BOOK: Storing Algorithm-Invariant Episodes for Deep Reinforcement Learning

Simyung Chang[1,2], YoungJoon Yoo[3], Jaeseok Choi[1] and Nojun Kwak[1]
[1] *Seoul National University, Seoul, Korea*
[2] *Samsung Electronics, Suwon, Korea*
[3] *Clova AI Research, NAVER Corp, Seongnam, Korea*

Keywords: BOOK, Book Learning, Reinforcement Learning.

Abstract: We introduce a novel method to train agents of reinforcement learning (RL) by sharing knowledge in a way similar to the concept of using a book. The recorded information in the form of a book is the main means by which humans learn knowledge. Nevertheless, the conventional deep RL methods have mainly focused either on *experiential learning* where the agent learns through interactions with the environment from the start or on *imitation learning* that tries to mimic the teacher. Contrary to these, our proposed *book learning* shares key information among different agents in a book-like manner by delving into the following two characteristic features: (1) By defining the linguistic function, input states can be clustered semantically into a relatively small number of core clusters, which are forwarded to other RL agents in a prescribed manner. (2) By defining state priorities and the contents for recording, core experiences can be selected and stored in a small container. We call this container as 'BOOK'. Our method learns hundreds to thousand times faster than the conventional methods by learning only a handful of core cluster information, which shows that deep RL agents can effectively learn through the shared knowledge from other agents.

1 INTRODUCTION

Recently, reinforcement learning (RL) using deep neural networks (Mnih et al., 2013; Van Hasselt et al., 2016; Mnih et al., 2016) has achieved massive success in control systems consisting of complex input states and actions, and applied to various research fields (Silver et al., 2016; Abbeel et al., 2007). The RL problem is not easy to directly solve via cost minimization problem because of the constraint that it is difficult to immediately obtain the output according to the input. Therefore, various methods such as Q-learning (Bellman, 1957) and policy gradient (Sutton et al., 1999) have been proposed to solve the RL problems.

The recent neural-network (NN)-based RL methods (Mnih et al., 2013; Van Hasselt et al., 2016; Mnih et al., 2016) approximate the dynamic-programming-based (DP-based) optimal reinforcement learning (Jaakkola et al., 1994) through the neural network. However, this process has the problem that the Q-values for independent state-action pairs are correlated, which violates the independence assumption. Thus, this process is no longer optimal (Werbos, 1992). This results in differences in performance and convergence time depending on the experiences used to train the network. Hence, the effective selection of the experiences becomes crucial for successful training of the deep RL framework.

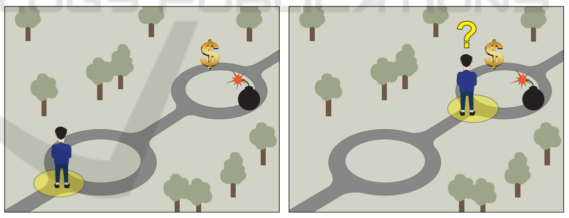


Figure 1: Example illustration of the semantically important state. In the left image, the person (agent) standing on the yellow circle (state) can choose either ways, and the results for two actions would be the same. Conversely, in the right image, the result will be largely different (bomb or money) according to the action (direction) the agent choose on the turning point (yellow). In our work, the state in the right image is considered to be more important than the state in the left image and this state is stored for further usage in the learning of other agents.

To gather the experiences, most deep-learning-based RL algorithms have utilized experience memory in the learning process (Lin, 1992; Mnih et al., 2013; Van Hasselt et al., 2016; Mnih et al., 2016; Schulman et al., 2015; Riedmiller, 2005), which stores batches of state-action pairs (experiences) that frequently appear in the network for repetitive use in the future learning process. Also, in (Schaul et al., 2015), a method of prioritized experience memory that finds priorities of each experience is proposed, based on which a batch is created. Eventually, the key to creating such a memory is to compute the priorities of the credible experiences so that learning can focus on the reliable experiences.

However, in the existing methods, just a few episodes have meaningful information, and the usability of the gathered episodes are highly algorithm-specific. It can be largely inefficient compared to humans who can select semantically meaningful (credible) events for learning the proper behaviors, regardless of the training method. Figure 1 shows the examples of the case. In the situation shown in the left image, choosing an action does not bring much difference to the agent. However, in the case of the right image, the result (bomb or money) of choosing an action (left or right) can be significantly different for the agent, and it is natural to think that the later case is much important for deciding the movement of the agent.

Inspired by the observation, in this paper, we propose a method of extracting and storing important episodes that are invariant to diverse RL algorithms. First, we propose an importance and a priority measures that can capture the semantically important episodes during entire experiences. More specifically, in this paper, the importance of a state is measured by the difference of the rewards resultant from different actions and the priority of a state is defined as the product of importance and the frequency of the state in episodes.

Then, we gather experiences during an arbitrary deep RL learning procedure, and store them into dictionary-type memory called ‘BOOK’ (Brief Organization of Obtained Knowledge). The process of generating a BOOK during the learning of a writer agent will be termed as ’Writing the BOOK’ in the followings. The stored episodes are quantized with respect to the state, and the quantized states are used as a key in the book memory. All the experiences in the BOOK are dynamically updated by upcoming experiences having the same key. To efficiently manage the episode in the BOOK, some linguistics inspired terms such as linguistic function and state are proposed.

