Stabilizing Reinforcement Learning in Dynamic Environment with Application to Online Recommendation*

Shi-Yong Chen
National Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, Jiangsu, China
chensy@lamda.nju.edu.cn

Yang Yu
National Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, Jiangsu, China
yuy@nju.edu.cn

Qing Da
Alibaba Group
Hangzhou, Zhejiang, China
daqing.dq@alibaba-inc.com

Jun Tan
Alibaba Group
Hangzhou, Zhejiang, China
tanjun.tj@alibaba-inc.com

Hai-Kuan Huang
Alibaba Group
Hangzhou, Zhejiang, China
haikuan.hhk@alibaba-inc.com

Hai-Hong Tang
Alibaba Group
Hangzhou, Zhejiang, China
haihong.thh@alibaba-inc.com

ABSTRACT

Deep reinforcement learning has shown great potential in improving system performance autonomously, by learning from iterations with the environment. However, traditional reinforcement learning approaches are designed to work in static environments. In many real-world problems, the environments are commonly dynamic, in which the performance of reinforcement learning approaches can degrade drastically. A direct cause of the performance degradation is the high-variance and biased estimation of the reward, due to the distribution shifting in dynamic environments. In this paper, we propose two techniques to alleviate the unstable reward estimation problem in dynamic environments, the stratified sampling replay strategy and the approximate regretted reward, which address the problem from the sample aspect and the reward aspect, respectively. Integrating the two techniques with Double DQN, we propose the Robust DQN method. We apply Robust DQN in the tip recommendation system in Taobao online retail trading platform. We firstly disclose the highly dynamic property of the recommendation application. We then carried out online A/B test to examine Robust DQN. The results show that Robust DQN can effectively stabilize the value estimation and, therefore, improves the performance in this real-world dynamic environment.

KEYWORDS

reinforcement learning, dynamic environment, stratified sampling replay, approximate regretted reward, recommendation

*This work is supported by Jiangsu SF (BK20170013), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220122

ACM Reference Format:
Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. 2018. Stabilizing Reinforcement Learning in Dynamic Environment with Application to Online Recommendation. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220122

1 INTRODUCTION

Reinforcement learning aims at learning a policy from trial and error that maximizes the cumulative reward by interacting with the environment autonomously. It has achieved significant progress in solving optimal sequential decision problems autonomously, such as playing Atari games [10] and taking part of the AlphaGo system [17, 18] for the game of Go. Sequential decision problems are ubiquitous in real-world applications. For example, in an online recommendation system, it interacts with the customer as recommending items and receiving customer actions iteratively, with goal that the customer finds the desired item as soon as possible. Therefore, it is ideal to employ reinforcement learning in solve real-world problems.

However, applying reinforcement learning in online real-world environments faces many obstacles. One major obstacle is that real-world environments are commonly dynamic. But traditional reinforcement learning methods are designed to work in static environments. For examples, Atari games, where reinforcement learning can perform better than human players, all have static environments; even for the symmetric zero-sum game of Go, since it is in a discrete state space, the dynamic opposite agent can be eliminated with the help of the Monte Carlo tree search. However, in real-world applications such as online recommendation, the environment follows an unknown dynamic, and it is almost impossible to simulator all of the possible dynamic environments. As a result, we have noticed very few successful applications of reinforcement learning in such environments.

The dynamic property of environments brings direct challenge on the reward estimation in reinforcement learning. Taking online recommendation system as the example, the distribution of the customers changes. Therefore, in a batch of data collected from a short

period, the reward estimation can have a large variance. Moreover, the customers' behavior is naturally different in different time and evolving long time. When the agent receives an increase of the reward, it is unable to distinguish if the increase is the consequence of the previous action, or is due to the environment change. As a summary, the shifting distribution in a dynamic environment would lead to a high-variance and biased estimation of the reward, which probably mislead the reinforcement learning towards a poor performance.

To improve the reward estimation in dynamic environments, we propose two strategies in this paper. To address the variance caused by the customer distribution changes, we propose the *stratified sampling replay*, to replace the traditional experience replay that is one of the frequently used technologies in deep reinforcement learning. Stratified sampling replay introduces a prior customer distribution, and assesses any batch of replay according to the prior distribution. In such way the estimation variance can be significantly reduced. To address the reward bias caused by environment changes, it is ideal if the regret to the optimal decision is known, which is unfortunately hard to obtain. We introduce the approximate regretted reward by taking the reference baseline from a left alone sample of customers. We integrate the two strategies into a version of the classical Q-learning algorithm, i.e., Double DQN, resulting in the Robust DQN algorithm.

We apply Robust DQN to the tip recommendation in Taobao, which is a service providing customers "shortcut" keywords among the search results in order to refine the search queries. We firstly verify that for the recommendation task, the environment changes along time, as most online tasks. Then we conducted online A/B test comparing Robust DQN with the original Double DQN. The test results show that each of the proposed strategies can help improve the reward estimation effectively. Consequently, Robust DQN achieves improved performance in this dynamic environment.

The following sections describe in turn the background, the Robust DQN algorithm, the application to the tip recommendation in Taobao, the experiments, and the conclusion.

2 BACKGROUND

2.1 Reinforcement Learning

In reinforcement learning, the agent learns the policy that can maximize long-term cumulative reward through interacting with the environment [19]. Markov Decision Process (MDP) is usually adopted to study reinforcement learning which can be represented with a tuple of (S, A, T, R, γ), where S is the set of states, A is the set of action, $T(s_{t+1}|s_t, a) : S \times A \times S \to \mathbb{R}$ is the transition probability of reaching state s_{t+1} after executing action a on state s_t, $R(s, a) : S \times A \to \mathbb{R}$ is the immediate reward after executing action a from state s_t, and γ is the discount factor. The goal of reinforcement learning is to learning an optimal policy that maximizes the discounted accumulated rewards $\pi^* = \arg\max_\pi \mathbb{E}^\pi \{ \sum_{t=0}^{\infty} \gamma^t r_t \}$.

For deriving a policy, the state-action value function (Q value function) can evaluate how good it is to choose a particular action a when in a state s under a stochastic policy π, which can be defined as $Q^\pi(s, a) = \mathbb{E}^\pi \left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau | s_t = s, a_t = a \right]$.

