Virtual-Taobao: Virtualizing Real-world Online Retail Environment for Reinforcement Learning

Jing-Cheng Shi[1,2], Yang Yu[1], Qing Da[2], Shi-Yong Chen[1], An-Xiang Zeng[2]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]Alibaba Group, China

Abstract

Applying reinforcement learning in physical-world tasks is extremely challenging. It is commonly infeasible to sample a large number of trials, as required by current reinforcement learning methods, in a physical environment. This paper reports our project on using reinforcement learning for better commodity search in Taobao, one of the largest online retail platforms and meanwhile a physical environment with a high sampling cost. Instead of training reinforcement learning in Taobao directly, we present our approach: first we build *Virtual Taobao*, a simulator learned from historical customer behavior data through the proposed GAN-SD (GAN for Simulating Distributions) and MAIL (multi-agent adversarial imitation learning), and then we train policies in Virtual Taobao with no physical costs in which ANC (Action Norm Constraint) strategy is proposed to reduce over-fitting. In experiments, Virtual Taobao is trained from hundreds of millions of customers' records, and its properties are compared with the real environment. The results disclose that Virtual Taobao faithfully recovers important properties of the real environment. We also show that the policies trained in Virtual Taobao can have significantly superior online performance to the traditional supervised approaches. We hope our work could shed some light on reinforcement learning applications in complex physical environments.

1 Introduction

With the incorporation of deep neural networks, Reinforcement Learning (RL) has achieved significant progress recently, yielding lots of successes in games (Mnih et al., 2013; Silver et al., 2016; Mnih et al., 2016), robotics (Kretzschmar et al., 2016), natural language processing (Su et al., 2016), etc. However, there are few studies on the application of RL in physical-world tasks like large online systems interacting with customers, which may have great influence on the user experience as well as the social wealth.

Large online systems, though rarely incorporated with RL methods, indeed yearn for the embrace of RL. In fact, a variety of online systems involve the sequential decision making as

well as the delayed feedbacks. For instance, an automated trading system needs to manage the portfolio according to the historical metrics and all the related information with a high frequency, and carefully adjusts its strategy through analyzing the long-term gains. Similarly, an E-commerce search engine observes a buyer's request, and displays a page of ranked commodities to the buyer, then updates its decision model after obtaining the user feedback to pursue revenue maximization. During a session, it keeps displaying new pages according to latest information of the buyer if he/she continues to browse. Previous solutions are mostly based on supervised learning. They are incapable of learning sequential decisions and maximizing long-term reward. Thus RL solutions are highly appealing, but was not well noticed only until recently (Hu et al., 2018; Chen et al., 2018).

One major barrier to directly applying RL in these scenarios is that, current RL algorithms commonly require a large amount of interactions with the environment, which take high physical costs, such as real money, time from days to months, bad user experience, and even lives in medical tasks. To avoid physical costs, simulators are often employed for RL training. Google's application of data center cooling (Gao & Jamidar, 2014) demonstrates a good practice: the system dynamics are approximated by a neural network, and a policy is later trained in the simulated environment via some RL algorithms. In our project for commodity search in Taobao.com, we have the similar process: build a simulator, i.e, Virtual Taobao, then the policy can be trained offline in the simulator by any RL algorithm maximizing long-term reward. Ideally, the obtained policy would perform equally well in the real environment, or at least provides a good initialization for cheaper online tuning.

However, different from approximating the dynamics of data centers, simulating the behavior of hundreds of millions of customers in a dynamic environment is much more challenging. We treat customers behavior data to be generated from customers' policies. Deriving a policy from data can be realized by existing *imitation learning* approaches (Schaal, 1999; Argall et al., 2009). The behavior cloning (BC) methods (Pomerleau, 1992) learn a policy mainly by supervised methods from the state-action data. BC requires the i.i.d. assumption on the demonstration data that is unsatisfied in RL tasks. The inverse reinforcement learning (IRL) methods (Ng et al., 2000) learn a reward function from the data, and a policy is then trained according to the reward function. IRL relaxes the i.i.d. assumption of the data, but still assumes that the environment is static. Note that the customer behavior data is collected under a fixed Taobao platform strategy. When we are training the platform strategy, the environment of the customers will change and thus the learned policy could fail. All the above issues make these method less practical for building the Virtual Taobao.