We have shown that the ’BOOK’ memory is particularly effective for two aspects. First, we can use the memory as a good initialization data for diverse RL training algorithms, which enables fast convergence. Second, we can achieve compatible, and sometimes higher performances by only using the experiences in the memory when training a RL network, compared to the case that entire experiences are used. The experiences stored in the memory is usually a few hundred times smaller compared to the experiences required in usual random-batch-based RL training (Mnih et al., 2013; Van Hasselt et al., 2016), and hence give us much effectiveness in time and memory space required for the training.

The contributions of the proposed method are as follows:

(1) The dictionary termed as BOOK that stores the credible experience, which is useful for diverse RL network training algorithms, expressed by the tuple (cluster of states, action, and the corresponding Q-value) is proposed.

(2) The method for measuring the credibility: importance and priority terms of each experience valid for arbitrary RL training algorithms, is proposed.

(3) The training method for RL that utilizes the BOOK is proposed, which is inspired by DP and is applicable to diverse RL algorithms.

To show the efficiency of the proposed method, it is applied to the major deep RL methods such DQN (Mnih et al., 2013) and A3C (Mnih et al., 2016). The qualitative as well as the quantitative performances of the proposed method are validated through the experiments on public environments published by OpenAI (Brockman et al., 2016).

2 BACKGROUND

The goal of RL is to estimate the sequential actions of an agent that maximize cumulative rewards given a particular environment. In RL, Markov decision process (MDP) is used to model the motion of an agent in the environment. It is defined by the *state* $s_t \in \mathbb{R}^S$, *action* $a_t \in \{a_1, \ldots, a_A\}$ which occurs in the state s_t, and the corresponding *reward* $r_t \in \mathbb{R}$, at a time step $t \in \mathbb{Z}^+$. [1] We term the function that maps the action a_t for a given s_t as the *policy*, and the future state s_{t+1} is defined by the pair of the current state and the action, (s_t, a_t). Then, the overall cost for the entire sequence from the MDP is defined as the accumulated discounted reward, $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, with a discount factor $\gamma \leq 1$.

Therefore, we can solve the RL problem by finding the optimal policy that maximizes the cost R_t.

[1] $\mathbb{R}$ and $\mathbb{Z}^+$ denote the real and natural numbers respectively.

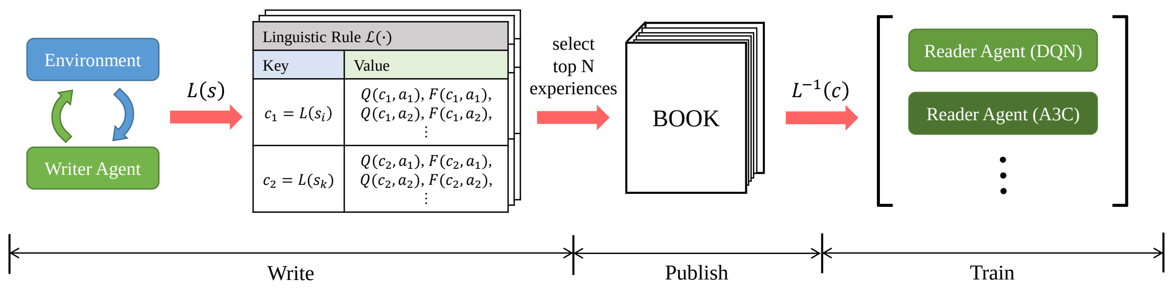


Figure 2: Overall framework of the proposed model. Similar experiences (state-action pairs) from multiple episodes are grouped into a cluster and the credible experiences corresponding to large clusters are written in a BOOK with their Q-values and frequencies Fs. The BOOK is published with Top N experiences after learning. Then, reader agents use this information in training.

However, it is difficult to apply the conventional optimization methods in finding the optimal policy. It is because we should wait until the agent reaches the terminal state to see the cost R_t resulting from the action of the agent at time t. To solve the problem in a recursive manner, we define the function $Q(s_t,a_t) = \mathbb{E}[R_t|s = s_t, a = a_t, \pi]$ denoting the expected accumulated reward for (s_t,a_t) with a policy π. Then, we can induce the recurrent Bellman equation (Bellman, 1957):

$$Q(s_t,a_t) = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}). \qquad (1)$$

It is proven that the Q-value, $Q(s_t,a_t)$, for all time step t satisfying (1) can be calculated by applying dynamic programming (DP), and the resultant Q-values are optimal (Jaakkola et al., 1994). However, it is practically impossible to apply the DP method when the number of state is large, or the state is continuous. Recently, the methods such as Deep Q-learning (DQN), Double Deep-Q-learning (DDQN) solve the RL problem with complex state s_t by using approximate DP that trains Q-network. The Q-network is designed so that it calculates the Q-value for each action when a state is given. Then, the Q-network is trained by the temporal difference (TD) (Watkins and Dayan, 1992) method which reduces the gap between Q-values acquired from the Q-network and those from (1).