Due to the Markov property, it can be calculated recursively with dynamic programming:

$$Q^\pi(s, a) = \mathbb{E}_{s'} \left[r + \gamma \mathbb{E}_{a' \sim \pi(s')} [Q^\pi(s', a')] \right],$$

where r is the immediate reward of executing action a on state s. Given the Q value function, a policy can be derived from the value as $\pi(s) = \text{argmax}_a Q^\pi(s, a)$. When the update equation converges,

$$Q^\pi(s, a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a' \sim \pi(s')} [Q^\pi(s', a')] \right],$$

the optimal Q value as well as an optimal policy is obtained.

2.2 Q-Learning

A major branch of reinforcement learning approaches focus on estimating the value function, among them Q-Learning [24] has attracted lots of attention for its simplicity as well as effectiveness. Q-learning in a discrete state space uses an online off-policy update,

$$Q^\pi(s, a) = Q^\pi(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)),$$

for the estimation of the Q value. Armed with deep neural network, Q-Learning demonstrates its superiority on the successful applications in the Atari Games [9], with its deep version coined deep Q-network (DQN) [11]. In DQN, the loss function at iteration i that needs to be optimized is the following:

$$L_i(\theta_i) = \mathbb{E}_{s, a, r, s'} \left[(\hat{y_i} - Q(s, a; \theta_i))^2 \right],$$

where $\hat{y_i} = r + \gamma \max_{a'} Q(s', a'; \theta')$, and θ' denotes the parameters of the separate *target network*, which is the same as the online network except that its parameters are copied every fixed number of steps from the online network $Q(s', a'; \theta_i)$, and kept fixed on all other steps. And the gradient update of the online network is as follows:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a, r, s'} \left[(\hat{y_i} - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Experience replay is another import ingredient of DQN as it can reduce the correlation among the samples and reduces the variance [11, 14]. The agent accumulates a buffer $\mathcal{D} = \{t_1, t_2, ...t_i\}$ with experiences $t_i = (s_i, a_i, r_i, s_{i+1})$ from many episodes. And the network would be trained by sampling from $\mathcal{D}$ uniformly at random instead of directly using the current samples. And the loss function can be expressed as:

$$L_i(\theta_i) = \mathbb{E}_{s, a, r, s' \sim u(\mathcal{D})} \left[(\hat{y_i} - Q(s, a; \theta_i))^2 \right]$$

However, both Q-learning and DQN are known to overestimate action values under certain conditions, as the max operator uses the same values to both select and evaluate an action [20, 21]. So the Deep Double Q-network (DDQN) is proposed to alleviate such issue, where the target is replaced as:

$$\hat{y_i} = r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta_i); \theta^-)$$

And the other parts are as the same as DQN. Some other variants of DQN include prioritized experience replay [16], dueling network [22], bootstrapped DQN [13] and etc. Although there are further improvements from DDQN to better estimate the Q value, we will base on DDQN due to its simplicity.

2.3 Related Work

Reinforcement learning has shown its strong learning ability on many issues, however, most of them are designed for the static environment. When these traditional approaches are used directly in the complex real world, the learning performance may degrade drastically due to the dynamic environments.

In order to apply reinforcement learning method to the dynamic environments, Wiering et al. proposed to instantiate information about dynamics objects in environment model and to replan when this information changes [25]. Nagayoshi et al. proposed a detection method of environmental changes to adapt to dynamic environments [12]. These methods try to model the environment or detect environmental changes, which are very difficult to use in real world environments. Abdallah et al. introduced repeated update Q-learning which aims to resolve the undesirable artifact of Q-learning in dynamic environment [1]. Pieters et al. proposed two variations of experience replay, where experiences are reused based on time or based on the obtained reward [15]. However, these methods do not pay attention to the problem of reward estimation caused by dynamic changes in the environment as mentioned before.

3 ROBUST DQN

3.1 Stratified Random Sampling

Stratified sampling is a probability sampling technique wherein the entire population would be broken up into different L subgroups, known as *Strata*, then the final subjects are randomly selected from the different strata [5, 7]. Suppose that for stratum l, there are N_l units from the population ($\sum_{l=1}^{L} N_l = N$) and the value for the units in stratum l are $X_{1l}, X_{2l}, \ldots\ldots X_{N_l l}$. Let $W_l = \frac{N_l}{N}$ and $\mu_l = \frac{1}{N_l}\sum_{i=1}^{N_l} X_{il}$. It is obvious that we can take a stratified random sampling of size n_l from each stratum ($\sum_{l=1}^{L} n_l = n$) instead of taking samples of n units from the total population. And the mean and variance of sampling the stratum l are:

$$\bar{X}_l = \frac{1}{n_l}\sum_{i=1}^{n_l} X_{il}, S_l^2 = \frac{1}{n_l - 1}\sum_{i=1}^{n_l}(X_{il} - \bar{X}_l)^2$$

Then an estimate of the population mean μ is

$$\bar{X}_S = \sum_{l=1}^{L}\frac{N_l}{N}\bar{X}_l = \sum_{l=1}^{L} W_l \bar{X}_l$$

And we have:

$$\mathbb{E}[\bar{X}_S] = \sum_{l=1}^{L} W_l \mathbb{E}[\bar{X}_l] = \sum_{l=1}^{L} W_l \mu_l = \frac{1}{N}\sum_{l=1}^{L}\sum_{i=1}^{N_l} X_{il} = \mu$$

This means that $\bar{X}_S$ is an unbiased estimate of μ. And the variance then is

$$Var(\bar{X}_S) = \sum_{l=1}^{L} W_l^2 Var(\bar{X}_l) = \sum_{l=1}^{L} W_l^2 \frac{1}{n_l}(1 - \frac{n_l - 1}{N_l - 1})\sigma_l^2 \quad (1)$$

where $\sigma_l^2 = \frac{1}{n_l}\sum_{i=1}^{n_l}(x_{il} - \mu_l)^2$. It can be seen from (1) that it is import to decide how to allocate for stratified sampling as it can depend $Var(\bar{X}_S)$ There are two main allocation schemes:

1) Proportional allocation: $\frac{n_1}{N_1} = \frac{n_2}{N_2} = \ldots = \frac{n_L}{N_L}$, which leads to: $n_l = n\frac{N_l}{N} = nW_l$. The sampling variance then is $Var(\bar{X}_{SP}) = \frac{1}{n}\sum_{l=1}^{L} W_l\sigma_l^2$, then the variance reduction can be written as:

$$\begin{aligned} Var(\bar{X}) - Var(\bar{X}_{SP}) =& \frac{1}{n}\left(\sum_{l=1}^{L} W_l\sigma_l^2 + \sum_{l=1}^{L} W_l(\mu_l - \mu)^2\right) - \frac{1}{n}\sum_{l=1}^{L} W_l\sigma_l^2 \\ =& \frac{1}{n}\sum_{l=1}^{L} W_l(\mu_l - \mu)^2 \geq 0 \end{aligned}$$

It suggests that stratified sampling with proportional allocation can lead to a smaller variance.

