**1. A deep reinforcement learning framework**

In this section we first review the Markov Decision Process(MDP). Then formulate a joint pricing and inventory control problem for a perishable product. And a simulator is established to interact with the agent, retailer represents the agent. Last the two deep reinforcement learning methods are presented to get the optimal policy.

C:\Users\Tony\AppData\Local\Temp\1566977943(1).png

C:\Users\Tony\AppData\Local\Temp\1566977943(1).png

Figure 1. an agent interacts with environment (Sutton & Barto, 2018)

**1.1 Markov Decision Process**

Bellman (1957) proposed the Markov Decision Process, or MDP, MDP is a process with Markov property, that is, the conditional probability of the process is only related to the current state of the system, but independent and irrelevant to its past history or future state.

*P*[*St+1|St*] *= P* [*St+1|S1, …, St*]

MDP is a decision process based on Markov Reward Process (MRP), which composed of five elements: <S, A, P, R, γ>. S is a set of all the states, or state space, A is a set of all actions, or action space, P is state-transition matrix, which describes the probability from state (s∈S) takes action a (a∈A) transmit to state , which we can denote as a three-argument function *p*:*SSA*[0, 1] (Sutton & Barto,2018) or

= *P* [*St+1= s’ |St=s, At=a*]

, R is a reward function which is rely on the current state and action, and γ is a discount factor, in MDPs at time t, current immediate rewards are more valuable than future reward, so we need to discount future rewards to the present.

The MDP framework is abstract and flexible and can be applied to many different issues in a variety of ways and the MDPs is an idealized mathematical form of reinforcement learning problem, which can be accurately described theoretically (Sutton & Barto,2018). Reinforcement learning is learning what to do-how to map situation to actions-so as to maximize the reward signal, which is based on MDPs, so solving MDP is to find the best policy , which deterministically maps the state to an action, and there is always a definite optimal strategy for MDP. Hence, find the optimal policy \*, one can get the maximum cumulative expected rewards (Kochenderfer, 2015).

**1.2 problem formulation**

Here we study a periodic-review single perishable product inventory system over a finite horizon of T periods. As we know perishable products have finite lifetime, set as *l*, and order lead time, set as *k*, (for simplicity, we set k =1 in this research)*, k < l.* In each time interval, each order has a variable cost *c* and a single price *p* is charged for inventory of different ages, here we assume that the customer is not sensitive to the lifetime of the product, as long as the lifetime is acceptable, hence first-in-first-out (FIFO) policy (Nahmias 2011) is optimal. What’s more, in each period, demand is always met to the maximum extend with the on-hand inventory and we assume that the unmet demand is lost, set *u* as the unmet cost. And inventory to be carried to next period generates holding cost, set as *h*, inventory to be outdating and disposed generates outdating cost, set as *ν.* So the object is to maximize long-term expected discounted profit through dynamic ordering, pricing in the planning horizon.

The problem can be modelled as an MDP game, with the states, actions, state transitions, and rewards defined as follows.

**An observation**. Here we assume the age of inventory is counted from the period when the order is placed and the beginning of each time window, an observation integrates inventory and age of the products on hand and in-transit can be observed. The observation here is an (*l*-1)-dimensional vector**,** which represents stock and age condition after receiving the order placed k periods ago but before placing an order, and the observation is given as below.

**= (***s1, . . . , sl-1* **)**

Here *si*indicates that the remaining life does not exceed the total inventory of the *i* period. And *sl-1*is the inventory position and *sl-k* is on-hand inventory,*0 ≤ s1≤ s2 ≤ … ≤ sl-1*. A typical state (10,20,30), 10 indicates the amount of stock that does not exceed 1 period, 20 indicates the amount of stock that does not exceed 2 periods, which includes the stock lifetime is 1, that is, stock with lifetime 2 is 10, 30 is the same.

**1.2.1 Action Space**

In each time interval, action space contains ordering and pricing decision. For simplify we set the bounds for ordering quantity() and pricing(*p*∈[]), more details in §1.3.

**1.2.2 State Space**

The state of the agent can be simply defined as its current observation vector for general condition, i.e. order lead time is zero. However, we consider a positive lead time case (k=1), which make this problem more complicated and cannot be captured by a single observation, so we define the state to be a sequence of interleaved observations and actions, i.e. , therefore state here is a [(*l*-1)+1]\*k+(*l*-1)]-dimension vector, where k is lead time, and is order quantity.

**1.2.3 State Transition**

When state changes from current to next state, the state of agent gets updated according to the action () and interaction with environment. For state here is a vector consisting of a set of observations () and actions (), we get the observation transition can express the state . The observation transition is as follows.

