Branching Deep Q-Network Agent for Joint Replenishment Policy

Hiroshi Suetsugu[1,a)] Yoshiaki Narusue[1] Hiroyuki Morikawa[1]

Abstract: This study proposes a reinforcement learning approach to find the near-optimal dynamic ordering policy for a multi-product inventory system with non-stationary demands. The distinguishing feature of multi-product inventory systems is the need to take into account the coordination among products with the aim of total cost reduction. The Markov decision process formulation has been used to obtain an optimal policy. However, the curse of dimensionality has made it intractable for a large number of products. For more products, heuristic algorithms have been proposed on the assumption of a stationary demand in literature. In this study, we propose an extended Q-learning agent with function approximation, called the branching deep Q-network (DQN) with reward allocation based on the branching double DQN. Our numerical experiments show that the proposed agent learns the coordinated order policy without any knowledge of other products' decisions and outperforms non-coordinated forecast-based economic order policy.

1. Introduction

Own-brand goods play an important role for retailers in terms of their profits. At their own risk, many retail companies have tried to purchase their goods in the production country and deliver these goods directly to selling countries. In this case, the joint replenishment policy (JRP), which takes multi-product situations into account, is required to achieve the minimum total cost. In a supply chain where multiple products need to be delivered via a container ship, the maritime transportation cost depends on the required number of containers. Thus, the transportation cost per product would decrease when several products are ordered simultaneously. However, JRP is considered to be non-deterministic polynomial-time (NP)-hard because of its combinatorial nature. There has been much research on JRP. According to [2], JRP research can be divided into three categories depending on the demand assumptions: deterministic, dynamic or stochastic. In the real-world business setting, almost all the businesses fall into the stochastic demand setting. Some studies use the Markov decision process to solve the JRP with the stochastic demand. The problem is complex in the stochastic demand setting; therefore, a number of studies were conducted on the assumption of the stationary demand which follows a Poisson or normal distribution.

On the other hand, many researchers have started using reinforcement learning (RL) for the supply chain management setting because of recent advances in RL. A typical problem setting is the Beer Game ([5], [1]), where a multi-echelon supply chain problem is considered under uncertain future demand and supply. Although the existing literature showed positive results in multi-echelon supply chains, only one product has been considered so far. This is because of the problem of large discrete action spaces, which appear in the general RL approach. To take full advantage of RL in the supply chain management, these exponentially increasing action space problems with multiple products need to be resolved.

[6] proposed the branching DQN (BDQN), in which function approximated Q-values are represented with individual network branches followed by a shared decision module that encodes a latent representation of the input and helps with the coordination of the branches. This architecture enables the linear growth of the total number of network outputs with increasing action dimensionality.

In this paper, we propose the RL agent with function approximation to find the near-optimal dynamic multi-product ordering policy under non-stationary demand based on BDQN combined with credit assignment. The proposed agent is based on the idea of treating a single agent as a multi-agent learner with a credit assignment policy that is a hybrid of the global and the local rewards with the aim of coping with the combinatorial increase in the action space.

2. Method

2.1 Problem setting

We consider a multi-product inventory system between one supplier and one retailer. Our objective is to minimize

[1] Graduate School of Engineering
University of Tokyo, Tokyo, Japan
[a)] hsuetsugu@sglab.co.jp

the total retailer cost, which includes the holding, penalty, and transportation costs. We assumed a non-stationary demand and a demand forecast conditioned by the forecast error parameter, which means the agent knows the expected demand forecast accuracy as prior knowledge. We have used the following notations:

i : Item number, $i = 1, ..., N$,
t : Period, $t = 1, ..., T$,
LT_i : Lead time of item i from supplier to retailer,
l_i : Lot size of item i, (in palette),
$d_{i,t}$: Demand for item i during period t, (in palette),
$f_{i,t}$: Forecast of the demand for item i during period t,(in palette),
$x_{i,t}$: Order quantity for item i made at time t, (in palette),
$r_{i,t}$: Replenishment for item i from supplier during period t, (in palette),
$\hat{r}_{i,t}$: Replenishment forecast for item i from supplier during period t, (in palette),
$I_{i,t}$: Inventory position of item i at the start of time t, (in palette),
$\hat{I}_{i,t,\hat{t}}$: Inventory position forecast for item i at time $\hat{t}$ forecasted at time t (in palette),
$u_{i,t}$: Unsatisfied demand of item i during period t, (in palette),
$s_{i,t}$: Shipment of item i from retailer during period t, (in palette),
E_i : Forecast error parameter of item i.

