Faster Deep Q-learning
using Neural Episodic Control

Daichi Nishio
Institute of Science and Engineering
Kanazawa University
Kanazawa, Japan
Email: dnishio@csl.ec.t.kanazawa-u.ac.jp

Satoshi Yamane
Institute of Science and Engineering
Kanazawa University
Kanazawa, Japan
Email: syamane@is.t.kanazawa-u.ac.jp

Abstract—**The research on deep reinforcement learning which estimates Q-value by deep learning has been attracted the interest of researchers recently. In deep reinforcement learning, it is important to efficiently learn the experiences that an agent has collected by exploring environment. We propose NEC2DQN that improves learning speed of a poor sample efficiency algorithm such as DQN by using good one such as NEC at the beginning of learning. We show it is able to learn faster than Double DQN or N-step DQN in the experiments of Pong.**

Keywords-**Deep reinforcement learning; DQN; Neural Episodic Control; Sample efficiency**

I. INTRODUCTION

Deep Q-Network (DQN) [1] have made a success of deep reinforcement learning end-to-end, and various algorithms have been proposed since then [2]. However learning a task with large state space is difficult, and many learning steps are necessary especially in an environment where rewards are sparse.

TABLE I
THE DIFFERENCES BETWEEN DQN AND NEC

	DQN	NEC
Estimation	Neural Network	Neural Network + DND
Sampling	Random or with Priority	Random
Memory cost	Large	Huge

In order to solve them, it is necessary to efficiently use experiences obtained by exploration. DQN uses Experience Replay [3] which stores experiences in memory called Replay Buffer and It trains with minibatch randomly. Prioritized Experience Replay [4] has been proposed to learn more efficiently than random sampling. It considers experiences with large train error as important experiences, and greatly improves learning speed and performance.

Neural Episodic Control (NEC) [5] is another way to efficiently learn. It uses the memory module called Differentiable Neural Dictionary (DND) to learn stably with a smaller number of learning steps. The agent can decide its action by taking advantage of past similar experiences stored in DND. In addition, it is able to learn end-to-end because it is differentiable inside a neural network.

However, DND needs a large memory for each action. It also requires a lot of calculation time and memory usage. We show their relations in Table I.

In this research, we propose a method of improving learning efficiency and speed by adapting NEC's learning efficiency to a simple network like DQN.

II. DEEP REINFORCEMENT LEARNING

We target reinforcement learning assuming general Markov Decision Process(MDP). We define the state of the environment at time t as s_t, and the agent selects the action a_t by the policy π. Then it obtains the reward r_t and the next state s_{t+1} corresponding to a_t from the environment. The revenue is discounted return $G_t = \sum_t (\gamma^t r_t)$, where γ is discount rate as the degree of consideration of the future. The action-value function the agent uses for selecting action is defined as $Q_\pi(s, a) = E_\pi[G_t \mid s, a]$. The optimal Q-value is based on the Bellman optimal equation [6] as follows.

$$Q^*(s, a) = E[r + \gamma \max_{a'} Q(s', a') \mid s, a] \qquad (1)$$

Q-learning [7] is used to obtain the optimal Q-value.

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \quad (2)$$

In Q-learning, assuming that samples (s, a, r, s') can be obtained infinitely from all pairs of (s, a), (2) obtains the optimal Q-value function $Q^*(s, a)$. Also, it converges to the same value since the Q-value function does not depend on the policy. On the other hand, the convergence may take a long time if there are pairs (s, a) that are not tried.

In DQN, the agent uses the ε-greedy policy for tradeoff between exploration and exploitation. This $\varepsilon(0 < \varepsilon < 1)$ is a constant, or there is a method of linear decaying as increasing learning steps.

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon & (a = \mathrm{argmax}_a Q(s, a)) \\ \varepsilon & (otherwise) \end{cases} \qquad (3)$$

DQN aiming at learning from images uses Convolutional Neural Network (CNN) [8] as a state feature extractor. The Q-value for embedding h obtained from CNN is estimated by fully-connected layers.