First, in current time interval *t*, the observation of agent is ***Ot***= (*s1, . . ., sl-k, sl-k+1, …, sl-1*), and agent takes an action *At*and meet the demand *d* from the simulator (which will be presented in next section). Then agent get a reward and next observation ***Ot+1***= (*ŝ1, . . ., ŝl-k, ŝl-k+1, …, ŝl-1*), here ***Ot+1*** is obtained by the following rules by Chen et al. (2014), but a little different: (1)when *d* *≤ sl-k*, then = for *i = 1, …, l-k-1,* and = for *j = l-k, …, l-1*; (2) when d ≥ *sl-k*, then = 0 for *i = 1, …, l-k-1,* and = for *j = l-k, …, l-1.*

**1.2.4 Reward Function**

The presented perishable inventory joint pricing management aims to maximize the accumulative reward by dynamic ordering and pricing in planning horizon. In each period when agent makes ordering and pricing at the very beginning, agent will get a corresponding return and by trial-and-error many periods, agent can tell the difference between the action by the reward. Thus the reward function is determined as follows:

Where means on-hand inventory.

**1.3 Simulator**

One of the most important component for reinforcement learning is an environment, which needs to interact with the agent and train the agent, and single-agent and multi-agent reinforcement learning are shown to achieve great performance in such artificial environment.

In this section, we design a simulator to dynamically represent the environment. Almost all researches in perishable inventory joint pricing management set demand is price-sensitive and demand is a function of price, take an *additive* form or a *multiplicative* form and demand is independent in different time intervals (see, e.g., Chen & Simchi-Levi 2004a, b). For simplicity, here we exploit a special additive form, that is, demand in period *t* is given as follows (Petruzzi & Dada 1999; Chen et al. 2014):

*,*

where is the expected demand in period *t* and is strictly decreasing in selling price *p* in this period, and is a random variable with zero mean. But here we don’t fully adopt this form, like many reinforcement learning researches we set the demand satisfies the Poisson distribution (see, e.g., Rana & Oliveira 2015) and with a parameter , and

,

which makes the demand more reasonable and more like the real world, and can get a more robust performance. As mentioned above, for simplicity, *p* is bounded and only takes integers, for *p∈[]*, hence *d∈[],* where .

The sequence of events in time period t is as follows, noted there will be a designed initial stock.

* At the beginning of time period t, get the order place *k* (*k =*1) periods ago, and initialize state here on-hand stock and different ages is observed.
* Based on on-hand stock and make an ordering (*oq∈[]*), pricing (*p*∈[]).
* During time interval, demand arrives, which is generated by Poisson distribution, and parameter *dt* is a function of *p*.
* Last we get the reward *rt* , and next state , all inventory’s lifetimes decrease by one.

Given this setting, we may find an optimal benchmark to test our final performance in a sense. The benchmark is an inventory system with zero lead time, which means in each time interval the order action is instantaneous, and the order quantity is a -quantile of the demand distribution based on price action, i.e. an order quantity of newsvendor model with known demand distribution (Scarf 1958). And this optimal performance is the agent makes arduous efforts to reach, although there maybe still some unreasonable place, this can be a useful metric to gauge the performance of agent.

**1.4 Deep reinforcement learning methods**

As the name implies, DRL is the introduction of deep learning into reinforcement learning. The objective of reinforcement learning is to find the optimal policy Π\* to achieve a maximum accumulative reward for agent and Q-learning (Watkins,1989) is a popular and widely used RL method, it iteratively determines the value of taking an action in a special state and estimates the expected total discounted reward of state-action pairs in a Q-table by Bellman equation. But many real problems have a large state and action space, which can not be all recorded by Q-table, it’s Inefficient and wasteful. In order to solve the problem, Mnih et al. (2015) proposed a new RL method based on Q-learning named Deep Q-network (DQN), which used a neural network to approximate the state-action values. Thus, DQN is suitable for our research to solve the problem of large state space and action space. The details of the algorithm of proposed DQN are shown in ***Algorithm 1***. DQN has two characteristics: fixed Q-target and experience replay. There are two same neural networks and have same initial parameter, one is target-net has fixed parameter used to get the Q-target values and another one is evaluate-net (eval-net) used to model the behavior of agent. Eval-net uses backpropagation algorithm to update the parameter by minimizing the difference between Q-target values and Q-eval values. And the training data is random sampled from a memory pool, which records the actions, rewards, and results of the next state in each state *(s, a, r, s')*. The size of the memory pool is limited. When the data is full, the next data will overwrite the first data in the memory, the memory pool is updated like this. And randomly extracting data from the memory pool for learning, disrupting the correlation between experiences, making neural network updates more efficient, and fixed Q-targets allows target-net to delay updating parameters and thus disrupt correlation.