3 RELATED WORK

Recently, deep learning methods (Mnih et al., 2013; Hasselt, 2010; Van Hasselt et al., 2016; Wang et al., 2015; Mnih et al., 2016; Schaul et al., 2015; Salimans et al., 2017) have improved performance by incorporating neural networks to the classical RL methods such as Q-learning (Watkins and Dayan, 1992), SARSA (Rummery and Niranjan, 1994), evolution learning (Salimans et al., 2017), and policy searching methods (Williams, 1987; Peters et al., 2003) which use TD (Sutton, 1988).

(Mnih et al., 2013), (Hasselt, 2010) and (Van Hasselt et al., 2016) replaced the value function of Q-learning with a neural network by using a TD method. (Wang et al., 2015) proposed an algorithm that shows faster convergence than the method based on Q-learning by applying dueling network method (Harmon et al., 1995). Furthermore, (Mnih et al., 2016) applied the asynchronous method to Q-learning, SARSA, and Advantage Actor-Critic models.

The convergence and performance of deep-learning-based methods are greatly affected by input data which are used to train an approximated solution (Bertsekas and Tsitsiklis, 1995) of classical RL methods. (Mnih et al., 2013) and (Van Hasselt et al., 2016) solved the problem by saving experience as batch in the form of *experience replay memory* (Lin, 1993). In addition, Prioritized Experience Replay (Schaul et al., 2015) achieved higher performance by applying replay memory to recent Q-learning based algorithms by calculating priority based on the importance of experience. (Pritzel et al., 2017) proposed a Neural episodic control (NEC) to apply tabular based Q-learning method for training the Q-network by first, semantically clustering the states and then, updates the value entities of the clusters.

Also, imitation learning (Ross and Bagnell, 2014; Krishnamurthy et al., 2015; Chang et al., 2015) which solves problems through expert's experience is one of the main research flows. This method trains a new agent in a supervised manner using state-action pairs obtained from the expert agent and shows faster convergence speed and better performance using experiences of the expert. However, it is costly to gather experiences from experts.

The goal of our work is different to the mentioned approaches as follows. (1) compared to imitation learning, the proposed method differs in the aspect that credible data are extracted from the past data in an unsupervised manner, and more importantly, (2) compared to the prioritized experience replay (Schaul et al., 2015), our work proposes a method to generate a memory that stores core experiences useful for training diverse RL algorithms. (3) Also, compared to the NEC (Pritzel et al., 2017), our work aims to use the BOOK memory for good initialization and fast convergence, when training the RL network regardless of the algorithm used. but, the dictionary of NEC can not provide all the information necessary for learning, such as states, so it is difficult to use it to train other RL networks.

4 PROPOSED METHOD

In this paper, our algorithm aims to find the core experience through many experiences and write it into a BOOK, which can be used to share knowledge with other agents that possibly use different RL algorithms. Figure 2 describes the main flow of the proposed algorithm. First, from the RL network, the terminated episodes of a writer agent are extracted. Then, among experiences from the episodes, the core and credible experiences are gathered and stored into the BOOK memory. In this process, using the semantic cluster of states as a key, the BOOK stores the value information of the experiences related to the semantic cluster. This 'writing' process is iterated until the end of training. Then, the final BOOK is 'published' with the top N core experiences of this memory, that can be directly exploited in the 'training' of other reader RL agents.

In the following subsections, how to design the BOOK and how to use BOOK in the training of RL algorithms are described in more detail.

4.1 Desigining the BOOK Structrue

Given a state $s \in \mathbb{R}^S$ and action $a \in \{a_1, ..., a_A\}$, we define the memory $\mathcal{B}$ termed as 'BOOK' which stores the credible experience in the form appropriate for lookup-table inspired RL. Assuming there exists semantic correlation among states, the input state $s_i, i = 1 \ldots N_s$ can be clustered into the core K clusters $\mathcal{C}_k \in \mathcal{C}, k = 1, \ldots, K$. To reduce the semantic redundancy, the BOOK stores the information related to the cluster $\mathcal{C}_k$, and the corresponding information is updated by the information of the states s_i included in the cluster. It means that the memory space of the BOOK in the 'writing' process is $O(AK)$. To map the state s_i to the cluster $\mathcal{C}_k$, we define the mapping function $L : s \rightarrow c_k$, where $c_k \in \mathbb{R}^S$ denotes the representative value of the cluster $\mathcal{C}_k$. We term the mapping