The agent stores tuples of (s_t, a_t, r_t, s_{t+1}) in Replay Buffer so that it can learn with a minibatch formed randomly from

IEEE
computer
society

the buffer. In addition, for stability of the target value, it uses a target network for calculating a target value separately from a neural network for learning. The target network uses parameter θ^- which is slightly older than the current parameter θ of the learning network.

A neural network learns from the loss function $L(\theta) = E[y_t - Q(s, a; \theta)]$ using the target value in (4).

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) \qquad (4)$$

Hasselt et al. [9] show that DQN overestimates the action-value when the number of experience samples obtained from a environment is small, and Double Q-learning is a solution to it. Double Q-learning updates the Q-value using two Q-value estimators A and B.

$$Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(r + \gamma Q^B(s', a^*) - Q^A(s, a)) \quad (5)$$

$$Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(r + \gamma Q^A(s', b^*) - Q^B(s, a)) \quad (6)$$

$$where \ a^* = \operatorname*{argmax}_a Q^A(s', a), b^* = \operatorname*{argmax}_a Q^B(s', a)$$

Q^A prevents Q^B from overestimated, and Q^B prevents Q^A from it, respectively.

The algorithm called Double DQN [10] using Double Q-learning for DQN has been also proposed. Equation (4) is rewritten as follows.

$$y_t = r_t + \gamma Q(s_{t+1}, \operatorname*{argmax}_a Q(s_{t+1}, a; \theta); \theta) \qquad (7)$$

Hasselt et al. have incorporated the idea of Double Q-learning into (7).

$$y_t^{Double} = r_t + \gamma Q(s_{t+1}, \operatorname*{argmax}_a Q(s_{t+1}, a; \theta); \theta^-) \qquad (8)$$

This makes it possible to get higher scores with 90% of games played by DQN, and it is still widely used as an better algorithm than DQN.

III. RELATED WORK

Neural Episodic Control(NEC) is one of the method of efficient sampling from Replay Buffer. It is the algorithm based on episodic memory and improved Model-Free Episodic Control (MFEC) [11] to learn end-to-end from state mappings to estimations of Q-values.

Differentiable Neural Dictionary(DND) has been proposed to make it successful. DND for a action $a \in A$ is a dictionary $M_a = (K_a, V_a)$ which saves a pair of key K_a and value V_a. The key is the embedding h which is the feature extracted the state $s \in S$ with CNN, and the value is the Q-value.

We can perform two kinds of operations for DND, $Lookup$ and $Write$. In $Lookup$, when h featured by CNN and corresponding action a are entered, we lookup the top p-nearest neighbors for h in M_a using kd-trees [12]. We weight the value v_i corresponding to p keys and we set it as Q_a.

$$w_i = \frac{k(h, h_i)}{\sum_j k(h, h_j)} \qquad (9)$$

$$Q_a = \sum_i w_i v_i \qquad (10)$$

$k(h, h_i)$ is a kernel function for h and h_i. In DND, (11) is used.

$$k(h, h_i) = \frac{1}{\|h - h_i\|_2^2 + \delta} \qquad (11)$$

Although δ is a parameter to prevent division by zero, we should make it little larger such as $\delta = 10^{-3}$ because each value of p-nearest neighbors is referred to a certain extent.

In $Write$, we write an input h and a corresponding Q-value in DND. If the key that already matches the input h exists in M_a, update the corresponding Q-value according to the following with a learning rate α.

$$Q_i \leftarrow Q_i + \alpha(Q^{(N)}(s, a) - Q_i) \qquad (12)$$

When the size of the dictionary reaches the upper limit, we overwrites a pair that has not been referred to recently as the top p-nearest neighbor value according to Least Recently Used (LRU).

NEC uses N-step Q-learning [13] as a target value.

$$Q^{(N)}(s_t, a) = \sum_{j=0}^{N-1} \gamma^j r_{t+j} + \gamma^N \max_{a'} Q(s_{t+N}, a') \qquad (13)$$

However, N-step Q-learning is hard to be stabilized by off-policy algorithm [14].

We use NEC in our proposed algorithm, but we replace the output Q-value of NEC network with the Q-value defined in Chapter IV.