Algorithm 1 Deep Q-learning (DQN) with experience replay

1: Initialize replay memory pool D to capacity N

2: Use random weights to initialize the action-value function *Q* (eval-net)

3: Initialize target action-value function with weights =

4: **For** epoch = 1 to number of epochs **do**

5: Reset the environment and initialize state s0

6: **For** *t =* 1, *T* **do**

7: With probability select a random action *at*, otherwise select

*at* = ()

8: Execute action *at* in simulator and observe reward *rt* and

9: Set

10: Store transition *(, at, rt, st+1)* in D

11: Sample random mini-batch of transitions *(, ai, ri, si+1)* from D

12: Set *yi*

13: Perform a gradient decent step on *( yi -)2* with

respect to the network (eval-net) parameters

14: Every C steps reset

15: **End for**

*16:* ***End For***

The second reinforcement learning method is Actor-Critic (A2C). Actor-Critic (A2C) is a popular deep reinforcement method and combines two types of reinforcement learning algorithms, Value-based (such as Q-learning) and Policy-based (such as Policy Gradients). Here A2C constructs two networks, that is, policy network and value network. Policy network is known as actor, which used to output policy π(*a/s*), and value network is known as critic, which used to evaluate the performance of the policy and return *TD-error*, thus actor adjusts its policy by *TD-error*. The parameters of value network is update by minimizing a loss function and an advantage function *A*(from TD-error) is introduced here to update the parameters of policy network,.

The details of the Actor-Critic(A2C) is illustrated in ***Algorithm 2.***

*Algorithm 2 Advantage Actor-Critic*

1: Use random weights , to initialize the policy network and value network

2: **For** epoch = 1 to number of epochs **do**

3: Reset the environment and initialize state s0

4: **For** *t =* 1, *T* **do**

5: Sample action of agent, *at* based on action probability *P()*

6: Execute action *at* in simulator and observe reward and

7: Update the parameters of value network by minimizing a loss function *L () = (-* *()),* where

8: Get td-error and advantage function *A (st, at)= rt+\*(st+1)-(st)*

9: Update the policy network parameters*J*() where *J*() = *log(at/st)A(st, at)*

10: Set =

11: **End for**

*12:* ***End for***

**2. Experiments**

In this section, we conduct the simulator we designed before to evaluate our proposed DRL methods on perishable inventory joint price management and sensitivity analyses are also conducted to investigate the impacts of the key parameters.

Here the planning horizon *T* is 30 days, and the other experimental factors for simulation are given in **Table 1**. Here lifetime of products is assumed to be 2, 3 and 4 periods, respectively, which can test whether long lifetime has a benefit in accumulate profit in planning horizon. And lead time is set to be 0 and 1, for we study a positive lead time inventory management and 0 is to investigate the effect of lead time on accumulate profit in planning horizon. Demand, as mentioned before, is a Poisson distribution and *λ* is a function of price *p*, and *p* is bounded, , here for the simplicity of computing and computer processing the function is set to be *λ* = 50 – 3*p*. The ordering quantity here is bounded in [0,30]. What’s more, initial cost here includes variable cost *c* is 6, unmet demand cost *u* is 4, disposal cost *v* is 3, and holding cost *h* is 1.

**Table 1** Parameters for simulation experiment

*Parameters Values*

Product lifetime *l* {2, 3, 4}

Lead time *k* {0, 1}

Price *p* [10,14]

Poisson distribution *λ* (*λ =* 50 – 3p)

Order quantity (*oq)* [0,30]

Initial cost (*c*, *h, u,* *ν*) (6, 1, 4, 3)

In DRL the effect of hyper-parameters on the final result is very large, so we need to set the relevant parameters, exploration rate , learning rate and discount factor , more details in **Table 2** . In an -greedy policy, set the initial is 0.9 and 1, and the value of linearly decrease and takes search-then-convergence procedure suggested by Darken et al. (1992).

where *y* = , is initial . And learning rate here we assumed to be 0.1, 0.01 and 0.001, respectively. Discount factor here is 0.9 and 0.95.

**Table 2** Hyper-parameters for simulation experiment

*Parameters Values*

Initial exploration rate () {0.9, 1}

Learning rate () {0.1, 0.01, 0.001}

Discount factor () {0.9, 0.95}

*Decay parameters for exploration () {1104, 1105, 1106}*

**2.1 Results and analysis**

**2.1.1** **Results on different DRL methods in designed environment and compared to benchmark**

After simulate twenty thousand times, the performance (mean epochs reward) of two proposed DRL methods, DQN with experience replay and Advantage Actor-Critic (A2C) on different lifetime is presented as follows.