Algorithm 1: Writing a BOOK.

```
Define linguistic function $L$ for states and reward and initialize it.
Initialize BOOK $\mathcal{B}$ with capacity $K$.
for episode $= 1, \ldots, M$ do
    Initialize Episode memory $\mathcal{E}$
    Get initial state s
    for $t = t_{start}, \ldots, t_{terminal}$ do
        Take action $a_t$ with policy $\pi$
        Receive new state $s_{t+1}$ and reward $r_t$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{E}$
        Perform general reinforcement algorithm
    end for
    for $t = t_{terminal}, \ldots, t_{start}$ do
        Take transition $(s_t, a_t, r_t, s_{t+1})$ from $\mathcal{E}$
        $c_k = L(s_t)$
        Update $Q(c_k, a_t)$, $F(c_k, a_t)$ to $\mathcal{B}$ with equation (2), (3), (4), (5)
    end for
    if $episode\%T_{decayPeriod} == 0$ then
        Decay $F(c_k, a_j)$ in $\mathcal{B}$ for all $k \in \{1, \ldots, K\}$ and $j \in \{1, \ldots, A\}$
    end if
end for
```

function $L(\cdot)$, the representative c_k, and the reward of c_k as *linguistic function*, *linguistic state*, and *linguistic reward*, respectively [2]. To cluster the states and define the *linguistic function*, arbitrary clustering methods or quantization can be applied. For simplicity, we adopt the quantization in this paper.

Consequently, the element of a BOOK $b_{k,j} \in \mathcal{B}$ is defined as $b_{k,j} \in \{c_k, Q(c_k, a_j), F(c_k, a_j)\}$, where $Q(c_k, a_j)$ and $F(c_k, a_j)$ denote the Q-value of (c_k, a_j) and the hit frequency of the $b_{k,j}$. Then, the information regarding the input state s_i is stored into $b_k = [b_{k,1}, \ldots, b_{k,A}]$, where $c_k = L(s_i)$. The Q-value $Q(c_k, a_j)$ is iteratively updated by $Q^t(s_t = s_i, a_t = a_j)$ which denotes the Q-value from the credible experience $\{s_t, a_t, r_t, s_{t+1}\}$.

4.2 Iterative Update of the BOOK using Credible Experiences

To fill the BOOK memory by credible experiences, we first extract the credible experiences from the entire possible experiences. We extract the credible experiences based on the observation that the terminated episode[3] holds valid information to judge whether an

[2]The term 'linguistic' is used to represent both characteristics of 'abstraction' and 'shared rule'.

[3]An episode denotes a sequence of state-action-reward until termination.

agent's action was good or bad. At least in the terminal state, we can evaluate whether the state-action pair performed good or bad by just observing the result of the final action; for example, success or failure. Once we get credible experience from terminal sequences, then we can get the related credible experiences using the upcoming equation (2). More specifically, the BOOK is updated using the experience E_t from the terminated episode $\mathcal{E} = \{E_1, ..., E_T\}$ in backward order, i.e., from E_T to E_1, where $E_t = \{s_t, a_t, r_t, s_{t+1}\}$. Consider that for an experience E_t at time t, the current state, current action, and the future state are $s_t = s_i, a_t = a_j$ and $s_{t+1} = s_{i'}$, respectively. Also, assume that $s_i \in \mathcal{C}_k$. Then, the Q-value $Q(c_k, a_j)$ stored in the content $b_{k,j}$ is updated by

$$Q(c_k, a_j) = \beta Q(c_k, a_j) + (1-\beta) Q^t(s_i, a_j), \qquad (2)$$

where

$$Q^t(s_i, a_j) = r_t + \gamma \max_{a'} Q(s_{i'}, a'). \qquad (3)$$

$$\beta = F(c_k, a_j) / \{F(c_k, a_j) + F(L(s_{i'}), \arg\max_{a'} Q(s_{i'}, a'))\} \qquad (4)$$

Here, $F(c_k, a_j)$ refers to the hit frequency of the content $b_{k,j}$. The term $Q^t(s_i, a_j)$ denotes the estimated Q-value of (s_i, a_j) acquired from the RL network. In (4), $F(L(s_{i'}), \arg\max_{a'} Q(s_{i'}, a'))$ is initialized to 1 when the term regarding $L(s_{i'})$ is not yet stored in the BOOK. We note that we calculate $Q^t(s_t, a_t)$ from $Q^t(s_{t+1}, a_{t+1})$ in backward manner, because only the terminal experience E_T is fully credible among the episode $\mathcal{E}$ acquired from the RL network. The update rule for the frequency term $F(c_k, a_j)$ in the content $b_{k,j}$ is defined as

$$F(c_k, a_j) = \min(F(c_k, a_j) + F(L(s_{i'}), \arg\max_{a'} Q(s_{i'}, a'), F_l), \qquad (5)$$

where F_l is the predefined limit of the frequency $F(\cdot,\cdot)$. The frequency $F(\cdot,\cdot)$ is reduced by 1 for every predefined number of episodes to avoid $F(\cdot,\cdot)$ from being continually increasing. To extract the episode $\mathcal{E}$, we can use arbitrary deep RL algorithm based on Q-network. Algorithm 1 summarizes the procedure of writing a BOOK.

