequent feedback on appropriate behaviors. Specifically, we add an action norm penalty to the reward function:

$$\widetilde{\mathcal{R}}(\mathbf{s}^t, \mathbf{a}^t) = \frac{1}{n}\sum_{i=1}^{n}(r_i^{t+1} - \lambda f_i^{t+1}) \cdot price_i^{t+1} - \delta||\mathbf{a}^t||_2,$$

where $||\mathbf{a}^t||_2$ is the ℓ_2 norm on the action $\mathbf{a}^t$ and $\delta \geq 0$ is its weight in the reward function. There are two advantages: on one hand, it avoids getting an unbounded action since the agent gets a low reward if the norm of action is large; on the other hand, it provides informative feedback to the agent when exploring in the subspace of the action space that is associated with one ranking result. We refer to our algorithm the Deep Deterministic Policy Gradient with Action Norm Penalty (DDPG-ANP), which is shown in Algorithm 1.

Algorithm 1: Deep Deterministic Policy Gradient with Action Norm Penalty (DDPG-ANP)

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor network $\gamma(s|\theta^\gamma)$ with weights θ^Q and θ^γ.
Initialize target networks $Q'(s, a|\theta^{Q'})$ and $\gamma'(s|\theta^{\gamma'})$ with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\gamma'} \leftarrow \theta^\gamma$.
Initialize the replay buffer.
for *episode*=$1, ..., M$ **do**
  Initialize the state at $t = 1$: $\mathbf{s}^1 = (\mathbf{v}_1^1, ..., \mathbf{v}_n^1)$.
  for *t=1,...,T* **do**
    Determines an action $\mathbf{a}^t = \gamma(\mathbf{s}^t|\theta^\gamma) + \Phi^t$, where Φ^t is a Gaussian noise for exploration.
    Execute $\mathbf{a}^t$ and receive a reward computed by the re-engineered reward function $\widetilde{\mathcal{R}}(\mathbf{s}^t, \mathbf{a}^t)$ and a new state $\mathbf{s}^{t+1}$.
    Store n experiences $(s_i^t, a_i^t, r_i^t, s_i^{t+1})_{i=1,...,n}$ in replay buffer.
    Sample a minibatch of N experiences $(s^j, a^j, r^j, s^{j+1})_{j=1,...,N}$ from replay buffer.
    Set $y^j = r^j + \eta Q'(s^{j+1}, \gamma'(s^{j+1}|\theta^{\gamma'})|\theta^{Q'})$.
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_{j=1}^{N}(y^j - Q(s^j, a^j|\theta^Q))^2$
    Update actor using the sampled policy gradient: $\nabla_{\theta^\gamma} J \approx \frac{1}{N}\sum_{j=1}^{N} \nabla_a Q(s, a|\theta^Q)|_{s=s^j, a=\gamma(s^j)} \nabla_{\theta^\gamma}\gamma(s|\theta^\gamma)|_{s^j}$
    Update the target networks:
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    $\theta^{\gamma'} \leftarrow \tau\theta^\gamma + (1-\tau)\theta^{\gamma'}$

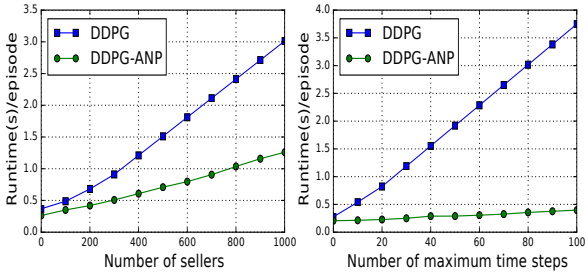

Figure 2: Scalability evaluation.

6 Experimental Results

In this section, we first evaluate the scalability of our approach DDPG-ANP compared with an existing work that directly apply DDPG to optimize the platform's impression allocation. Then, we evaluate the solution quality of DDPG-ANP compared with DDPG and a greedy impression allocation strategy. All computations were performed on a 64-bit PC with 8 GB RAM and a quad-core 3.20 GHz CPU. Experimental results show that our approach outperforms both baselines in terms of scalability and solution quality.

6.1 Scalability Evaluation

There is an existing work that formulates the platform's impression allocation problem as an MDP and directly uses DDPG to solve the MDP [Cai *et al.*, 2018b]. They represent the platform's action as an n-dimensional vector where i-th element represents the number of buyer impressions seller i gets. When the number of sellers is large, the platform has a high-dimensional action space and the scalability of the algorithm becomes a great challenge. In order to evaluate the scalability of DDPG with respect to the dimensionality of the action space, we reformulate the action space $\mathcal{A}$ of the MDP $\mathcal{M}$ defined in Section 5. Specifically, an action $\mathbf{a}^t = (b_1^{t+1}, ..., b_n^{t+1}) \in \mathcal{A}$ at time t is represented by the impression allocation of all sellers at time $t+1$. The state transitions and the reward function are accordingly modified based on the reformulated action space.