IV. PROPOSED ALGORITHM

NEC currently has the following problems.
1) As the number of pairs of key and value in DND increase, the computation time for finding top p-nearest neighbors increases.

2) The number of dictionaries M of DND is the same as the size of the action space $|A|$.

3) State space and action space need to be discrete because Q-learning is used.

Regarding 3, Matsumori et al. [15] are addressing research in an environment where the state space is continuous and Partially Observable Markov Decision Process (POMDP).

Our work focuses on the problems 1 and 2. Continuing to use NEC requires many computational resources due to constraints of time computational quantity and space computational quantity. Therefore, we will address using NEC's sampling efficiency only in early learning of other deep Q-learning algorithms. We will use DQN as an example which is the simplest Deep Q-learning network and we call this algorithm NEC2DQN(N2D).

As mentioned in [6], there is always one optimal action-value Q^*, and if it is the same policy, it always converges to the same value. For this reason, both DQN algorithm and NEC algorithm are possible to head to the same Q^*. Therefore, the Q-value estimated by other algorithms can be taken as the
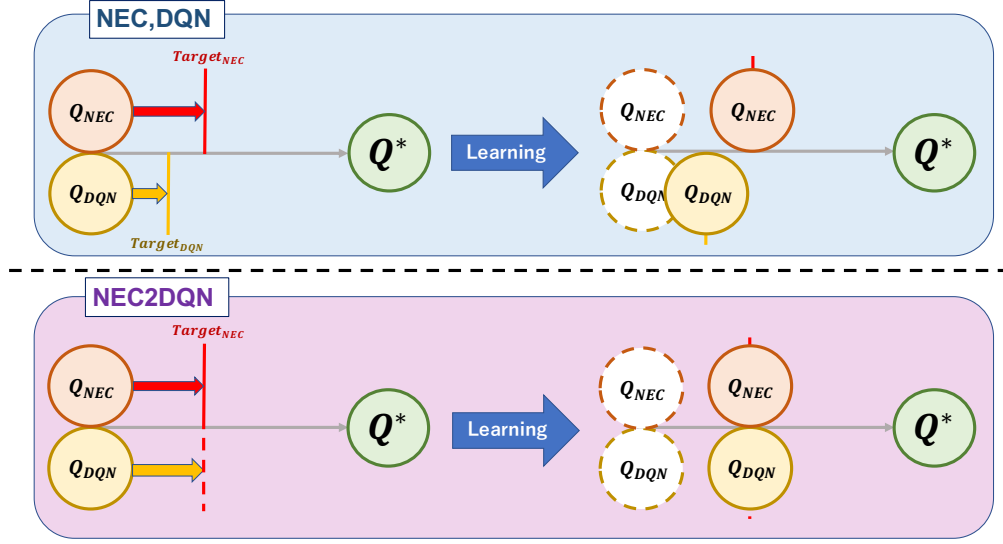
Figure 1. The image of the convergence of NEC2DQN: Q_{NEC} and Q_{DQN} will converge to the same optimal state-action value Q^*. The top shows the progress of convergence when NEC and DQN learn respectively. The bottom is the progress of convergence of each Q-value by using NEC2DQN. Although Q^* is unknown and cannot be observed directly, NEC and DQN should be close to Q^*. In the early stage of learning, it is better to use the $Target_{NEC}$ as shown in the figure. Therefore, we want DQN to learn by approaching not $Target_{DQN}$ but $Target_{NEC}$.

target value. Hence it is easier to converge by using a better target value, the value approaches to Q^* faster. We show this simple image in Figure 1.