4.3 Priority based Contents Recoding

In many cases, the number of clusters becomes large, and it is clearly inefficient to store all the contents without considering the priority of a cluster. Hence, we maintain the efficiency of BOOK by continuously removing contents with lower priority from the BOOK. In our method, the priority $p_{k,j}$ is defined by the product of the frequency term $F(c_k, a_j)$ and the importance term $I(c_k)$,

$$p_{k,j} = I(c_k) F(c_k, a_j). \qquad (6)$$

The importance term $I(c_k)$ reflects the maximum gap of reward for choosing an action for a given linguistic state c_k, as the following:

$$I(c_k) = \max_a Q(c_k, a) - \min_a Q(c_k, a). \qquad (7)$$

In Fig. 1, we can see the concept of the importance term. At the first crossroad (state) in the left, the penalty of choosing different branches (actions) is not severe. However, at the second crossroad, it is very important to choose a proper action given the state. Obviously, the situation in the right image is much crucial, and the RL should train the situation more carefully. Now, we can keep the size of the BOOK as we want by eliminating the contents with lower priority $p_{k,j}$ (left image in the figure).

4.4 Publishing a BOOK

We have seen how to write a BOOK in the previous subsections. In the 'writing' stage in Fig. 2, it limits the contents to be kept according to priority, but maintains a considerable capacity K to compare information of various states. However, our method finally publish the BOOK with only the top $N (< K)$ priority states with the same rule as the subsection 4.3 after learning of the writer agent. We have shown through experiments that we can obtain good performance even if a relatively small-sized BOOK is used for training. See section 4.5 for more detailed analysis.

4.5 Training Reader Network using the BOOK

As shown in Figure 2, we train the RL network using the BOOK structure that stores the experience from the episode. The BOOK records the information of the representative states that is useful for RL training. The information required to learn the general reinforcement learning algorithm can be obtained in the form of $(s, a, Q(s,a))$ or $(s, a, V(s), A(s,a))$ through our recorded data. Here, $V(s)$ and $A(s,a)$ are the value of the state s and the advantage of the state-action pair (s,a).

To utilize the BOOK in the learning of the environment, the linguistic state c_k has to be converted to the real state s. The state s can be decoded by implementing the inverse function $s = L^{-1}(c_k)$, or one of the state $s \in \mathcal{C}_k$ can be stored in the BOOK as a sample when the BOOK is made.

In the first case of using Q-value $Q(s,a)$ in the training, the recorded information can be used as it is.

In the second case, $V(s)$ is calculated as the weighted sum of the $Q(s,a)$ and the difference between the Q-value and the state value V is used as the advantage $A(s,a)$ as follows:

$$V(s) \approx \frac{\sum_{a_i} F(c_k,a_i)Q(c_k,a_i)}{\sum_{a_i} F(c_k,a_i)}, \tag{8}$$

$$A(s,a) \approx Q(c_k,a) - V(s). \tag{9}$$

A BOOK stores only the measured (experienced) data regardless of the RL model without bootstrapping. The learning method of each model is used as it is, in the training using the BOOK. Since DQN (Mnih et al., 2013) requires state, action and Q-value in learning, it learns by decoding this information in the BOOK. On the other hand, A3C (Mnih et al., 2016) and Dueling DQN (Wang et al., 2015) require state, action, state-value V and advantage A, so these decode the corresponding information in the BOOK as shown in equations (8) and (9). Because a BOOK has all the information needed to train an RL agent, the agent is not required to interact with the environment while learning the BOOK.

We note that our learning process shares the essential philosophy with the classical DP in that the learning process explores the state-action space based on credible $Q(s,a)$ stored in the BOOK without bootstrapping and dynamically updates the values in the solution space using the stored information. As verified by the experiments, we confirmed that our methods achieved better performance with much smaller iteration compared to the existing approximated DP-based RL algorithms (Mnih et al., 2013; Mnih et al., 2016).

5 EXPERIMENTS

To show the effectiveness of the proposed concept of BOOK, we tested our algorithm on 4 problems from 3 domains. These are carpole (Barto et al., 1983), acrobot (Geramifard et al., 2015), Box2D (Catto, 2011) lunar lander, and Q*bert from Atari 2600 games. All the experiments were performed using OpenAI gym (Brockman et al., 2016).

The purpose of the experiments is to answer the following questions: (1) Can we effectively represent valuable information for RL among the entire state-action space and find important states? If so, can this information be effectively transfered to train other RL agent? (2) Can the information generated in this way be utilized to train the network in different architecture? For example, can a BOOK generated by DQN be effectively used to train A3C network?

5.1 Performance Analysis

In these experiments, we first trained the conventional network of A3C or DQN. During the training of the conventional writer network, a BOOK is written. Then, we tested the effectiveness of this BOOK with two different scenarios. First, we trained the RL networks using only the contents of the BOOK as described in Section 4.5. For the second scenario, we conducted additional training for the RL networks that are already trained using the BOOK at first scenario.

5.1.1 Performance of BOOK based Learning

Table 1 shows the performance when training the conventional RL algorithm with only the contents of the BOOK. The BOOK is written in the training of writer network with DQN and A3C and published in size of 1,000. Then, reader networks were trained with this BOOK using several different algorithms such as DQN, A3C, and Dueling DQN. This normally took much less time (less than 1 minute in all experiments) than the training of the conventional network from the start without utilizing BOOK. Then, we tested the performance of 100 random episodes without updating the network. The column ‘Score’ in the table shows the average score of this setting. The ‘Transition’ indicates the number of transitions (timesteps) that each network has to go through to achieve the same score without BOOK. The ’Ratio’ means the ratio of the book size over transition to confirm the sample efficiency of our method. For example, if Dueling DQN learns the BOOK of size 1,000 from A3C in Q*bert, it can get the score of 388.1. If this network learns without BOOK, it has to go through 1,080K transitions. The ratio is 0.09%, which is 1,000 / 1,080K.