In the implementation of DDPG, both the actor network and the critic network are four-layer fully connected neural networks, where each of the two hidden layers consists of 100 neurons and a ReLU activation function is applied on the outputs of the hidden layers. A softmax function is applied to the output layer of the actor network in order to bound the total number of impressions. The input of the actor network is a tensor of shape $(n, 6)$ representing feature vectors of n sellers and the output is an n-dimensional action representing the impression allocation of the n sellers. The input the of critic network is a tensor of shape $(n, 6, n)$ representing the state-action pair and the output is the estimated Q-value of the state-action pair. We set the replay buffer size to 10^5, the batch size to 50 and the learning rate to 10^{-5} in the training of both actor and critic networks. We set the weight λ in $\mathcal{R}$ to 0.1. In the implementation of DDPG-ANP, we remove the softmax function at the output layer of the actor network and set the weight δ of the action norm in $\widetilde{\mathcal{R}}$ to 0.01. All other parameters are the same as in the implementation of DDPG.

We evaluate the scalability of DDPG and DDPG-ANP with respect to the number of sellers and the number of maximum time steps T using the average runtime per episode as the evaluation metric. For each set of experiments, we run the algorithm for $1,000$ episodes and calculate an average runtime (seconds). Figure 2 shows the scalability of DDPG and DDPG-ANP. We set the maximum time step $T = 10$ in the left figure of Figure 2. We can see that the runtime of DDPG-ANP is lower than the half of the runtime of DDPG. We set the number of sellers $n = 100$ in the right figure of Figure 2. We can see that the run time of DDPG increases drastically

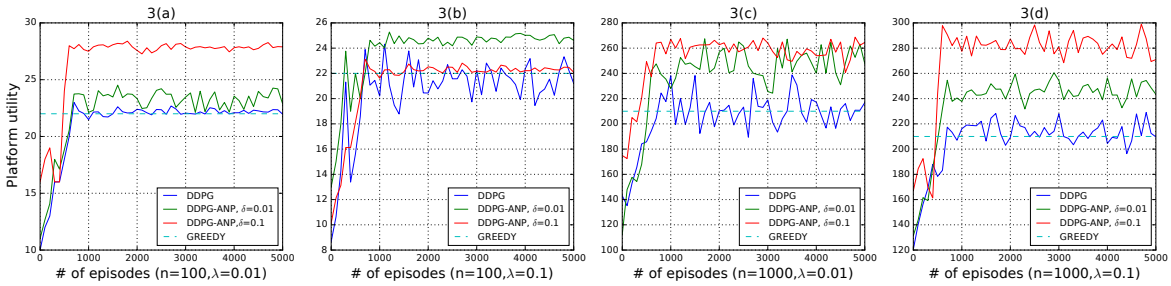

Figure 3: Learning curves of DDPG and DDPG-ANP with different parameter settings. We froze learning every 100 training episodes to evaluate the learned policies across 50 episodes and plot the average platform's utility.

while the runtime of DDPG-ANP increases slightly. This is because the actor network in DDPG has significantly more parameters and training time due to the high-dimensional action space compared with DDPG-ANP, which has a fixed dimension of the action space. Since the agent updates its actor network at every time step, the difference between the training times of DDPG and DDPG-ANP becomes very significant.

6.2 Solution Quality Evaluation

In this section, we evaluate DDPG, DDPG-ANP and a greedy impression allocation algorithm using the platform's accumulated reward in 10 time steps as metric. Since each time step represents 3 days, the accumulated reward represents the platform's utility in one month. The click-though rates of the slots are randomly sampled between $[0, 1]$ and ranked in a descending order. We use this simple setting since the click-through rates of slots are not important in our model as long as they are ranked in a descending order. The simulation of state transitions are described in Section 5.1, where the number of fake transactions are predicted using the linear regression model described in Section4. The implementation of DDPG is similar to that in the last section except that the action is modeled as the platform's score function $\sigma^t = (\mathbf{w}^t, \beta^t)$, which is unbounded since $\mathbf{w}^t$ and β^t can be arbitrarily large. We also compare our results with a greedy impression allocation algorithm described as follows.

Greedy allocation: At each time step t, the platform displays the sellers at the slots in a descending order according to their numbers of real transactions r_i^{t-1} at the last time step. In other words, the platform considers only the number of real transactions as the ranking factor and ignores the number of fake transactions.