2) Optimal allocation: For a given total sample size n, we choose $n_1, n_2, \ldots, n_l$ to minimize $Var(\bar{X}_S)$, which gives $n_l = n\frac{W_l\sigma_l}{\sum_{k=1}^{L} W_k\sigma_k}$ and the sampling variance $Var(\bar{X}_{SO}) = \frac{1}{n}\left(\sum_{l=1}^{L} W_l\sigma_l^2\right)^2$. Then the variance reduction of $\bar{X}_{SO}$ over $\bar{X}_{SP}$ is

$$\begin{aligned} Var(\bar{X}_{SP}) - Var(\bar{X}_{SO}) =& \frac{1}{n}\sum_{l=1}^{L} W_l\sigma_l^2 - \frac{1}{n}\left(\sum_{l=1}^{L} W_l\sigma_l^2\right)^2 \\ =& \frac{1}{n}\sum_{l=1}^{L} W_l(\sigma_l - \bar{\sigma})^2 \geq 0 \end{aligned}$$

It can be seen that optimal allocation could bring smaller variance while it is necessary to know the variances for each strata to get the sample sizes as well as the fraction of units falling into each strata is only need for proportional sampling.

3.2 Stratified Sampling Replay

In the original DQN, experiences are stored in a replay memory, and the agent samples from the memory uniformly rather than using the current experience as standard temporal difference learning [11]. Previous experimental results show that experience replay has greatly improved the learning performance, partly because it can reduce the correlation between the samples. Now this strategy has been widely used in deep reinforcement learning [21, 23].

In a dynamic environment, it is improper to have a large replay pool from a long time period [15]. However, in a short time period, the customer distribution may not match the true distribution closely, or even has a large difference. For example, after some production promotion, particular customer group acts more than average temporally; more over, different customer groups are active usually in different time. If the agent is not aware of temporal difference, the estimated Q value could vibrate and has a large variance.

In order to alleviate the sample variance problem, stratified sampling replay strategy is proposed to replace the original experience replay. Firstly, from a long period of time, we select some attributes on which the customer distributions seems to be stable along time, such as gender, age, and geography. We count the customer ratios in each value of the multiple attributes. These attributes are used as the Strata. During learning, the agent still accumulates experience into memory as before. When these experience are used, stratified random sampling is conducted according to these Strata, rather than uniformly random sampling.

Stratified sampling replay strategy, as mentioned earlier, can effectively reduce the volatility caused by changes in the data distribution and reduce the variance of reward estimation, thereby improving the learning performance in dynamic environments. Moreover, stratified sampling replay allows us to use a larger replay pool from older samples, instead of only very recent samples, for achieving an even better performance.

3.3 Approximate Regretted Reward

Through stratified sampling replay strategy, the impact of the changes in data distribution could be reduced, which would lead to lower variance of reward estimation. However, in addition to the above mentioned problems, there will be many changes and fluctuations in the dynamic environments. For example, on some e-commerce platforms, the click-through rate of commodity may be higher at some time of the day and lower in other time periods, which is may be caused by the intrinsic properties of the physical world and has nothing to do with the ranking or recommendation algorithm. Obviously, this kind of change in the environment will lead to a very large fluctuation in the reward estimation, thus affecting the final learning performance.

In order to alleviate this problem, we propose another strategy: approximate regretted reward. In multi-armed bandit, the regret ρ after T rounds is defined as the expected difference between the reward sum associated with an optimal policy and the sum of the collected rewards: $\rho = T\mu^* - \sum_{t=1}^{T} r_t$, where μ^* is the maximal reward mean, $u^* = max_k \mu_k$, and r_t is the reward in round t [2]. The regret can measure the gap between the current policy and the optimal policy, thus can fairly evaluate the learning performance in dynamic environment. However, the optimal policy is not available in the real world, and naturally, the regret can not be obtained. In situations where the optimal policy is not available, we believe that the performance of other policy can also be used to measure the changes in dynamic environments, thereby reducing the reward fluctuation. In order to eliminate the influence brought by the change of the policy itself, we propose to apply an off-line training model and collect the average reward in real time which can measure the environmental conditions.

For example, for an online retail trading platform, we can randomly select a subset of the customers as our benchmark users. Then we apply a well-trained off-line model to these users and collect the averaged reward r_b in real-time, which can be regarded as a benchmark reward to reflect the real-time nature of the environment. At the same time, our reinforcement learning approach will be applied to other users and collect the immediate reward r_t. Finally, we choose to take the difference between r_t and r_b as our ultimate reward, which can measure the actual learning performance of our method by mitigating the impact of environmental fluctuations, as well as guide our method to perform better than the off-line model. It is worth noting that this approach can also be applied to other environments where there are large fluctuations.

3.4 Robust DQN Algorithm

Stratified sampling replay strategy and the approximate regretted reward are proposed in this paper to improve the reward estimation in dynamic environment. Applying the two strategies with Double DQN, we propose the Robust DQN as it not only takes advantage of the strong learning ability of Double DQN that has been demonstrated in many problems, but also can adapt to dynamic environment steadily.

Algorithm 1 Robust DQN

Input:
- N_r: Replay buffer maximum size
- N_b: Training batch size
- $N^{'}$: Target network replacement freq
- r_b: Real-time benchmark reward.

1: Initialize network parameters θ, $\theta^{'}$
2: $D \leftarrow \varnothing$
3: **for** each episode $e \in$ {1,2,3,...} **do**
4: **for** $t \in 0, 1, \ldots$ **do**
5: Obtain state s_t, action a_t, reward r_t, next state s_{t+1} from environment $\mathcal{E}$
6: Generate approximated regretted reward: $\tilde{r_t} = r_t - r_b$
7: $\mathcal{D} \leftarrow D \cup \{(s_t, a_t, \tilde{r_t}, s_{t+1})\}$
8: **if** $|\mathcal{D}| \geq N_r$ **then**
9: Replacing the oldest tuples
10: **end if**
11: Sample a minibatch of N_b tuples $(s, a, r, s') \sim SRS(\mathcal{D})$
12: Let $a^*(s'; \theta) = \text{argmax}_{a'} Q(s', a'; \theta)$, and for $1 \leq i \leq N_b$, calculate the target $\hat{y_i}$ as

$$\hat{y_i} = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^*(s'; \theta); \theta^{'}) & \text{otherwise} \end{cases},$$