In this work, we make Virtual Taobao through generating customers and generating interactions. Customers with search request, which has a very complex and widely spanned distribution, come into Taobao and trigger the platform search engine. We propose the GAN-for-Simulating-Distribution (GAN-SD) approach to simulate customers including their request. Since the original GAN methods often undesirably mismatch with the target distribution (**?**), GAN-SD adopts an extra distribution constraint to generate diverse customers. To generate interactions, which is the key component of Virtual Taobao, we propose the Multi-agent Adversarial Imitation

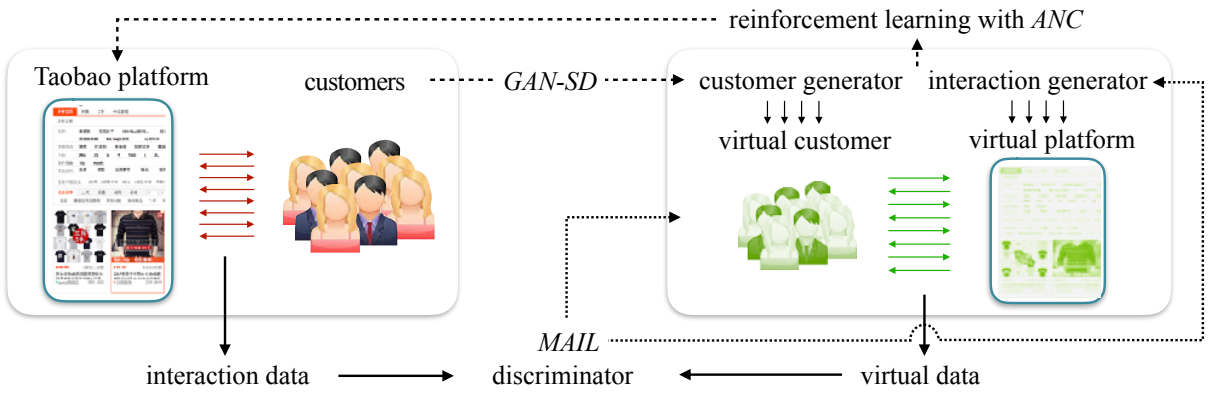


Figure 1: Virtual-Taobao architecture for reinforcement learning.

Learning (MAIL) approach. We could have directly invoked Taobao platform strategy in Virtual Taobao, which however makes a static environment that will not adapt to changes in the real environment. Therefore, MAIL learns the customers' policies and the platform policy simultaneously. In order to learn the two sides, MAIL follows the idea of GAIL (Ho & Ermon, 2016) using generative adversarial framework (Goodfellow et al., 2014). MAIL trains a discriminator to distinguish the simulated interactions from the real interactions; the discrimination signal feeds back as the reward to train the customer and platform policies for generating more real-alike interactions. After generating customers and interactions, Virtual Taobao is built. As we find that a powerful algorithm may over fit to Virtual Taobao, which means it can do well in the virtual environment but poorly in the real, the proposed Action Norm Constraint (ANC) strategy can reduce such over-fitting. In experiments, we build Virtual Taobao from hundreds of millions of customers' records, and compare it with the real environment. Our results disclose that Virtual Taobao successfully reconstructs properties very close to the real environment. We then employ Virtual Taobao to train platform policy for maximizing the revenue. Comparing with the traditional supervised learning approach, the strategy trained in Virtual Taobao achieves more than 2% improvement of revenue in the real environment.

The rest of the sections present the background, the Virtual Taobao approach, the offline and online experiments, and the conclusion, in order.

2 Background

2.1 Reinforcement Learning

Reinforcement learning (RL) solves sequential decision making problems through trial-and-error. We consider a standard RL formulation based on a Markov Decision Process (MDP). An MDP is described by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$ where $\mathcal{S}$ is the observation space, $\mathcal{A}$ is the action space and γ is the discount factor. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function to generate next states from

the current state-action pair. $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function. At each timestep t, the RL agent observes a state $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$ following a policy π. Then it will be transfered into a new state s_{t+1} according to $\mathcal{T}(s_t, a_t)$ and receives an immediate reward $r_t = \mathcal{R}(s_t, a_t)$. $R_t = \sum_{k=0}^{\infty} \gamma^t r_{t+k}$ is the discounted, accumulated reward with the discount factor γ. The goal of RL is to learn a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that solves the MDP by maximizing the expected discounted return, i.e., $\pi^* = \arg\max E[\, R_t \,]$.

2.2 Imitation Learning

It is apparently a more effective task to act following by a teacher rather than learning a policy from scratch. In addition, the manual-designed reward function in RL may not be the real one (Farley & Taylor, 1991; Hoyt & Taylor, 1981) and a tiny change of reward function in RL may result in a totally different policy. In imitation learning, the agent is given trajectory samples from some expert policy, and infers the expert policy.

Behavior Cloning & **Inverse Reinforcement Learning** are two traditional approaches for imitation learning: behavior cloning learns a policy as a supervised learning problem over state-action pairs from expert trajectories (Pomerleau, 1992); and inverse reinforcement learning finds a reward function under which the expert is uniquely optimal (Russell, 1998), and then train a policy according to the reward.