The demand forecasts are generated so that the proportion of standard deviation of forecast error to the standard deviation of demand itself equals E_i. We permit lost sales. Replenishment at time t can be used from time $t+1$. In this study, we do not take supplier stock-out or any supply delay into consideration. Thus, the relationship between inventory, replenishment, shipment, demand, unsatisfied demand and inventory position forecast can be formulated as follows.

$$\hat{r}_{i,t+LT_i} = x_{i,t}, \tag{1}$$
$$r_{i,t} = \hat{r}_{i,t}, \tag{2}$$
$$s_{i,t} = min(d_{i,t}, I_{i,t}), \tag{3}$$
$$u_{i,t} = d_{i,t} - s_{i,t}, \tag{4}$$
$$I_{i,t+1} = I_{i,t} - s_{i,t} + r_{i,t}, \tag{5}$$
$$\hat{I}_{i,t,\hat{t}+1} = I_{i,t} - \sum_{t'=t}^{\hat{t}} f_{i,t'} + \sum_{t'=t}^{\hat{t}} \hat{r}_{i,t'}. \tag{6}$$

Cost is defined as follows:

$$C_{i,t}^{hold} = U^{hold} \times I_{i,t}, \tag{7}$$
$$C_{i,t}^{pel} = U^{pel} \times u_{i,t}, \tag{8}$$
$$C_t^{trans} = U^{trans} \times \lceil \frac{\sum_i x_{i,t}}{CAP} \rceil, \tag{9}$$

where CAP represents container capacity (in palette) and $\lceil \cdot \rceil$ is ceiling function. $C_{i,t}^{hold}$, $C_{i,t}^{pel}$, and C_t^{trans} represent the holding, penalty, and transportation costs of item i during period t respectively, and U^{hold}, U^{pel}, and U^{trans} are the unit holding, shortage, and transportation costs respectively.

2.2 MDP formulation for multi-product inventory system with demand forecasts

2.2.1 Observations and state variables

At every time step, the agent obtains information about the future inventory positions according to the demand forecast. The on-order quantity $OO_{i,t}$ of item i at time t (i.e. the items that have been ordered but have yet been received) can be defined by $\sum_t \hat{r}_{i,t}$. Let $[\cdot]^{st:T}$ be the summation of $[\cdot]$ from st to $st+T$. In each period, the agent has observations $o_t = [(I_{i,t}, OO_{i,t}, \hat{I}_{i,t,t+LT}, f_{i,t}^{t:LT}, f_{i,t}^{t+LT:M})]_{i=1}^N$ and makes a decision based on o_t. Here, M is the parameter which decides how far the future demand needs to be considered and we let M be four weeks.

True information on $I_{i,t+LT}$, $d_{i,t}^{t:LT}$, and $d_{i,t}^{t+LT:M}$ can be observed afterward by use of the actual demand. Thus, we can define the state by $s_t = [(I_{i,t}, OO_{i,t}, I_{i,t+LT}, d_{i,t}^{t:LT}, d_{i,t}^{t+LT:M})]_{i=1}^N$.

Let us assume that the average demand forecast error does not change from time to time and the agent knows its degree of forecast error as prior knowledge. Then, our *belief* state $b(s)$ should be conditioned only on o_t and can be defined by $P(s_t|o_t)$ instead of $P(s_t|h)$ in a general POMDP. Also assuming that expected forecast error follows normal distribution, we can infer state s_t from observation o_t by using E_i (see Section 3.4 for further explanation).

Note that our belief state $b(s)$ can be calculated by the use of only o_t and E_i. When selecting action greedily, we used the following for the greedy policy based on [4]:

$$a_t = \arg\max_a \mathbb{E}[Q(\hat{\boldsymbol{s}}_t, a)], \tag{10}$$

where $\hat{\boldsymbol{s}}_t$ are estimated states that use the Monte Carlo sampling. In our experiments, we generated 300 samples at each step.

2.2.2 Action space

In each period, an agent orders $x_{i,t} \in X_i$, which can be any multiples of lot sizes l_i. However, infinite action space is not practical and also taking a large number of orders compared with the demand is unrealistic from a supply chain point of view. Therefore, we limited action space X_i to $X_i = \{l_i a \mid a \in \{0, 1, 2, 3\}\}$.

2.3 Branching deep Q-network with reward allocation

In [6], they examined the action branching agent for environment in which only a global reward was available. In our case, each branch consisted of one product. Our objective variable was total cost; the transportation cost was calculated across multiple products whereas the holding cost and penalty cost were calculated independently of each product,

which means that the total cost included both global and local rewards.

After several numerical experiments, our best result came from the architecture shown in Fig. 1 which had the distinguishing feature of allocating rewards to each branch.