We set $Q^A(s,a) = Q_{DQN}(s,a)$, $Q^B(s,a) = Q_{NEC}(s,a)$ in (5) and (6). Then we rewrite them as follow.

$$Q_{DQN}(s,a) \leftarrow Q_{DQN}(s,a) + \alpha(r + \gamma Q_{NEC}(s',a^*) - Q_{DQN}(s,a)) \tag{14}$$

$$Q_{NEC}(s,a) \leftarrow Q_{NEC}(s,a) + \alpha(r + \gamma Q_{DQN}(s',b^*) - Q_{NEC}(s,a)) \tag{15}$$

$$where \ a^* = \operatorname*{argmax}_a Q_{DQN}(s',a), b^* = \operatorname*{argmax}_a Q_{NEC}(s',a)$$

Double DQN is one-step Q-learning, but NEC uses N-step returns. Thus we want to rewrite (14) and (15) like N-step Double DQN. Instead of preparing a target network, NEC stores the target value $Q_{NEC}^{(N)}$ in Replay Buffer. In order to fit this target value, we adopt N-step returns for DQN(hereinafter, we call it N-step DQN).

$$Q_{DQN}(s,a) \leftarrow Q_{DQN}(s,a) + \alpha(Q_{NEC}^{(N)}(s,a^*) - Q_{DQN}(s,a)) \tag{16}$$

$$Q_{NEC}(s,a) \leftarrow Q_{NEC}(s,a) + \alpha(Q_{DQN}^{(N)}(s,b^*) - Q_{NEC}(s,a)) \tag{17}$$

Equations (16) and (17) show $Q_{NEC}^{(N)}(s,a^*)$ is necessary for learning $Q_{DQN}(s,a)$, and $Q_{DQN}^{(N)}(s,b^*)$ is necessary for learning $Q_{NEC}(s,a)$. However, Q_{NEC} may not be able to learn well until DQN learns as a network, and there is a possibility that Q_{DQN} cannot be learned well due to the influence. Especially, NEC should be able to learn earlier by learning using the original $Q_{NEC}^{(N)}$. Therefore we rewrite (17) like (19).

$$Q_{DQN}(s,a) \leftarrow Q_{DQN}(s,a) + \alpha(Q_{NEC}^{(N)}(s,a^*) - Q_{DQN}(s,a)) \tag{18}$$

$$Q_{NEC}(s,a) \leftarrow Q_{NEC}(s,a) + \alpha(Q_{NEC}^{(N)}(s,b^*) - Q_{NEC}(s,a)) \tag{19}$$

$Q_{DQN}(s,a)$ can learn with a better target value by NEC algorithm at the beginning of learning. Even if learning this way, Q_{DQN} converges to the same value as Q_{NEC}. But we need to consider the learning has advanced because this research aims to use NEC only in the early stage of learning. Since Q_{DQN} maybe estimated correctly to some extent, it can be learned like Double DQN using N-step returns.

$$Q_{DQN}(s,a) \leftarrow Q_{DQN}(s,a) + \alpha(Q_{DQN}^{(N)}(s,a^*) - Q_{DQN}(s,a)) \tag{20}$$

$$Q_{NEC}(s,a) \leftarrow Q_{NEC}(s,a) + \alpha(Q_{DQN}^{(N)}(s,b^*) - Q_{NEC}(s,a)) \tag{21}$$

Comparing (18) with (19), they use the same target value $Q_{NEC}^{(N)}$. Similarly, both (20) and (21) use $Q_{DQN}^{(N)}$. Hence we replace them with the same Q-value $Q_{N2D}^{(N)}$.

$$Q_{DQN}(s,a) \leftarrow Q_{DQN}(s,a) + \alpha(Q_{N2D}^{(N)}(s,a^*) - Q_{DQN}(s,a)) \tag{22}$$

$$Q_{NEC}(s,a) \leftarrow Q_{NEC}(s,a) + \alpha(Q_{N2D}^{(N)}(s,b^*) - Q_{NEC}(s,a)) \tag{23}$$

From here, we define $Q_{N2D}(s,a)$ to satisfy the above property. We prepare networks of NEC and DQN separately as shown in Figure 2. Both of the networks output their Q-value for the state s_t, and we combine them as $Q_{N2D}(s_t,a)$.

$$Q_{N2D}(s_t,a) = \lambda(t)Q_{NEC}(s_t,a) + (1-\lambda(t))Q_{DQN}(s_t,a) \tag{24}$$