While behavior cloning methods are powerful for one-shot imitation (Duan et al., 2017), it needs large training data to work even on non-trivial tasks, due to compounding error caused by covariate shift (Ross & Bagnell, 2010). It also tends to be brittle and fails when the agent diverges too much from the demonstration trajectories (Ross et al., 2011). On the other hand, inverse reinforcement learning finds the reward function being optimized by the expert, so compounding error, a problem for methods that fit single-step decisions, is not an issue. IRL algorithms are often expensive to run because they need reinforcement learning in a inner loop. Some works have focused on scaling IRL to large environments (Finn et al., 2016).

2.3 Generative Adversarial Networks

Generative adversarial networks (GANs) (Goodfellow et al., 2014) and its variants are rapidly emerging unsupervised machine learning techniques. They are implemented by a system of two neural networks contesting with each other in a zero-sum game framework. They train a discriminator D to maximize the probability of assigning the correct labels to both training examples and generated samples, and a generator G to minimize the classification accuracy according to D. The discriminator and the generator are implemented by neural networks, and are updated alternately in a competitive way. Recent studies have shown that GANs are capable of generating faithful real-world images (**?**), implying their applicability in modeling complex distributions.

Generative Adversarial Imitation Learning (Ho & Ermon, 2016) was recently proposed to overcome the brittleness of behavior cloning as well as the expensiveness of inverse reinforcement

learning using GAN framework. GAIL allows the agent to interact with the environment and learns the policy by RL methods while the reward function is improved during training. Thus the RL method is the generator in the GAN framework. GAIL employs a discriminator D to measure the similarity between the policy-generated trajectories and the expert trajectories.

GAIL is proven to achieve the similar theoretical and empirical results as IRL and GAIL is more efficient. GAIL has become a popular choice for imitation learning (Kuefler et al.) and there already exist model-based (Baram et al., 2016) and third-person (Stadie et al., 2017) extensions.

3 Virtual Taobao

3.1 Problem Description

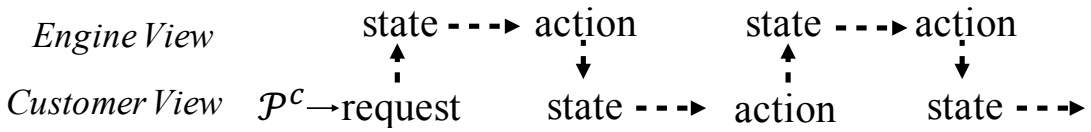
Commodity search is the core business in Taobao, one of the largest retail platforms. Taobao can be considered as a system where the search engine interacts with customers. The search engine in Taobao deals with millisecond-level responses to billions of commodities, while the customers' preference of commodities are also rich and diverse. From the engine's point of view, Taobao platform works as the following. A customer comes and sends a search request to the search engine. Then the engine makes an appropriate response to the request by sorting the related commodities and displaying the page view (PV) to the customer. The customer gives the feedback signal, e.g. buying, turning to the next page, and leaving, according to the PVs as well as the buyer's own intention. The search engine receives the signal and makes a new decision for the next PV request. The business goal of Taobao is to increase sales by optimizing the strategy of displaying PVs. As the feedback signal from a customer depends on a sequence of PVs, it's reasonable to consider it as a multi-step decision problem rather than a one-step supervised learning problem.

Considering the search engine as an agent, and the customers as an environment, commodity search is a sequential decision making problem. It is reasonable to assume customers can only remember a limited number, m, of the latest PVs, which means the feedback signals are only influenced by m historical actions of the search agent. Let $a \in \mathcal{A}$ denote the action of the search engine agent and F_a denote the feedback distribution from customers at a , given the historical actions from the engine $a_0, a_1, ..., a_{n-1}$, we have $F_{a_n|a_{n-1},a_{n-2},...,a_0} = F_{a_n|a_{n-1},...,a_{n-m}}$.

On the other hand, the shopping process for a customer can also be viewed as a sequential decision process. As a customer's feedback is influenced by the last m PVs, which are generated by the search engine and are affected by the last feedback from the customer. The customers' behaviors also have the Markov property. The process of developing the shopping preference for a customer can be regarded as a process optimizing his shopping policy in Taobao.

The engine and customers are the environments of each other. Figure 2 shows the details. In this work, PV info only contains the page index.

We use $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \pi \rangle$ and $\mathcal{M}^c = \langle \mathcal{S}^c, \mathcal{A}^c, \mathcal{T}^c, \mathcal{R}^c, \pi^c \rangle$ to represent the decision



	Engine view	Customer view
State	customer feature and request	customer feature, engine action and PV info
Action	parametermized as a vector in $\mathbb{R}^d$	buy, turn page or leave

Figure 2: Taobao search in engine view and in customer view

process for engine and customers. Note that the customers with requests come from the real environment $\mathcal{P}^c$.