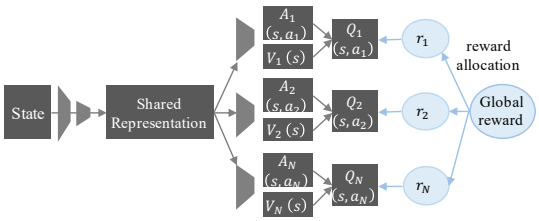

Fig. 1 illustration of our proposed agent

2.3.1 Reward allocation

Unlike [6], we modified our temporal difference target equation as follows:

$$y_d = r_d + \gamma Q_d^- \left(s', \arg\max_{a'_d \in \mathcal{A}_d} Q_d \left(s', a'_d \right) \right), \qquad (11)$$

where r_d refers to the reward per product. The transportation costs depend on the number of containers; the remaining costs can be calculated separately for each product. There are several options for allocating the total transportation cost to each product. The best results come from the following allocation by which the total transportation cost is allocated equally to all the products even if a specific product is not ordered:

$$r_d = -(C_{i,t}^{hold} + C_{i,t}^{pel} + \frac{C_t^{trans}}{N}). \qquad (12)$$

Intuitively, this allocation method would encourage each branch to put an order simultaneously.

Thus, the loss function should be defined for each branch, and all the branches backpropagation gradients are rescaled by $1/N$ for the shared part of our architecture.

$$L_d = \mathbb{E}_{(s,a_d,r_d,s') \sim \mathcal{D}} \left[L_\delta \left(y_d, Q_d \left(s, a_d \right) \right) \right], \qquad (13)$$

where L_δ is the Huber loss function.

2.3.2 State-value estimator

BDQN has a common state-value estimator. It is natural to set the branch-independent state-value estimator as an adaptation of dueling network into our proposed agent with reward allocation. Thus, the branch independent state value and advantage can be simply defined as follows:

$$Q_d \left(s, a_d \right) = V_d(s) + A_d(s, a_d), \qquad (14)$$

where $V_d(s)$ denotes the branch independent state-value and $A_d \left(s, a_d \right)$ denotes the corresponding sub-action advantage.

3. Experiments

3.1 Experimental settings

We conducted numerical experiments to examine the following questions:

1) Can the proposed agent learn the coordinated order policy across multiple products?
2) Can the proposed agent converge with a large number of products?
3) Can the proposed agent converge with a non-stationary demand?
4) Can the proposed agent find an optimal policy as compared with the benchmark policy?

For validation, we used the standard Dueling Double DQN, which we simply call DQN in this paper, and a forecast-based economic order policy (F-EOP) as the benchmark approach. Each episode consisted of 200 time steps, and initial 20 steps were ignored from the evaluation so as to exclude the effect of initial inventory setting.

Regarding the branching architecture, we tried three types of agents: BDQNs (BDQN with state-value estimator), BDQN-RA (BDQN with reward allocation), and BDQN-RAs (BDQN with reward allocation and state-value estimator).

In order to validate above-mentioned questions, we conducted three experiments by varying the number of products and the demand stationarity.

3.2 Benchmark methodology: Forecast-based economic order policy

There has been no established JRP under non-stationary demand and demand forecast; therefore, we selected a non-coordinated order policy based on [3] as our benchmark policy, which consisted of forecast-based order-point and an economic replenishment quantity based on Wagner-Whitin dynamic lot size model with extension to incorporate demand forecasts.

3.2.1 Forecast-based order-point

When demand forecasts are available, a replenishment order takes place when the forecasted inventory position at time $t + LT$ drops to order-point or lower. Assuming that forecast error follows normal distribution, order-point can be defined as $s = k \times \sigma\sqrt{LT}$ where k is the safety factor and σ is the standard deviation of forecast error. k can be determined so that sum of the expected penalty cost and expected holding cost for the safety stock are minimized.

3.2.2 Economic replenishment quantity with demand forecasts

At each time step, we choose the order quantity $x_{i,t} \in X_i$. When $x \in X_i$ is selected at time t, the expected unit time cost $C(x)$ from $t + LT$ to the next replenishment timing is calculated by dividing the sum of the expected holding cost and the transportation cost by T where T is determined by estimating the timing for which the forecasted inventory position drops to or lower than the order-point on condition that x is replenished at time $t + LT$. Thus, the economic replenishment quantity with the demand forecasts can be derived by $argmin_x(C(x))$.