This $\lambda(t)$ is a monotonously decreasing function related to the current learning step t. It represents how much to consider NEC. We make it linear decay from 1 to 0 as increasing
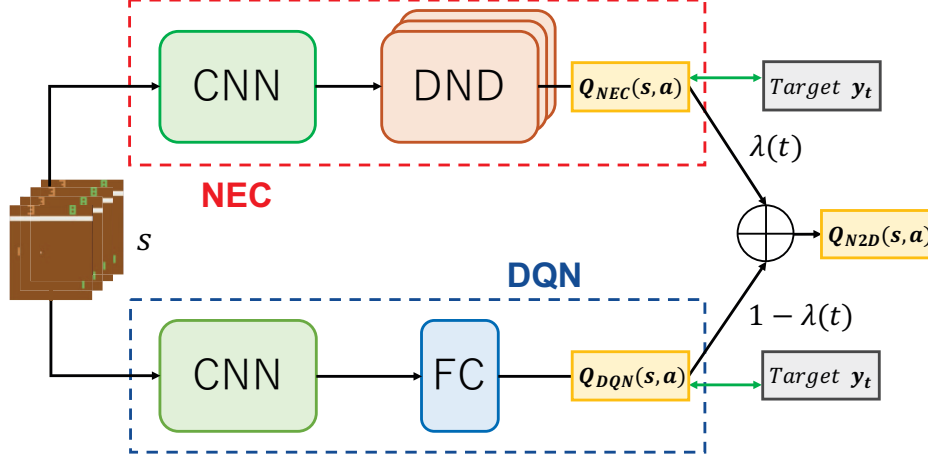
Figure 2. NEC2DQN Network

learning steps such as (25).

$$\lambda(t) = \begin{cases} 1 - \frac{t}{CS} & (t < CS) \\ 0 & (otherwise) \end{cases} \quad (25)$$

At the beginning of learning, it refers to the Q-value of NEC, and gradually refers to the Q-value of DQN as learning progresses. Thus it is possible to switch naturally.

We set the steps CS to start to fully depend on DQN. It is not necessary to calculate $Q_{NEC}(s_t, a)$, and the calculation time is also reduced because $\lambda(t) = 0$ after CS.

Loss functions of NEC and DQN are required respectfully because they are separate networks, but we use the same target value y_t. Although we use a^* and b^* in (22) and (23), we do not use Double Q-learning but simply use N-step returns since we do not want to use $Q_{N2D}^{(N)}(s, a^*)$.

$$y_t = Q_{N2D}^{(N)}(s, a) = \sum_{j=0}^{N-1} \gamma^j r_{t+j} + \gamma^N \max_{a'} Q_{N2D}(s_{t+N}, a') \quad (26)$$

This is based on the fact that it is hard for overestimation of the Q-value to occur when the amount of experience accumulated in Replay Buffer is small because NEC can refer to DND.

These loss functions also use same target y_t.

$$L_{NEC}(\theta_t) = E[y_t - Q_{NEC}(s_t, a_t)] \quad (27)$$

$$L_{DQN}(\theta_t) = E[y_t - Q_{DQN}(s_t, a_t)] \quad (28)$$

Similarly to original NEC, Replay Buffer stores a tuple of (s_t, a_t, y_t). We accumulate the trajectory of the episode at the end of each episode because y_t requires N steps of reward data and the subsequent state s_{t+N}.

We show the overall algorithm in Algorithm 1.

V. IMPLEMENTATION

We use Pong of Atari 2600 provided with OpenAI Gym [16] which can easily share the result of library and reinforcement learning algorithms.