13: Do a gradient descent to minimize $\sum_{i=1}^{N_b} ||\hat{y_i} - Q(s, a; \theta)||^2$
14: Replace target parameters $\theta^{'} \leftarrow \theta$ every $N^{'}$ steps
15: **end for**
16: **end for**

Algorithm 1 shows how Robust DQN works. From line 3 to line 5, the agent accumulates original transition tuple through interacting with environment $\mathcal{E}$. Then, the reward would be modified using the approximate regretted reward strategy in line 6 and the transition tuple $(s_t, a_t, \tilde{r_t}, s_{t+1})$ would be stored to replay buffer $\mathcal{D}$ in line 7. The network is trained by sampling mini-batches of experiences from $\mathcal{D}$ with stratified random sampling(SRS) method instead of traditional uniform random sampling in line 11, where the subgroups design depends on specific tasks. The other parts of this algorithm remains the same as Double DQN. Robust DQN with the proposed two strategies can alleviate the unstable reward estimation problem in dynamic environment from the sample aspect and the reward aspect.

4 APPLICATION TO TIP RECOMMENDATION IN TAOBAO

4.1 Tip Recommendation in Taobao

Customers can search for what they want from over one billion of commodities by entering a keyword, also called a query, on Taobao's search engine. However, queries are often not clear enough to limit the range of commodities that the customers really want,

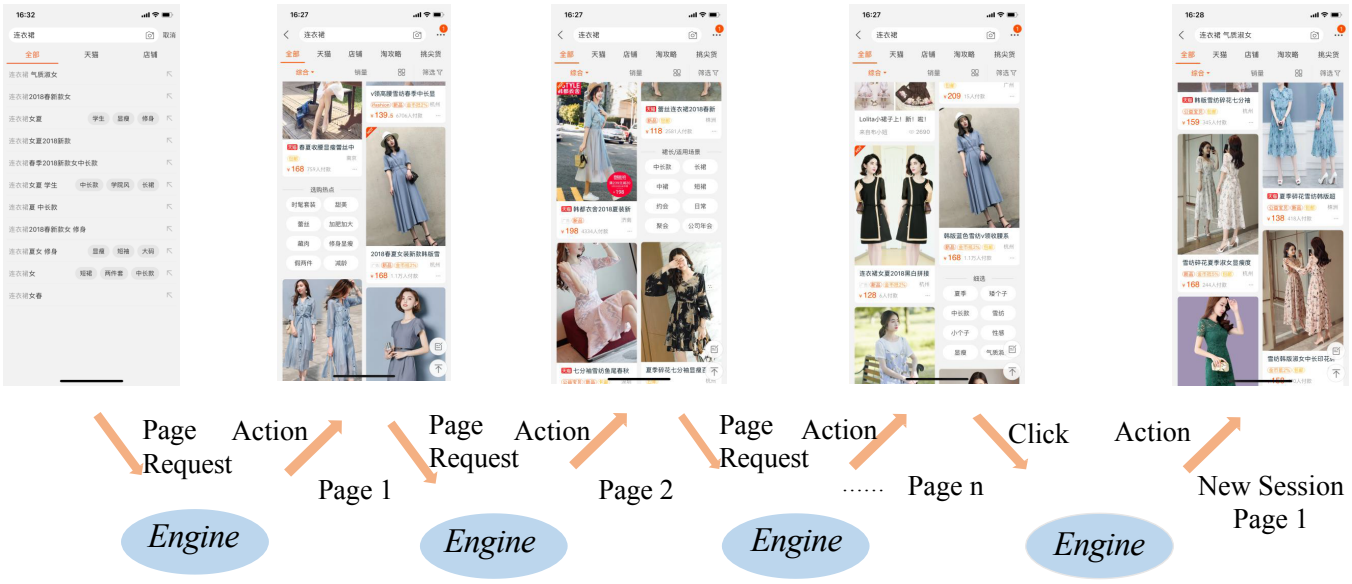


Figure 1: Tip Recommendation in Taobao.

which would lead customers to spend more time to satisfy their needs. For example, when a customer's query is "shirt", what he really wants may be just a long-sleeved black shirt while others will be ignored by him. To help customers find the needed commodity more quickly, Taobao offers a product called *tip*, which is actually quite a shopping guide to provide such options. Usually, a tip can be displayed as a text box with a lot of phrases. When the customer is browsing, a tip may be displayed among the commodities, and when one of the phrases in the tip is clicked, a new search request will be initiated by combining this phrase with the original query, and then jump to a new session, as demonstrated in Figure 1. In addition, a tip can also be a form of interactive question-and-answer, which may pose a question like "do you need a pair of blue pants?" with two candidate answers, "yes" and "no", as long as one of the answers is clicked, a new session will be entered similarly.

There are more than 20,000 types of tips on Taobao, each plays a different role. A common type is "age", which consists primarily of a number of age-limiting phrases, such as "18-24," "30-35," etc. Other types of tips, such as "sleeve length", "skirt length" and "style" are similar to the "age", except for the phrases displayed in it. It becomes very important to choose a suitable type of tip to display on the current page as there are too many types with different functions, which means that it is very necessary to design a recommendation system for the tip. A recommendation system that aims to recommend items that a user may be interested in among many items has been widely used in many websites and has changed the way of communication between websites and customers. There are many popular approaches for recommendation systems, the most widely used of which is collaborative filtering [4]. Among the applications of recommendation system, some are also implemented by using reinforcement learning, as it is not just a one-step prediction problem, but a continuous sequential decision problem [3]. Liebman et al. present a novel reinforcement-learning framework for

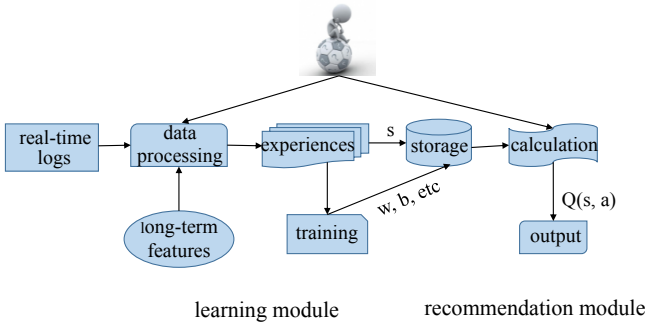


Figure 2: Architecture of the recommendation system.

music recommendation and it can provide a significant improvement over more straightforward approaches [8]. Hu et al. propose to use reinforcement learning to learn an optimal ranking policy in e-commerce search engine and the experimental results demonstrate that the algorithm performs much better than the state-of-the-art LTR methods [6].