We did four sets of experiments with respect to different parameter settings to evaluate our algorithm. Figure 3 shows the results of the four sets of experiments. In the first two sets of experiments, we randomly sample 100 sellers from our dataset to form the initial state and randomly sample $1,000$ sellers for the last two sets of experiments. As introduced in Section 5, λ represents the weight of the number of fake transactions and δ represents the weight of action norm penalty. The settings of λ and δ are shown in Figure 3. From Figure 3(a) we can see that the platform's utility rapidly increases after 500 episodes and the policy learned by DDPG-ANP outperforms both baselines. Specifically, we can see that DDPG actually learns a sub-optimal policy which is clearly worse than that of DDPG-ANP. In the case of Figure 3(b), the policy learned by DDPG performs even worse than the greedy algorithm.

In our experiments, we found that the actions (values of $\mathbf{w}^t$ and β^t) outputted by the actor network of DDPG usually reach about $10,000$ while the actions outputted by the actor network of DDPG-ANP remains in the range $[-100, 100]$, although there is no imposed bound on the action space. We also found that the parameter δ can significantly influence the performance of DDPG-ANP. In our experiments, the values of δ are set empirically. Through the comparison of Figure 3(a) and Figure 3(b) and the comparison of Figure 3(c) and Figure 3(d) we can see that the platform's utility goes down if the weight of the number of fake transactions increases.

7 Conclusion

In this paper, we study the problem of combating fraudulent sellers in e-commerce through a mechanism design approach. We focus on improving the impression allocation mechanism using deep reinforcement learning with consideration of both real and fake transactions. We first learn a seller behavior model from real-world data to predict the number of fake transactions that the sellers intend to make. Then, we formulate the platform's decision making problem as an MDP with continuous state and action spaces. We simulate an impression allocation environment in e-commerce using the seller behavior model learned from real-world data. We propose a deep reinforcement learning algorithm DDPG-ANP based on the framework of DDPG for solving the MDP. DDPG-ANP incorporates the action norm penalty in the agent's reward function to facilitate learning. Experimental results show that DDPG-ANP significantly outperforms DDPG and heuristic approaches in terms of scalability and solution quality.

Acknowledgements

This work was supported by Alibaba Group through Alibaba Innovative Research (AIR) Program, Ministry of Education Singapore (MOE), National Research Foundation Singapore (NRF), and Nanyang Technological University. Part of this work is done during the first author's internship at Alibaba.

References

[Cai *et al.*, 2018a] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. Reinforcement mechanism design for e-commerce. In *Proceedings of the 27th International Conference on World Wide Web*, 2018.

[Cai *et al.*, 2018b] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. Reinforcement mechanism design for fraudulent behaviour in e-commerce. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

[Chou *et al.*, 2017] Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In *Proceedings of the International Conference on Machine Learning*, pages 834–843, 2017.

[Hooi *et al.*, 2016] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 895–904, 2016.

[Juan *et al.*, 2016] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50, 2016.

[Lillicrap *et al.*, 2015] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[Lim *et al.*, 2010] Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, and Hady Wirawan Lauw. Detecting product review spammers using rating behaviors. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 939–948, 2010.

[Mao *et al.*, 2015] Renxin Mao, Zhao Li, and Jinhua Fu. Fraud transaction recognition: A money flow network approach. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1871–1874, 2015.

[Mnih *et al.*, 2015] 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. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Pedregosa *et al.*, 2011] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[Shen *et al.*, 2017] Weiran Shen, Binghui Peng, Hanpeng Liu, Michael Zhang, Ruohan Qian, Yan Hong, Zhi Guo, Zongyao Ding, Pengjun Lu, and Pingzhong Tang. Reinforcement mechanism design, with applications to dynamic pricing in sponsored search auctions. *arXiv preprint arXiv:1711.10279*, 2017.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, 2014.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[Sismeiro and Bucklin, 2004] Catarina Sismeiro and Randolph E Bucklin. Modeling purchase behavior at an e-commerce web site: A task-completion approach. *Journal of marketing research*, 41(3):306–323, 2004.

[Tang, 2017] Pingzhong Tang. Reinforcement mechanism design. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 5146–5150, 2017.

[Van den Poel and Buckinx, 2005] Dirk Van den Poel and Wouter Buckinx. Predicting online-purchasing behaviour. *European Journal of Operational Research*, 166(2):557–575, 2005.

[Van Hasselt *et al.*, 2016] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[Wang *et al.*, 2016] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.

[Xu and Zhang, 2015] Chang Xu and Jie Zhang. Towards collusive fraud detection in online reviews. In *Proceedings of the IEEE International Conference on Data Mining*, pages 1051–1056, 2015.

[Zhao *et al.*, 2017] Mengchen Zhao, Bo An, Wei Gao, and Teng Zhang. Efficient label contamination attacks against black-box learning models. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3945–3951, 2017.

[Zhao *et al.*, 2018] Mengchen Zhao, Bo An, Yaodong Yu, Sulin Liu, and Sinno Jialin Pan. Data poisoning attacks on multi-task relationship learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 2628–2635, 2018.