As shown in Table 1, even if RL agents only learn the small-sized BOOK, they can obtain scores similar to those of scores obtained when learning dozen to thousands of times more transitions. In the Cartpole environment, particularly, all models obtained the highest score of 500, except when DQN learn the BOOK written by DQN.

However, the obtained scores are quite different depending on the model that wrote the BOOK and the model that learned the BOOK. In most environments and training models, learning the BOOK written by A3C is better than learning the BOOK written by DQN. Also, even if the same BOOK is used, the performances are different according to the training algorithm. DQN has lower performance than A3C or Dueling DQN in most environments. The major difference in each method is that DQN uses only Q

Table 1: Performance of BOOK based learning. **Score**: An average score that can be obtained by learning the BOOK of size 1,000. **Transition**: the number of timesteps that is needed for each reader model to get the same 'Score' without learning a BOOK. **Ratio**: a ratio of the size of a BOOK over Transition, Ratio = the size of BOOK / Transition.

Model		Cartpole			Acrobot			Lunar Lander			Q*bert		
Writer	Reader	Score	Transition	Ratio	Score	Transition	Ratio	Score	Transition	Ratio	Score	Transition	Ratio
DQN	DQN	428.3	352K	0.28%	**-280.7**	13K	7.7%	-251.7	9.7K	10.3%	196.9	566K	0.18%
	A3C	500.0	324K	0.31%	-281.6	162K	0.62%	-178.2	337K	0.30%	302.5	182K	0.55%
	Dueling	500.0	624K	0.16%	-370.2	10K	10.0%	**-127.4**	12K	8.3%	**324.6**	931K	0.11%
A3C	DQN	500.0	792K	0.13%	-172.1	49K	2.0%	-241.4	9.9K	10.1%	290.0	880K	0.11%
	A3C	500.0	324K	0.31%	**-91.8**	372K	0.27%	**-144.9**	520K	0.19%	**436.0**	383K	0.26%
	Dueling	500.0	624K	0.16%	-177.9	32K	3.1%	-160.5	10K	10.0%	388.1	1,080K	0.09%

value, and A3C and Dueling DQN use state-value and advantage. Dueling DQN got good scores in most environments, but in the case of Acrobot, using the BOOK by DQN, it was lower than all other models. This indicates that the information stored in the BOOK can be more or less useful depending on the reader RL method.

5.1.2 Performance of Additional Training after Learning the BOOK

The graphs in Figure 3 show the performance when the BOOK is used for pre-training the conventional RL networks. After learning the BOOK, each network is trained by each network-specific method. For this study, we conducted the experiments with two different settings: (1) training the RL network using the BOOK generated by the same learning method, (2) training the RL network using the BOOK generated by the different learning method. For the first setting, we trained the network and BOOK using A3C (Mnih et al., 2016), while in the second, we generated the BOOK using DQN (Mnih et al., 2013) and trained the network with A3C (Mnih et al., 2016).

The results of these two different settings are the upper and the lower rows of Figure 3, respectively. In the upper row, the 'blue' line shows the score achieved through training an A3C network from scratch, the 'yellow' horizontal line shows the base score which can be achieved through training other A3C network only with a BOOK which is published by a trained A3C network. The 'red' line shows the additional training results after training the A3C network with BOOK. In the lower row, the three lines mean the same with the upper row except that the BOOK is published by a different RL network, DQN.

As shown in Figure 3, the scores achieved from pre-trained networks using a BOOK were almost the same as the highest scores achieved from conventional methods. Furthermore, additional training on the pre-trained networks was quite effective since they achieved higher scores than conventional methods as training progresses. Especially, BOOK was very powerful when it is applied to a simple environment like Cartpole, which achieved much higher

Table 2: Average scores that can be obtained by learning a BOOK of a certain size and the number of transitions that is needed for A3C to get the same score.

	Cartpole		Acrobot		Q*bert	
Size	Score	Transition	Score	Transition	Score	Transition
250	114.0	25.8K	-143.8	330K	231.6	78K
500	**500.0**	**324K**	-158.5	204K	371.8	271K
1000	500.0	324K	**-91.8**	**372K**	436.0	383K
2000	500.0	324K	-93.2	363K	**520.0**	**618K**

score than conventional training methods. Some experiments show that the maximum score of 'BOOK + A3C' is same with that of 'A3C' but this is because their environments have a limited maximum score. Also, almost every experiments show that the red score starts from lower than the yellow baseline as additional training progresses. It may seem weired but it is very natural phenomenon for the following reasons: (1) As additional training begins, exploration is performed. (2) BOOK stores Q value with actual reward without bootstrapping, but DQN and A3C use bootstrapped Q value, thus they (actual and bootstrapped Q-values) don't match exactly.