However, traditional reinforcement learning approaches are applied in these problems without exploring the impact of non-stationary environment, which would reduce the learning performance. Robust DQN which is more adaptive to dynamic environment is applied here to the recommendation system with the desire to achieve better performance.

4.2 The Robust DQN based Tip Recommendation System

In order to make full use of customers' real-time feedback and adapt to delayed feedback, we choose reinforcement learning for tip recommendation in Taobao. Therefore, we have designed the

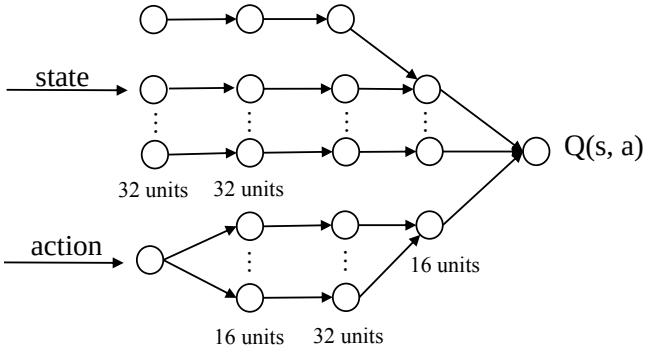


Figure 3: Network structure for tip recommendation.

framework of the recommendation system adapted to reinforcement learning approaches, as well as other components.

4.2.1 System Architecture. For security reasons, we divide the recommendation system into two parts: learning module and recommendation module, as shown in Figure 2. In learning module, the agent could obtain user logs, as well as user features, commodity features and so on in real time, which can be extracted into states, actions and rewards through data processing, so as to train and update the network. At the same time, the state and network parameters would be saved in the storage area.

The recommendation module is a part of the engine, which recomputes the Q value by obtaining information of state and network parameters in real time from the storage area, thereby determining the type of the tip that should currently be displayed.

4.2.2 State. A state is expected to express the customer's features, preference, and recent behavior. So the state includes: the customer's gender, age, purchasing power and some other basic features, the features of the current query, the customer preference for tip, as well as some more real-time features, such as the features of tip and goods that the customer recently clicks, etc. The dimension of a state is 156.

4.2.3 Action. The system aims to learn a policy that can determine which type of tip should be displayed on the current page, as well as the displayed tip would be clicked more, so the action here is designed to be an ID that can be mapped to a type of tip. Now there are over twenty thousand kinds of tip, which is the size of the action space.

4.2.4 Reward. Customers' click on a tip indicates that this display is valuable, implying that a positive reward should be given at this time, which aims to lead the system to provide more helpful guide to customers. In addition, the sooner the user clicks, the more valuable this display should be, as the tip is designed to guide users to find the needed goods faster. So the page number of the current tip displayed should also be considered in the reward design as: $r_1 = I * (1 + \rho * e^{-x})$, where I is 1 if click occurs else is 0, x is the current page number, and ρ is a coefficient.

An obvious fact is that different customers have great differences in the use of tip, some customers are very accustomed to using the tip, while others are rarely used, which means it's not fair to give them the same reward. A reasonable design is to offer a higher reward if the tip is clicked by users who rarely use it. So, we propose this form of reward: $r_2 = I * e^{-y}$, where y is the number of times the customers click on tip in the last 100 page views (PV).

Moreover, in order to promote the transaction, we give a positive reward when the customer has successfully purchased the product through the tip: $r_3 = c$, where c I is 1 if transaction occurs else is 0. Finally, reward can be expressed in the following form: $r = r_1 + \alpha * r_2 + \beta * r_3$, where α and β are coefficients.

4.2.5 Policy. In the traditional Double DQN, the Q values of all actions could be output in the last layer of the network, however, this is not applicable in our problem as there are more than 20,000 actions. In order to solve this problem, we adopt the following method: An action first goes through the embedding layer and then enters the network while the network outputs only the Q value of this action. Figure 3 shows the network structure used in this system.

There are also some preparations to do for applying Robust DQN in this problem. In order to use stratified sampling replay strategy, we need to determine which attributes to stratify on. Here, we choose gender, purchasing power, and gender with the addition of purchasing power, while the third refers to considering the both attributes, such as a customer is a woman with the first level of purchasing power. Moreover, it is necessary to know the variances for each strata to get the sample sizes for optimal allocation as well as the fraction of units falling into each strata is need for proportional sampling. In order to make a valid estimate, we collect the customer logs on Taobao in real time, and then we use the data of the recent seven days to estimate the mean and variance of each strata according to the above three attributes, so that the stratified sampling with the two allocation schemes can be realized. So far, the stratified sampling replay strategy with two allocation schemes and three attributes to stratify on has been completed.

In order to get approximate regretted reward, we randomly select a small number of users as benchmark users and we apply the offline model for tip recommendation among them. For the offline model, we can use the same network structure as reinforcement learning to expect similar responses to environmental changes. Then, we record the feedback of these users in real time to get the average reward. As mentioned earlier, it can be used to get approximate regretted reward to improve learning performance.

5 EXPERIMENTS

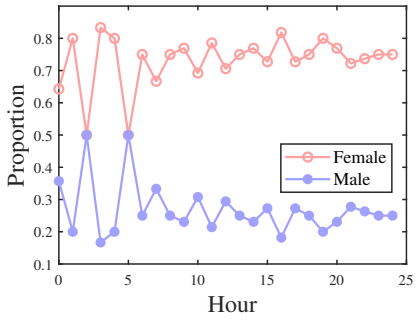
We apply Robust DQN in the tip recommendation system in Taobao and empirically evaluate it addressing the following questions:

$Q1$. Is this a dynamic environment that will deteriorate the learning performance as previously described?

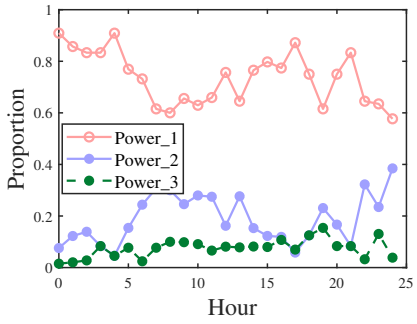
$Q2$. Is it appropriate to apply reinforcement learning to this problem? How does it compare with the offline supervised learning? How does it compare with the online supervised learning?