For the engine, the state remains the same if the customer turn to the next page. The state changes if the customer sends another request or leaves Taobao. For the customer, As $\mathcal{S}^c = \mathcal{S} \times \mathcal{A} \times \mathbb{N}$, where $\mathbb{N}$ denotes the page index space, $s^c \in \mathcal{S}^c$ can be written as $\langle s, a, n \rangle \in \mathcal{S} \times \mathcal{A} \times \mathbb{N}$. Formally:

$$\mathcal{T}(s,a) = \begin{cases} s, & \text{if } a^c = \text{turn page} \\ s' \sim \mathcal{P}^c, & \text{otherwise} \end{cases} \qquad \mathcal{T}^c(s^c, a^c) = \begin{cases} \langle s', \pi(s'), 0 \rangle, s' \sim \mathcal{P}^c \text{ if } a^c = \text{leave} \\ \langle s, a, n+1 \rangle, \text{if } a^c = \text{turn page} \\ \text{terminates, if } a^c = \text{buy, or } n > \text{MaxIndex} \end{cases}$$

For the engine, if the customer buy something, we give the engine a reward of 1 otherwise 0. For customers, the reward function is currently unknown to us.

3.2 GAN-SD: Generating Customers

To build the Virtual Taobao, we need to firstly generate the customer, i.e. sample a user U^c that includes its request from $\mathcal{P}^c$ to trigger the interaction process. We aims at constructing a sample generator to produce similar customers with that of the real Taobao. It's known that GANs are designed to generate samples close to the original data and achieves great success for generating images. We thus employ GANs to tightly simulate the real request generator.

However, we find that GANs empirically tends to generate the most frequent occurring customers. To generate a distribution rather than a single instance, we propose Generative Adversarial Networks for Simulating Distribution (GAN-SD), as in Algorithm 1. Similar to GANs, GAN-SD maintains a generator G and a discriminator D. The discriminator tries to correctly distinguish the generated data from the training data by maximizing the following objective function

$$E_{p_{x\sim\mathcal{D}}}[\log D(x)] + E_{p_{z\sim\mathcal{G}}}[\log(1 - D(G(z)))],$$

while the generator is updated to maximize the following objective function

$$E_{p_{x\sim\mathcal{D}}, p_{z\sim\mathcal{G}}}[D(G(z)) + \alpha\mathcal{H}(V(G(z))) - \beta KL(V(G(z))||V(x))].$$

Algorithm 1 GAN-SD

Input: Real data distribution $\mathcal{P}_{\mathcal{D}}$
Initialize training variables θ_D, θ_G
for $i = 0, 1, 2...$ **do**
  for k steps **do**
    Sample minibatch from noise prior $p_{\mathcal{G}}$ and real dataset $p_{\mathcal{D}}$
    Update the generator by gradient:

$$\nabla_{\theta_G} E_{p_{x\sim\mathcal{G}}, p_{x\sim\mathcal{D}}}[D(G(z)) + \alpha\mathcal{H}(G(z)) - \beta KL(G(z)||x)]$$

  end for
  Sample minibatch from noise prior $p_{\mathcal{G}}$ and real dataset $p_{\mathcal{D}}$
  Update the discriminator by gradient:

$$\nabla_{\theta_D} E_{p_{x\sim\mathcal{D}}}[\log D(x)] + E_{p_{x\sim\mathcal{G}}}[\log(1 - D(G(z)))]$$

end for
Output: Customer generator G

$G(z)$ is the generated instance from the noise sample z. $V(\cdot)$ denotes some variable associated with the inner value. In our implementation, $V(\cdot)$ is the customer type of the instance. $\mathcal{H}(V(G(z)))$ denotes the entropy of the variable from the generated data, which is used to make a wider distribution. $KL(V(G(z))\|V(x))$ is the KL-divergence between the variables from the generated and training data, which is used to guide the generated distribution by the distribution in the training data. With the entropy and KL divergence constraints, GAN-SD learns a generator with more guided information form the real data, and can generate much better distribution than the original GAN.

3.3 MAIL: Generating Interactions

We generate interactions between the customers and the platform, which is the key of Virtual Taobao, through simulating customer policy. We accomplish this by proposing the Multi-agent Adversarial Imitation Learning (MAIL) approach following the idea of GAIL.

Different from GAIL that trains one agent policy in a static environment, MAIL is a multi-agent approach that trains the customer policy as well as the engine policy. In such way, the learned customer policy is able to generalize with different engine policies. By MAIL, to imitate the customer agent policy π^c, we should know the environment of the agent, which means we also need to imitate $\mathcal{P}^c$ and $\mathcal{T}^c$, together with reward function $\mathcal{R}^c$. The reward function is designed as the non-discriminativeness of the generated data and the historical state-action pairs. The employed RL algorithm will maximize the reward, implying generating indistinguishable data.