3.3 Experiment result

Table.1 summarizes the experiments' results. Our pro-

Table 1 Summary of experiment settings and results

ID	# of Products	Demand Stationarity	F-EOP	DQN	BDQNs	BDQN-RA	BDQN-RAs	Improved(%)
1	2	Stationary	133.2	105.3	108.0	(*) 106.7	107.3	19.9%
2	2	Upward Trend	216.5	205.2	266.3	(*) 192.0	195.2	11.3%
3	10	Upward Trend	447.8	-	817.2	346.8	(*) 341.3	23.8%

The final results for DQN and BDQN-family were derived by calculating the averaged total cost with a greedy policy using the trained model after 10,000 episodes over 6 runs. Improved(%) represents the decrease in total cost compared with F-EOP. (*) denotes the item used for calculating Improved(%).

posed agent performed better than did the non-coordinated F-EOP policy. As the number of products increased, DQN and BDQN (without reward allocation) suffered from convergence whereas our proposed agent, BDQN-RA(s), performed well even with 10 products. DQN suffered from its combinatorial increasing action space for a large number of products. When the number of products equaled 10, its action space was around 10^6. As for BDQN, using a single global reward made the learning convergence unstable because the feedback signal to each branch was considered to be too noisy. For the proposed agent, the reward allocation strategy worked well in stable learning while achieving coordinated orders. The state-value estimator in the proposed agent did not show a significant impact on the results.

One of the most important validation items regarding the action branching agent in JRP is whether or not the coordinated order is possible across multiple products. As the learning process proceeded, our proposed agent learned to order these two products simultaneously such that the transportation cost was minimized. Even if the inventory position of one item was relatively high (i.e., immediate order did not need to take place), the order took place in accordance with the other order.

In experiment 3, we extended our experiments to a more complicated setting with 10 products, and an average unit time demand was still much less than the container capacity. BDQN failed to converge, whereas our proposed agent exhibited an efficient and stable learning process. BDQN-RAs agent achieved a 23.8% cost reduction as compared with F-EOP.

3.4 Experiment detail

We let cost parameters U^{hold}, U^{pel}, and U^{trans} be 0.02, 1.0, and 1, respectively. The demand and lot size of each product are provided in Table.2. Stationary demand was generated following $N(\mu, \sigma)$ and we let $\frac{\sigma}{\mu}$ be 0.4. Non-stationary data were generated by the simple addition of the linear upward trends until the demand at the end of the 200 time steps tripled, defined by; $d_{i,t} = \hat{d}_{i,t} + 2 * \mu_i * (t/200)$ where $\hat{d}_{i,t}$ denotes stationary demand. Demand forecasts were generated so that the proportion of the standard deviation of the forecast error to the standard deviation of the demand itself equaled 0.5.

Table 2 Demand setting in each experiment

ID	μ	Lot size
1	[2, 2]	[8, 8]
2	[2, 2]	[8, 8]
3	[.3, .4, .5, .5, .7, .9, 1., 1., 1.2, 1.2]	[1, 1, 1, 1, 2, 2, 3, 3, 3, 3]

4. Conclusion

We introduced extended branching Q-learning agent with function approximation designed for combinatorial action dimension with global and local reward based on the cost structure of multi-product inventory system. Our numerical experiments showed that as the number of products increased, both DQN and BDQN suffered from convergence; however, our proposed agent performed better when compared with F-EOP. Instability in the latter part of the learning process caused by the branch-independent action selection using our proposed agent should be investigated in future studies.

Our proposed agent needed only the demand forecast, which is usual in the real business setting; this expands the possibility to adapt our approach in real-world situations. In future studies, we also need to investigate how to extend this by including multi-layer and multi-retailer supply chains with realistic constraints.

References

[1] Chaharsooghi, K. S., Heydari, J. and Zegordi, H. S.: A reinforcement learning model for supply chain ordering management: An application to the beer game, Vol. 45, No. 4, pp. 949–959 (2008).

[2] dos Bastos, L., Mendes, M., de Nunes, D., Melo, A., Carneiro, M., do de Janeiro, B., Vargas, B. and do do Pará, B.: A systematic literature review on the joint replenishment problem solutions: 2006-2015, *Prod*, Vol. 27, No. 0 (online), DOI: 10.1590/0103-6513.222916 (2017).

[3] Ishigaki, A. and Hirakawa, Y.: Design of a Economic Order-Point System based on Forecasted Inventory Positions, *Journal of Japan Industrial Management Association*, Vol. 59, No. 4, pp. 290–295 (2008).

[4] Littman, M. L., Cassandra, A. R. and Kaelbling, L.: Learning policies for partially observable environments: Scaling up, pp. 362–370 (1995).

[5] Oroojlooyjadid, A., Nazari, M., Snyder, L. and Takáč, M.: A Deep Q-Network for the Beer Game: A Reinforcement Learning algorithm to Solve Inventory Optimization Problems (2017).

[6] Tavakoli, A., Pardo, F. and Kormushev, P.: Action Branching Architectures for Deep Reinforcement Learning, *CoRR*, Vol. abs/1711.08946 (online), available from ⟨http://arxiv.org/abs/1711.08946⟩ (2017).