$Q3$. Stratified sampling replay is used to replace random replay, does this strategy work in the system?

$Q4$. Approximate regretted reward is also adopted to mitigate the adverse effects caused by shifting distribution, does it work in the system?

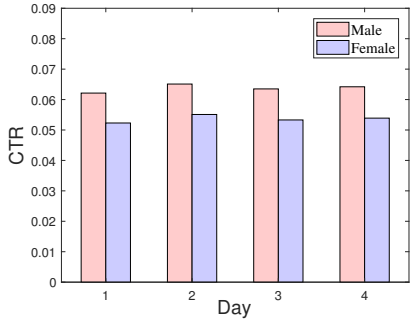


(a) Gender

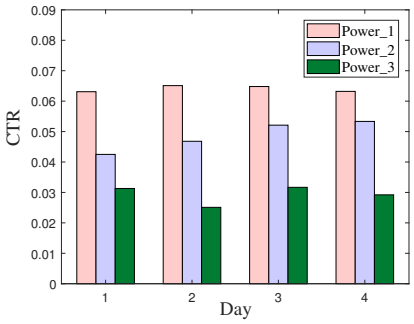


(b) Purchasing Power

Figure 4: Proportion of different types of customers in a given day.



(a) Gender



(b) Purchasing Power

Figure 5: CTR performance of different types of customers during 4 days.

$Q5$. Is it possible to achieve better performance if applying the proposed two strategies with Double DQN to this recommendation system?

5.1 Experiment Settings

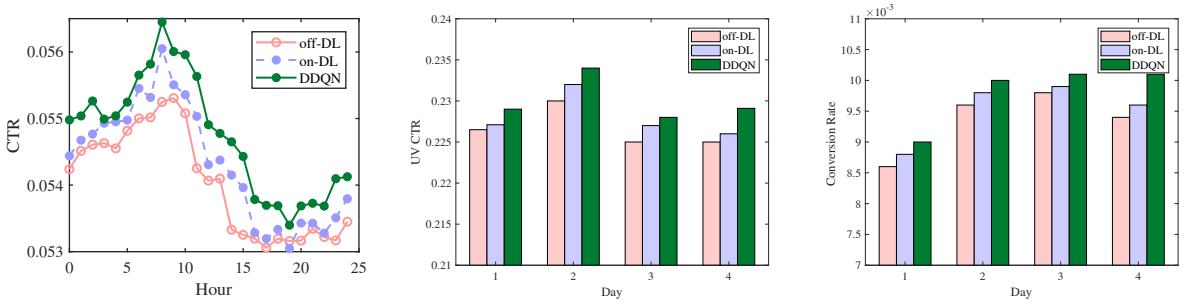
Bucket test or called A/B test is adopted for our experiments. All customers of Taobao are divided into many buckets in some way while the number and property of the customers in each bucket are basically the same. Other components of the system except the model, such as the position and frequency of tip display, remain the same in all the experiments, which aims to make an effective and fair comparison.

We first train an offline model, using the same network structure as in the Figure 3. As offline model can not introduce those real time features, some other features are used instead. And the label here indicates whether or not a tip is clicked. We denote this method as "off-DL". This is also the model used in the section of approximated regretted reward, which can provide benchmark reward for the reinforcement learning model. Besides, we also adopt a method of online learning denoted as"on-DL". Its features and network structure are exactly as the same as Robust DQN, while the label of it is just like "off-DL". And Double DQN is also adopted here as a comparison. In order to investigate the role of stratified sampling replay(SSR), experiments are conducted by applying it with Double DQN, while the proportional allocation as well as optimal allocation are examined, respectively, denoted as "SSR-PA" and "SSR-OA". We also test the performance of the second proposed strategy, approximate regretted reward, which is abbreviated to "ARR" here. The coefficients in the reward: α is 1, β is 0.5, and ρ is 0.5. For all of the experiments, the discount factor (γ) is 0.99.

For evaluation metrics, The following two are mainly considered: 1) CTR: CTR is the ratio of actual number of clicks to the number of Tip displayed; 2) UV CTR: UV CTR is the ratio of number of customers who click on Tip to the number of total customers who view Tip. CTR in our system can be calculated in real time, while UV CTR need to be obtained only by day. Due to the company's data security requirements, the data presented in the following figures are converted in some form.

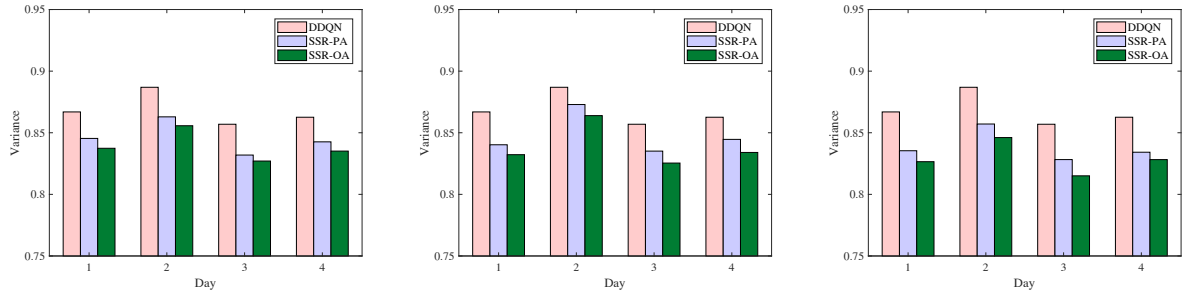
5.2 Experiment Results

To address $Q1$, we count the number of customers of different genders and purchasing powers at different times of a given day. There are two kinds of genders (male, female) and three kinds of purchasing powers (0-2). It can be seen from Figure 4 that the proportion of male and female customers is constantly changing in this day, as well as customers with different purchasing powers. And Figure 5 shows the click-through rate of different types of customers within four days while the model of this recommendation system comes



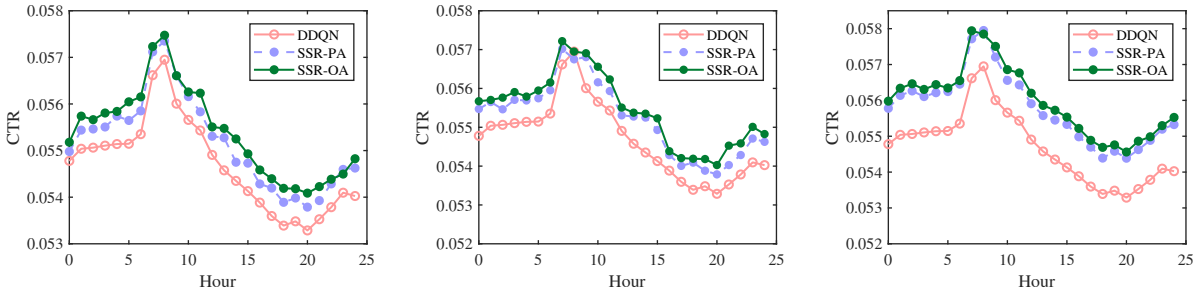
(a) CTR (b) UV CTR (c) Conversion rate

Figure 6: Comparion on the performance of off-DL, on-DL and DDQN.