However, training the two policies, e.g., iteratively, could results in a very large search space,

Algorithm 2 MAIL

1: **Input:** Expert trajectories τ_e, customer distribution $\mathcal{P}^c$
2: Initialize variables κ, σ, θ
3: **for** $i = 0, 1, 2..., I$ **do**
4: **for** $j = 0, 1, 2..., J$ **do**
5: $\tau_j = \emptyset,\ s \sim \mathcal{P}^c, a \sim \pi_\sigma(s, \cdot), s^c = \langle s, a \rangle$
6: **while** NOT TERMINATED **do**
7: sample $a^c \sim \pi(s^c, \cdot)$, add (s^c, a^c) to τ_j, generate $s^c \sim \mathcal{T}^c_\sigma(s^c, a^c | \mathcal{P}^c)$
8: **end while**
9: **end for**
10: Sample trajectories τ_g from $\tau_{0 \sim J}$
11: Update θ to in the direction to maximize

$$E_{\tau_g}[\log(\mathcal{R}^c_\theta(s, a))] + E_{\tau_e}[\log(1 - \mathcal{R}^c_\theta(s, a))]$$

12: Update κ, σ by optimizing $\pi^c_{\kappa,\sigma}$ with RL in $\mathcal{M}^c$
13: **end for**
14: **Output:** The customer agent policy π^c

and thus poor performance. Fortunately, we can optimize them jointly. We parameterize the customer policy π^c_κ by κ, the search engine policy π_σ by σ, and the reward function $\mathcal{R}_\theta$ by θ. We also denote $\mathcal{T}^c$ as $\mathcal{T}^c_\sigma$. Due to the relationship of the two policies:

$$\pi^c(s^c, a^c) = \pi^c(\langle s, a, n \rangle, a^c) = \pi^c(\langle s, \pi(s, \cdot), n \rangle, a^c)$$

which shows that, given the engine policy, the joint policy $\pi^c_{\kappa,\sigma}$ can be viewed as $S \times \mathbb{N}$ to A^c. As $\mathcal{S}^c = \mathcal{S} \times \mathcal{A} \times \mathbb{N}$, we still consider $\pi^c_{\kappa,\sigma}$ as a mapping from $\mathcal{S}^c$ to $\mathcal{A}^c$ for convenience. The formalization of joint policy brings the chance to simulate π and π^c together.

We display the procedure of MAIL in Algorithm 2. To run MAIL, we need the historical trajectories τ_e and a customer distribution $\mathcal{P}^c$, which is required by $\mathcal{T}^c_\sigma$. In this paper, $\mathcal{P}^c$ is learned by GAN-SD in advance. After initializing the variables, we start the main procedure of MAIL: In each iteration, we collect trajectories during the interaction between the customer agent and the environment (line 4-9). Then we sample from the generated trajectories and optimize reward function by a gradient method (line 10-11). Then, κ, τ will be updated by optimizing the joint policy $\pi^c_{\kappa,\sigma}$ in $\mathcal{M}^c$ with a RL method (line 12). When the iteration terminates, MAIL returns the customer agent policy π^c.

After simulating the distribution $\mathcal{P}^c$ and policy π^c, which means we know how customers are generated and how they response to PVs, Virtual Taobao is build. We can generate the interactions by deploy the engine policy to the virtual environment.

3.4 ANC: Reduce Over-fit to Virtual Taobao

We observe that if the deployed policy is similar to the historical policy, Virtual Taobao tends to behave more similarly than if the two policies are completely different. However, similar policy means little improvement. A powerful RL algorithm, such as TRPO, can easily train an agent to over-fit Virtual Taobao which means it can performs well in the virtual environment but poorly in the real environment. Actually, we need to trade off between the accuracy and the improvement. However, as the historical engine policy is unavailable in practice, we can not measure the distance between a current policy and the historical one. Fortunately, we note that the norms of most historical actions are relatively small, we propose the Action Norm Constraint (ANC) strategy to control the norm of taken actions. That is, when the norm of the taken action is larger than the norms of most historical actions, we give the agent a penalty. Specifically, we modified the original reward function as follow:

$$r'(s,a) = \frac{r(s,a)}{1 + \rho \max\{\|a\| - \mu,\, 0\}} \tag{1}$$

4 Empirical Study

4.1 Experiment Setting

To verify the effect of Virtual Taobao, we use following measurements as indicators.

- **Total Turnover (TT)** The value of the commodities sold.
- **Total Volume (TV)** The amount of the commodities sold.
- **Rate of Purchase Page (R2P)** The ratio of the number of PVs where purchase takes place.