From **Figure 2** below we can see the trend of epochs mean reward for three different lifetime with positive (k=1) lead time. Results show that as the times go on, the returns increase, indicating that the agent has learned something. And from **Table 3** we can see that as the lifetime increases from 2 to 4, two DRL methods final mean rewards also increase, which is in line with expectations, because the longer the life time is, the more similar it is to ordinary goods, and the cost of expiration will be smaller and smaller, when lifetime is 4, the ratio of mean epochs revenues to optimal mean epochs reward can reach 98% and 99% for two DRL methods, respectively, and **Figure 3** shows the variation process for two DRL methods in different lifetimes. And we also find that with the increase of lifetime, the increment becomes smaller and smaller, which has been consistent with expectations, the longer the lifetime, the closer it is to ordinary goods, **Figure 4** shows the changing process for DQN method. What’s more, DQN with experience replay always better than A2C from a long view.

**Figure 5** shows the epochs revenue variation for two DRL methods in three different lifetimes. **Figure 5** (a) (b) (c) are for three different lifetimes respectively, and the first subfigure for each lifetimes is the performance of A2C method and A2C optimal, the second one is for DQN and DQN optimal, and the last one is comparing the two DRL methods. From the figure we can see that compared with the ordinary inventory control system, although we increased the action space, but also quickly reached the convergence and stable. And two DRL methods have better performance as lifetime goes on, and when lifetime is 4, the yield curve almost coincides with the optimal yield curve. In addition, A2C method can reach its optimum more quickly and more stable than DQN, but from third subfigure of each lifetime and some figures before we can get DQN is always better than A2C.

**Table 3** Results after twenty thousand times

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Methods | Lifetime | Mean epochs reward | optimal mean epochs reward | Best rate to optimal |
| A2C | 2 | 1206.71 | 1577.55 | 0.764926627 |
|  | 3 | 1280.88 | 1785.25 | 0.717479345 |
|  | 4 | 1772.49 | 1831.78 | 0.967632576 |
| DQN | 2 | 1408.73 | 1574.47 | 0.894732831 |
|  | 3 | 1848.29 | 1880.34 | 0.982955210 |
|  | 4 | 1846.77 | 1855.77 | 0.995150261 |

Figure 2. Epochs mean reward for two methods

Figure 3. Epochs mean reward to optimal for two methods

Figure 4. Epochs mean reward for different lifetime

Figure 5. Epochs reward for different lifetime

**2.1.2 Sensitivity analysis in terms of hyper-parameters**

Next we do sensitivity analysis to look at the effects of learning rate () and exploration parameters () for the training of the proposed deep reinforcement learning, respectively. **Figure 6** demonstrates the mean epochs reward for three different learning rate on DQN method and it is found that learning rate () at 0.01 is the best in this case rather than higher 0.1 or lower 0.001. **Figure 7** shows the effects of exploration parameters () on DQN and when exploration parameter is 1$$104, agent get higher reward than other two parameters.

From above two sensitivity analysis cases, the importance of hyper-parameters is verified, and this is a common problem in deep learning, many times need to try and error or experience to determine the optimal hyper-parameters.

**2.1.3 The convergence rate of the difference between mean epochs rewards**

**Figure 8** shows the scatter plot of the difference between the mean epochs benchmark reward and mean epochs reward of the DQN algorithm. In order to better show the convergence rate, this figure is drawn on a log-log scale. **Figure 8** (a) (b) (c) are for lifetime 2,3,4, respectively, and they indicate that the difference between benchmark mean reward and DQN mean reward begins to decrease rapidly after 200 runs, this demonstrates our deep reinforcement learning method works, and agent gradually learned how to order and price is optimal. In addition, the fitting lines in the figure is used to depict the convergence rate, and we get following fitting line functions, function (13) (14) (15) are for lifetime 2,3 and 4, respectively, it is found that with the lifetime goes on, it has a faster convergence rate, but the R-squared goes down.

(13)

(14)

(15)

Figure 6. Mean epochs reward for learning rates

Figure 7. Mean epochs reward for

Figure 8. log-log scale mean epochs reward for DQN

**3. conclusions**

In this paper, we investigate a joint pricing and inventory control problem obtaining a near-optimal pricing and replenishment policy for stochastic perishable inventory systems with positive lead time by deep reinforcement learning algorithm. And in order to solve the lag of order, we reset the composition of the state and add the state and order information in the early stage, this adds dimension but helps retailer choose better action. In addition, we proposed two reinforcement learning algorithms, Deep Q-learning (DQN) with experience replay and Advantage Actor-Critic (A2C), to study this problem and found both two algorithms can help retailer get better returns, and Deep Q-learning (DQN) with experience replay will earn more than Advantage Actor-Critic (A2C).

In this paper, we only focus on the single perishable product and reinforcement learning is used more and more widely, it will be interesting to study multi-product inventory control and channel coordination by deep reinforcement learning.

**References**