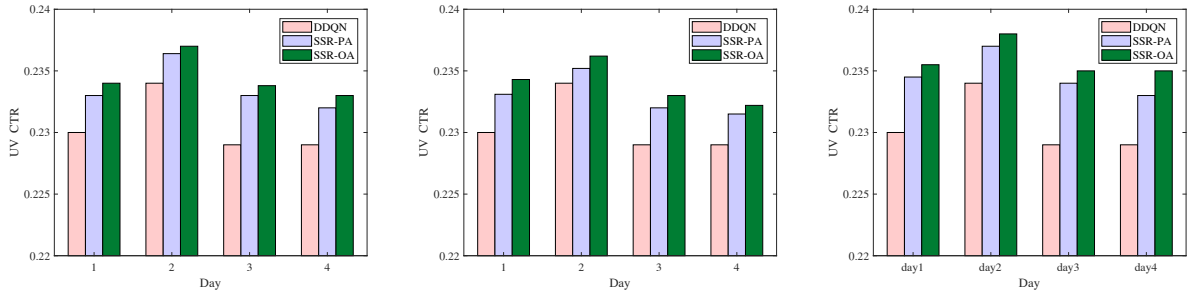
(a) SSR-Gender (b) SSR-Purchasing Power (c) SSR-Together

Figure 7: Comparion on the variance of stratified sampling replay (SSR) during 4 days.



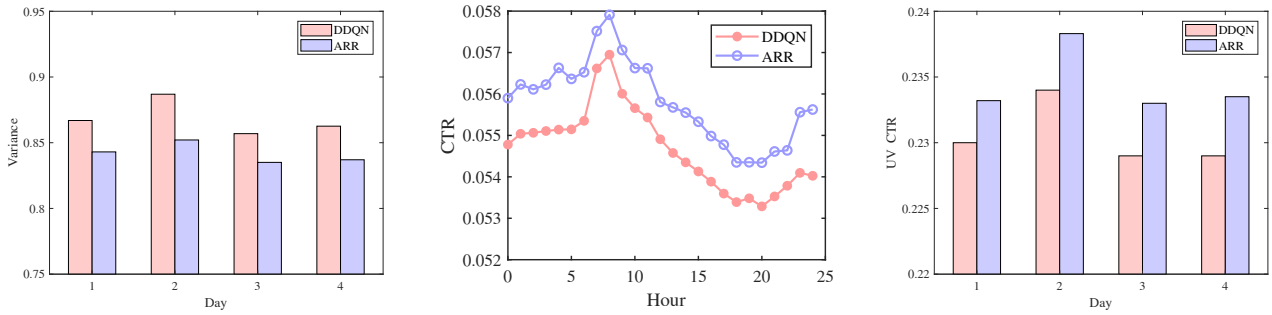
(a) SSR-Gender (b) SSR-Purchasing Power (c) SSR-Together

Figure 8: Comparison on the CTR performance of stratified sampling replay (SSR) during 24 hours.



(a) SSR-Gender (b) SSR-Purchasing Power (c) SSR-Together

Figure 9: Comparison on the UV CTR performance of stratified sampling replay (SSR) during 4 days.



(a) Variance (b) CTR (c) UV CTR

Figure 10: Comparison on the performance of approximate regretted reward (ARR).

from offline training, which is a fixed model. It can be seen that different types of customers have different usage patterns, and the proportion of different customers is also changing, which could really lead to a great change in the estimation of reward. Besides, off-DL in Figure 6 shows how the CTR changes in one day when the model of recommendation system is from off-line training, which further proves that this is a dynamic environment that could cause high-variance and biased reward estimation.

To address $Q2$, we first compare the performance of Double DQN, off-DL and on-DL, which can be seen in Figure 6. It can be observed that the three approaches have the consistent rank of performance in almost all of the time. Off-DL has the worst performance, as it lacks real time features and the model is always fixed. On-DL has a better performance than off-DL, as it knows more information and can update the model in real time. However, Double DQN still achieves the best performance, not only on the CTR, but also on the UV CTR. And then, we're going to focus on the conversion rate, which refers to the ratio of items purchased by customers after clicking on the tip. Figure 6(c) shows the conversion rate during 4 days and Double DQN also has the highest conversion rate. As mentioned earlier, Double DQN and on-DL use the same features and the same network structure, which implies that Double DQN can take into account the long-term impact and the characteristics of sequential decision making.

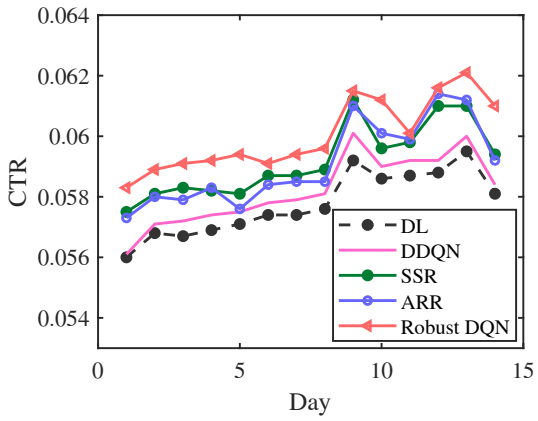
To address $Q3$, we explore the role of stratified sampling replay strategy. For our experiments, we carry out stratified sampling in three ways: according to gender, purchasing power, and gender with the addition of purchasing power, while they are denoted as "SSR-Gender", "SSR-purchasing power" and "SSR-Together". The third way will divide customers into six subgroups. Figure 7 shows the variance of reward over each method, which proves that SSR does reduce the variance effectively. In addition, it can be observed that the variance of SSR-OA is lower than SSR-PA, implying that the optimal allocation really plays a role. We also notice that SSR-Together has a higher CTR and UV CTR than SSR-Gender and SSR-Purchasing power, as shown in Figure 8 and 9, which means that this more fine-grained stratified sampling strategy can also make sense. Moreover, the rank of the performance of CTR and UV CTR is consistent with the rank of the variance of reward, which proves that the decrease in the variance of reward brings about an promotion of the performance.