All the measurements are adopted in online experiments. In offline experiments we only use R2P as we didn't predict the customer number and commodity price in this work. For the convenience of comparing these indicators between the real and virtual environment, we deployed a random engine policy in the real environments (specifically, an online A/B test bucket in Taobao) in advance and collect the corresponding trajectories as the historical data (about 400 million records). Note that, we have no assumption of the engine policy which generates the data, i.e., it could be any unknown complex model.

We simulate the customer distribution $\mathcal{P}^c$ by GAN-SD with $\alpha = \beta = 1$. Then we build Virtual Taobao by MAIL. The RL method in MAIL we used is TRPO. All of function approximators in this work are implemented by multi-layer perceptions. Due to the resource limitation, we can only compare two policies at the same time in online experiments.

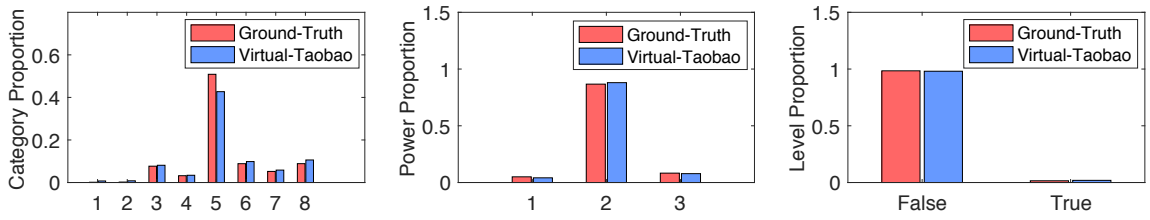


Figure 3: The customer distributions between Taobao and the Virtual Taobao.

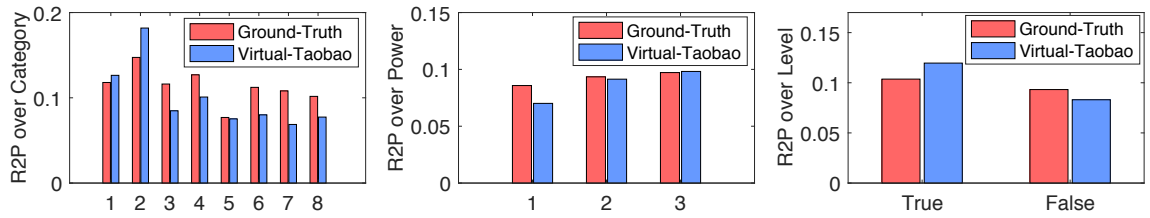


Figure 4: The R2P distributions between Taobao and the Virtual Taobao.

4.2 On Virtual Taobao Properties

4.2.1 Proportion & R2P over Features

The proportion of different customers is a basic criterion for testing Virtual Taobao. We generate 1,000,000 customers by $\mathcal{P}^c$ and calculate the proportions over three features respectively, i.e. query category (from one to eight), purchase power (from one to three) and high level indicator (True or False). We compare the result with the ground truth, i.e., proportions in Taobao. Figure 3 indicates that distribution is similar in the virtual and real environments.

People have different preference of consumption which reflects in R2P. We report the influence of customer features on the R2P in Virtual Taobao and compare it with the results in the real. As shown in Figure 4, the results in Virtual Taobao are quite similar with the ground truth.

4.2.2 R2P over Time

R2P of the customers varies with time in Taobao, thus Virtual Taobao should have the similar property. Since our customer model is independent of time, we divide the one-day historical data into 12 parts in order of time to simulate process of R2P changing over time. We train a virtual environment on every divided dataset independently. Then we deploy the same historical engine policy, i.e. the random policy, in the virtual environments respectively. We report the R2Ps in the virtual and the real environment. Figure 5 indicates Virtual Taobao can reflect the trend of R2P over time.