5.2 Qualitative Analysis

To further investigate the characteristics of the proposed method, we conducted some experiments by changing the hyper-parameters.

5.2.1 Learning with Different Sizes of BOOKs

To investigate the effect of the BOOK size, we tested the performance of the proposed method using the published BOOK size of 250, 500, 1000, and 2000. Table 2 shows the score obtained by our baseline network which was trained using only the BOOK in a specified size. Also, in the table, we showed the number of transitions (experiences) that a conventional A3C has to go through to achieve the same score. This result shows that a relatively small number of linguistic states can achieve a score similar to that of the conventional network with only the published BOOK. As shown in the table, training an agent in a complex environment requires more information and therefore a larger BOOK is needed.

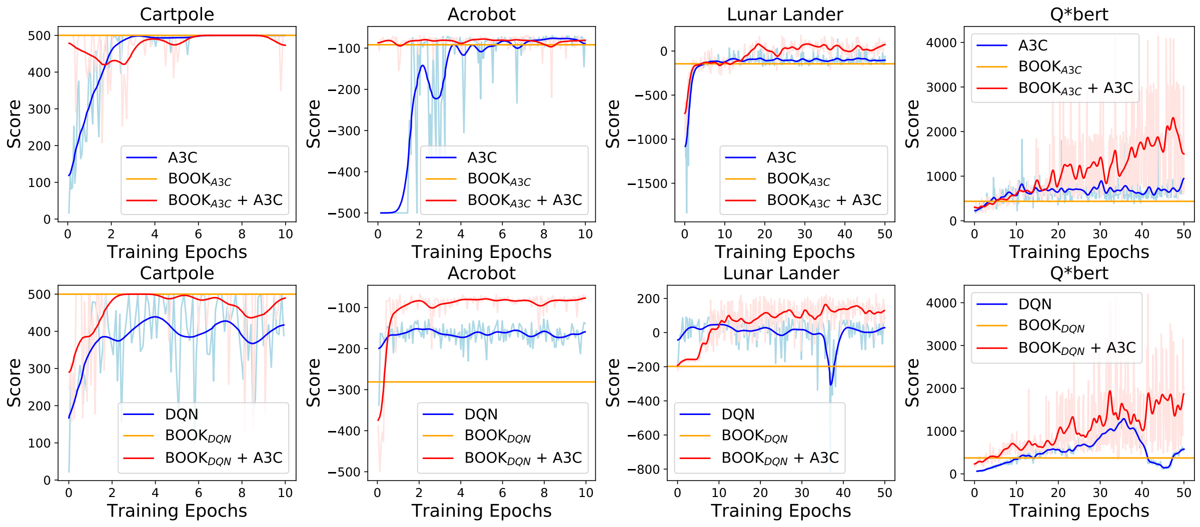


Figure 3: Performance for additional training after learning the BOOK. The upper row shows the case where a book is created in A3C and it is trained in a new A3C agent, and the lower row shows the case where a book is created in DQN and trained in A3C. **Blue**: Conventional method (A3C or DQN); **Yellow**: The score of the network that was trained only with the BOOK; **Red**: The network was trained using conventional method after learning the BOOK; An epoch corresponds to one hundred thousand transitions (across all threads). Light colors represent the raw scores and dark colors are smoothed scores.

5.2.2 Effects of Different Quantization Levels

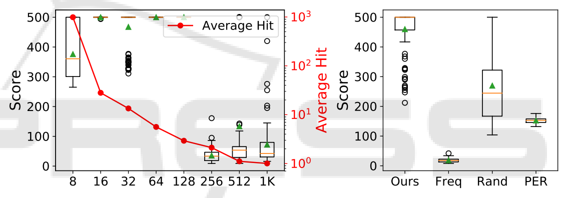
In this experiment, we confirmed the performance difference according to the resolutions of linguistic function. First of all, we differentiated the quantization level and published a BOOK of 1,000 size to check the difference of performance according to the resolutions of linguistic function. Figure 4(a) shows the distribution of scores according to the quantization level (quartile bar) and the average number of hits in each linguistic state c_k included in the BOOK (red line).

From Fig. 4(a), we found that the number of hit for each linguistic state decreases exponentially as the quantization level increases. Also, when the quantization level is high, the importance of c_k in equation (7) couldn't be defined and its score decreased because hit ratio becomes low. It can be seen that the highest and stable scores are obtained at quantization level of 64 and 128.

5.2.3 Comparison of the Priority Methods

Also, to verify the usefulness of our priority method (6), we tested the algorithm with different design of the priority; random selection, frequency only, method from *prioritized experience replay* (Schaul et al., 2015), and the proposed priority method. A book capacity K was set to 10,000 for this test.

As shown in Figure 4(b), the algorithm applying the proposed priority term achieved clearly far superior performance than other settings. We note that the case of using only frequency term marked the lowest performance, even lower than the random case. This is because the learning process proceeds only with the experiences that appear frequently when the priority is set only by the frequency. Correspondingly, the information of the critical, but rarely occurred experiences are not reflected enough to the training and hence, leads to inferior performance.