To address $Q4$, we investigate the effect of approximate regretted reward. We firstly observe in Figure 10(a) that approximate regretted reward strategy can lead to a lower variance of reward, implying that it can improve the reward estimation in dynamic environments. And we also observe that it achieves a higher CTR and UV CTR as shown in Figure 10(b) and Figure 10(c). So it can be concluded that approximate regretted reward can really improve the learning performance by improving the reward estimation.

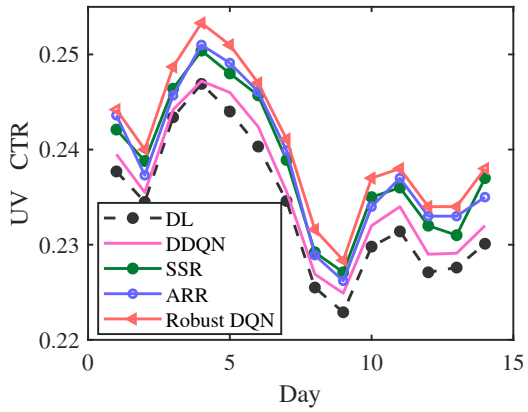
Finally, to address $Q5$, experiments are conducted by applying the proposed two strategies together with Double DQN, which is also called Robust DQN as mentioned before. For the stratified sampling replay strategy, the best performance comes from SSR-together with optimal allocation, so when this strategy is applied later, this is the way it would be used. Figure 11 shows the performance of Robust DQN and other methods during two weeks. Here, "DL" refers on-DL. It can be observed that DL has the worst performance every day, which is in line with our expectations. Moreover, we can observe that the stratified sampling replay and the approximate regretted reward both effectively improve the performance in two weeks, as explained earlier. And Robust DQN has achieved the best performance as it takes advantage of the previous two strategies, proving that it can better adapt to Taobao online retail trading platform which is really a highly dynamic environment.

6 CONCLUSION

This paper explores how to apply reinforcement learning to more complex real world and propose two strategies, stratified sampling replay and approximated regretted reward, to improve the reward estimation in dynamic environments. Then, we propose Robust DQN by applying the proposed strategies with Double DQN. Further, we apply Robust DQN in the tip recommendation system in Taobao which is a highly dynamic environment. Experiment results verify that stratified sampling replay and approximate regretted reward can evidently reduce the variance of reward and improve the performance. Moreover, the stratified sampling replay strategy would has better performance thanks to optimal allocation and more fine-grained stratified sampling, since they can reduce the variance further. Finally, when the proposed two strategies are used together, it improves the performance further, leading more customers to use the tip and more tip to be clicked. It is worth noting that the proposed strategies here can be used not only



(a) CTR



(b) UV CTR

Figure 11: Performance of these methods during 14 days.

in this system, but also in many dynamic environments, especially online systems and can lead to better learning performance.

The strategies proposed in this paper can improve the estimation of reward, thus effectively increasing the accuracy of recommendations, however, the distribution of observed data in real-world environments is still wide. Even if the exact same sequence of behavior might end up with a huge difference in rewards, meaning the variance of the reward is still very large, and we will study how to solve these problems effectively in the future.

REFERENCES

[1] Sherief Abdallah and Michael Kaisers. 2016. Addressing Environment Non-Stationarity by Repeating Q-learning Updates. *Journal of Machine Learning Research* 17 (2016), 46:1–46:31.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47, 2-3 (2002), 235–256.

[3] Sungwoon Choi, Heonseok Ha, Uiwon Hwang, Chanju Kim, Jung-Woo Ha, and Sungroh Yoon. 2018. Reinforcement Learning based Recommender System using Biclustering Technique. *CoRR* abs/1801.05532 (2018). http://arxiv.org/abs/1801.05532

[4] Debashis Das, Laxman Sahoo, and Sujoy Datta. 2017. A Survey on Recommendation System. *International Journal of Computer Applications* 160, 7 (2017).

[5] Mohammad Shahrokh Esfahani and Edward R. Dougherty. 2014. Effect of separate sampling on classification accuracy. *Bioinformatics* 30, 2 (2014), 242–250.

[6] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement Learning to Rank in E-Commerce Search Engine: Formalization, Analysis, and Application. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. London, UK.

[7] Andreas Karlsson. 2008. Survey sampling: theory and methods. *Metrika* 67, 2 (2008), 241–242.

[8] Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. 2015. DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*. Istanbul, Turkey, 591–599.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). http://arxiv.org/abs/1312.5602

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[12] Masato Nagayoshi, Hajime Murao, and H. Tamaki. 2013. Reinforcement learning for dynamic environment: a classification of dynamic environments and a detection method of environmental changes. *Artificial Life and Robotics* 18, 1 (2013), 104–108.

[13] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep Exploration via Bootstrapped DQN. *CoRR* abs/1602.04621 (2016). http://arxiv.org/abs/1602.04621

[14] Joseph O'Neill, Barty Pleydell-Bouverie, David Dupret, and Jozsef Csicsvari. 2010. Play it again: reactivation of waking experience and memory. *Trends in neurosciences* 33, 5 (2010), 220–229.

[15] Mathijs Pieters and Marco A. Wiering. 2016. Q-learning with experience replay in a dynamic environment. In *2016 IEEE Symposium Series on Computational Intelligence*. Athens, Greece, 1–8.

[16] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized Experience Replay. *CoRR* abs/1511.05952 (2015). http://arxiv.org/abs/1511.05952

[17] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

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

[19] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

[20] Hado van Hasselt. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems 24*. Vancouver, British Columbia, 2613–2621.

[21] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. Phoenix, Arizona, 2094–2100.

[22] Ziyu Wang, Nando de Freitas, and Marc Lanctot. 2015. Dueling Network Architectures for Deep Reinforcement Learning. *CoRR* abs/1511.06581 (2015). http://arxiv.org/abs/1511.06581

[23] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. 2016. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33th International Conference on Machine Learning*. New York City, NY, 1995–2003.

[24] Christopher John Cornish Hellaby Watkins. 1989. *Learning from delayed rewards*. Ph.D. Dissertation. King's College, Cambridge.

[25] Marco Wiering. 2001. Reinforcement Learning in Dynamic Environments using Instantiated Information. In *Proceedings of the 18th International Conference on Machine Learning*. Williamstown, MA, 585–592.