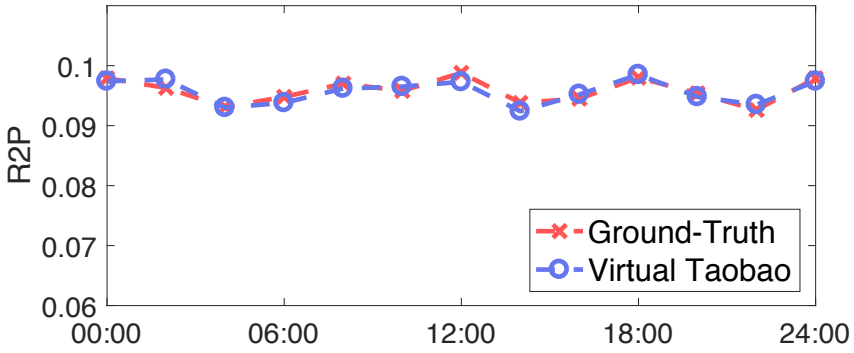


Figure 5: R2P over time

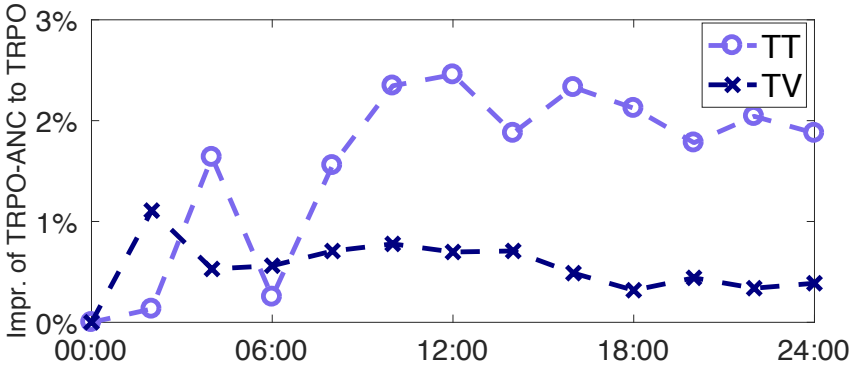


Figure 6: Impr. of TRPO-ANC to TRPO

4.3 Reinforcement Learning in Virtual Taobao

4.3.1 Generalization Capability of ANC

Virtual Taobao should have generalization ability as it is built from the past but serves for the future.

Firstly, we will show that the ANC strategy's ability on generalization. We train TRPO and TRPO-ANC, in which $\rho = 1$ and $\mu = 0.01$, in Vitual Taobao, and compare the results in real Taobao. Figure 6 shows the TT and TV increase of TRPO-ANC to TRPO. TRPO-ANC policy is always better than TRPO policy which indicates that the ANC strategy can reduce over-fitting to Virtual Taobao.

4.3.2 Generalization Capability of MAIL

Then, we'll verify the generalization ability of MAIL. We build a virtual environment by MAIL from one-day's data and build another 3 environments by MAIL whose data is from one day, a week and a month later. We run TRPO-ANC in the first environment and deploy the result policy in other environments and see the decrease of R2P. We repeat the same process except that we

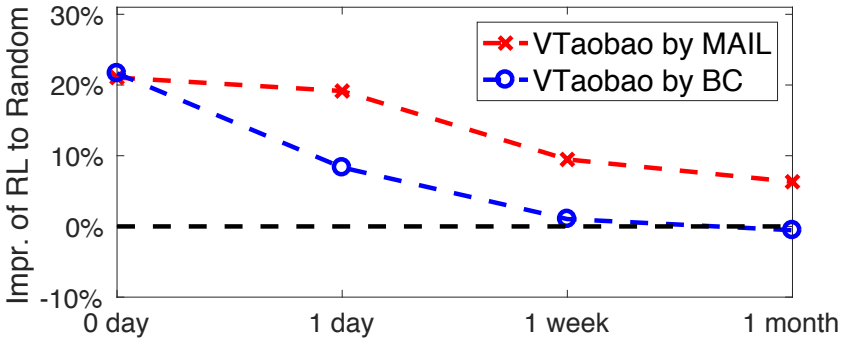


Table 1: Generalization capability of Virtual Taobao

replace MAIL with a behavior cloning method (BC). Figure 1 shows the R2P improvement of the policy trained in the two Virtual Taobao to a random policy. The R2P drops faster in the BC environment. The policy in BC environment even performs worse than then random policy after a month.

4.3.3 Online Experiments

We compare the policy generated by RL methods on Virtual Taobao (*RL+VTaobao*) with supervised learning methods on the historical data (*SL+Data*). Note that Virtual Taobao is constructed only from the historical data. To learn a engine policy using supervised learning method, we divide the historical data S into two parts: S_1 contains the records with purchase and S_0 contains the other.

The first benchmark method, denoted by SL_1, is a classic regression model.

$$\pi^*_{SL_1} = \text{argmin}_\pi \frac{1}{|S_1|} \sum_{(s,a)\in S_1} |\pi(s) - a|^2$$

The second benchmark SL_2 modified the loss function and the optimal policy is defined as follow

$$\pi^*_{SL_2} = \text{argmin}_\pi \frac{1}{|S_1|} \sum_{(s,a)\in S_1} |\pi(s) - a|^2 - \frac{\lambda_1}{|S_2|} \sum_{(s,a)\in S_2} |\pi(s) - a|^2 + \frac{\lambda_2}{|S|} \sum_{(s,a)\in S} |\pi(s)|^2$$

In our experiment, $\lambda_1 = 0.3$ and $\lambda_2 = 0.001$. As we can only compare two algorithms online in the same time due to the resource limitation, we report the results of RL v.s. SL_1 and RL v.s. SL_2 respectively. The results of TT and TV improvements to SL policies are shown in Figure 7 . The R2P in the real environment of SL_1, SL_2 and RL are 0.096, 0.098 and 0.101 respectively. The *RL+VTaobao* is always better than *SL+Data*.