(a) Quantization Level (b) Priority Methods

Figure 4: (a) Distribution of scores according to the quantization level and the average number of hits in each Linguistic State c_k included in the BOOK. (b) Distribution of scores when we use three different methods than our proposed priority method. **Freq**: state visiting frequency, **Rand**: random state selection, **PER**: priority term from *prioritized experience replay*. All data were tested in Cartpole, and scores were measured in 100 random episodes. The green triangle and the red bar indicate the mean and the median scores, respectively. Blank circles are outliers.

The priority term of the *prioritized experience replay* also marked poor results. It is better than using frequency only as a priority, but even lower than random selection. This algorithm is intended to give priority to the states that are not yet well learned among the entire experience replay memory, and is not designed to extract a few core states.

5.3 Implementation Detail

We set the maximum capacity K of a BOOK to $100,000$ while writing the BOOK. To maintain the size of the BOOK, only the top 50% experiences are preserved and the remaining experiences are deleted to save new experiences when the capacity exceeds K. As a linguistic rule, each dimension of the input state was quantized into 128 levels. We set the discount factor γ for rewards to 0.99. Immediate reward r was clipped from -1 to 1 at Q*bert and generalized with $\tanh(r/10)$ for the other 3 environments (Cartpole, Acrobot and Lunar Lander). The frequency limit F_l was set to 20 and the decay period T was set to 100.

Our method adopted the same network architecture with A3C for Atari Q*bert. But for the other 3 environments, we replaced the convolution layers of A3C to one fully connected layer with 64 units followed by ReLU activation. Each environment was randomly initialized. For Q*bert, it skipped a maximum of 30 initial frames for random initialization as in (Bobrenko, 2016). We used 8 threads to train A3C network and instead of using shared RMSProp, ADAM (Kingma and Ba, 2014) optimizer was used. All the learning rates used in our experiments were set to 5×10^{-4}. To write a BOOK, we trained only 1 million steps (experiences) for Cartpole and Acrobot and 5 million steps for Lunar Lander and Q*bert. After publishing a BOOK, we pre-trained a randomly initialized network for $10,000$ iterations with batch size 8, using only the contents in the published BOOK. It took less than a minute to learn a BOOK with 1 thread on Nvidia Titan X (Pascal) GPU and 4 CPU cores, for Q*bert.

6 CONCLUSION

In this paper, we have proposed a memory structure called BOOK that enables sharing knowledge among different deep RL agents. Experiments on multiple environments show that our method can achieve a high score by learning a small number of core experiences collected by each RL method. It is also shown that the knowledge contained in the BOOK can be effectively shared between different RL algorithms, which implies that the new RL agent does not have to repeat the same trial and error in the learning process and that the knowledge gained during learning can be kept in the form of a record.

As future works, we intend to apply our method to the environments with a continuous action space. Linguistic functions can also be defined in other ways, such as neural networks, for better clustering and feature representation.

ACKNOWLEDGEMENT

The research was supported by the Green Car development project through the Korean MTIE (10063267) and ICT R&D program of MSIP/IITP (2017-0-00306).

REFERENCES

Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846.

Bellman, R. (1957). A markovian decision process. Technical report, DTIC Document.

Bertsekas, D. P. and Tsitsiklis, J. N. (1995). Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE.

Bobrenko, D. (2016). Asynchronous deep reinforcement learning from pixels.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Catto, E. (2011). Box2D: A 2D physics engine for games.

Chang, K.-W., He, H., Daumé III, H., and Langford, J. (2015). Learning to search for dependencies. *arXiv preprint arXiv:1503.05615*.

Chang, S., Yang, J., Choi, J., and Kwak, N. (2018). Genetic-gated networks for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1752–1761.

Geramifard, A., Dann, C., Klein, R. H., Dabney, W., and How, J. P. (2015). Rlpy: a value-function-based reinforcement learning framework for education and research. *Journal of Machine Learning Research*, 16:1573–1578.

Harmon, M. E., Baird III, L. C., and Klopf, A. H. (1995). Advantage updating applied to a differential game. In *Advances in neural information processing systems*, pages 353–360.

Hasselt, H. V. (2010). Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krishnamurthy, A., EDU, C., Daumé III, H., and EDU, U. (2015). Learning to search better than your teacher. *arXiv preprint arXiv:1502.02206*.

Langley, P. (2000). Crafting papers on machine learning. In Langley, P., editor, *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 1207–1216, Stanford, CA. Morgan Kaufmann.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

Lin, L.-J. (1993). *Reinforcement learning for robots using neural networks*. PhD thesis, Fujitsu Laboratories Ltd.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20.

Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. (2017). Neural episodic control. In *International Conference on Machine Learning*, pages 2827–2836.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer.

Ross, S. and Bagnell, J. A. (2014). Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering.

Salimans, T., Ho, J., Chen, X., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015). Trust region policy optimization. In *ICML*, pages 1889–1897.

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

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Werbos, P. J. (1992). Approximate dynamic programming for real-time control and neural modeling. *Handbook of intelligent control*.

Williams, R. J. (1987). A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *Proceedings of the IEEE First International Conference on Neural Networks*, volume 2, pages 601–608.