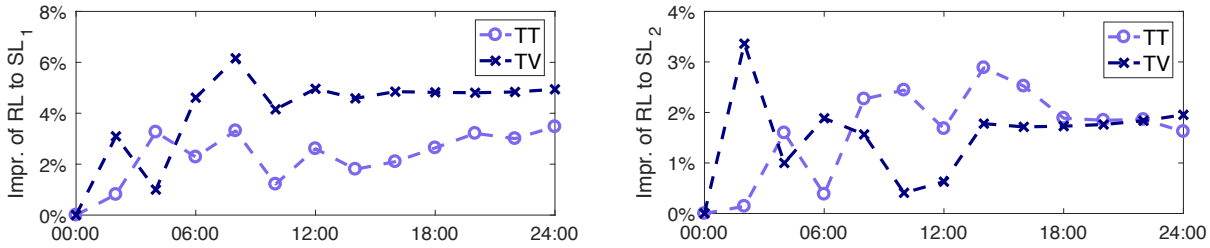


Figure 7: Improvement of RL+VTaobao to SL_1+data & SL_2+data in Taobao

5 Conclusion

To overcome the high physical cost of training RL for commodities search in Taobao, we build the Virtual Taobao simulator which is trained from historical data by GAN-SD and MAIL. The empirical results have verified that it can reflect properties of the real environment faithfully. We then train better engine policies with proposed ANC strategy in the Virtual Taobao, which has been shown to have better real environment performance than the traditional SL approaches. Virtualizing Taobao is very challenging. We hope this work can shed some light for applying RL to complex physical tasks.

References

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Baram, N., Anschel, O., and Mannor, S. Model-based adversarial imitation learning. *arXiv preprint arXiv:1612.02179*, 2016.

Chen, S.-Y., Yu, Y., Da, Q., Tan, J., Huang, H.-K., and Tang, H.-H. Stablizing reinforcement learning in dynamic environment with application to online recommendation. In *KDD'18*, London, UK, 2018.

Duan, Y., Andrychowicz, M., Stadie, B., Ho, J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. One-shot imitation learning. *arXiv preprint arXiv:1703.07326*, 2017.

Farley, C. T. and Taylor, C. R. A mechanical trigger for the trot–gallop transition in horses. *Science*, 253 (5017):306–308, 1991.

Finn, C., Levine, S., and Abbeel, P. Guided cost learning: Deep inverse optimal control via policy optimization. In *ICML'16*, New York, USA, 2016.

Gao, J. and Jamidar, R. Machine learning applications for data center optimization. *Google White Paper*, 2014.

Goodfellow, I., Pouget-Abadie, J., Mirza, M, Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *NIPS'14*, Montreal, Canada, 2014.

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *NIPS'16*, Barcelona, Spain, 2016.

Hoyt, D. F. and Taylor, C. R. Gait and the energetics of locomotion in horses. *Nature*, 192(16):239–240, 1981.

Hu, Y., Da, Q., Zeng, A., Yu, Y., and Xu, Y. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *KDD'18*, London, UK, 2018.

Kretzschmar, H., Spies, M., Sprunk, C., and Burgard, W. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research*, 35(11):1289–1307, 2016.

Kuefler, A., Morton, J., Wheeler, T., and Kochenderfer, M. Imitating driver behavior with generative adversarial networks. In *IV'17*, Los Angeles, USA.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *ICML'16*, New York, USA, 2016.

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

Ng, A. Y., Russell, S., et al. Algorithms for inverse reinforcement learning. In *ICML'00*, San Francisco, CA, 2000.

Pomerleau, D. A. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1992.

Ross, S. and Bagnell, D. Efficient reductions for imitation learning. In *AISTATS'10*, Sardinia, Italy, 2010.

Ross, S., Gordon, D., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS'11*, Lauderdale, FL, 2011.

Russell, S. Learning agents for uncertain environments. In *COLT'98*, Madison, Wisconsin, USA, 1998.

Schaal, S. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Stadie, B. C., Abbeel, P., and Sutskever, I. Third-person imitation learning. *arXiv preprint arXiv:1703.01703*, 2017.

Su, P., Gasic, M., Mrksic, N., Rojas-Barahona, L., Ultes, S., Vandyke, D., Wen, T., and Young, S. On-line active reward learning for policy optimisation in spoken dialogue systems. *arXiv preprint arXiv:1605.07669*, 2016.