Algorithm 1 NEC2DQN
1: Initialize the number of entire timesteps S to 0.
2: Initialize the change step CS for $\lambda(t)$
3: Initialize replay memory D to capacity C_D.
4: Initialize DND memories M_a to capacity C_{M_a}.
5: Initialize action-value function Q_{NEC} and Q_{DQN} with random weights.
6: **for** each episode **do**
7:     Initialize trajectory memory G .
8:     **for** $t = 1, 2, ...; T$ **do**   ▷ Explore and train.
9:         Receive observation s_t from environment.
10:         Receive $Q_{DQN}(s_t, a)$.
11:         **if** $S < CS$ **then**   ▷ If $S < CS$, we use NEC.
12:             Receive embedding h and $Q_{NEC}(s_t, a)$.
13:         **else**
14:             Set $Q_{NEC}(s_t, a)$ to free values.
15:         **end if**
16:         Calculate $Q_{N2D}(s_t, a)$.   ▷ Calculate by (24).
17:         $a_t \leftarrow \varepsilon$-greedy policy based on $Q_{N2D}(s_t, a)$.
18:         Take action a_t, receive reward r_t.
19:         Append (s_t, a_t, r_t) to G.
20:         Train on a random minibatch from D.
21:         $S \leftarrow S + 1$
22:     **end for**
23:     **for** $t = 1, 2, ...; T$ **do**   ▷ Calculate N-step returns.
24:         Calculate y_t.   ▷ Calculate by (26).
25:         Append (s_t, a_t, y_t) to D.
26:         **if** $S < CS$ **then**
27:             Append (h_t, y_t) to M_{a_t}.
28:         **end if**
29:     **end for**
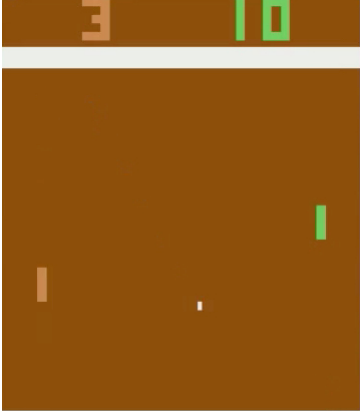30: **end for**

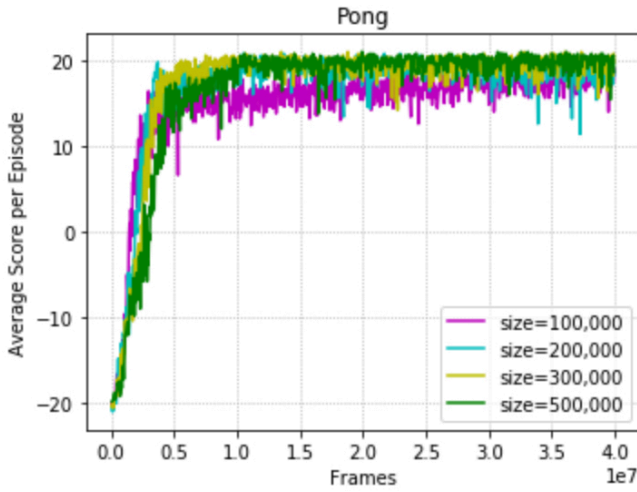Figure 3.   A Pong frame

Figure 4.   The comparing NEC2DQN with other algorithms

Figure 5.   The difference of NEC2DQN performance by Replay Buffer size
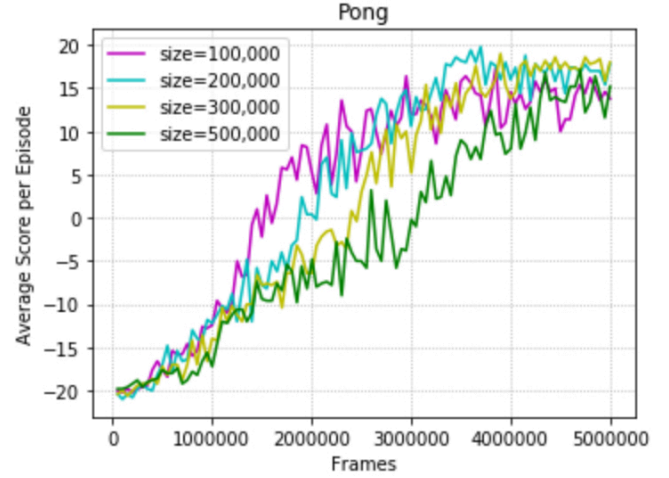

Figure 6.   The first 5M steps in Figure 5.

Pong is the game of 21 points win. If the opponent cannot hit the ball, we get a reward of +1, and if we cannot, we get a reward of -1. That is, the reward set $R = \{-1, 0, 1\}$. Many of the algorithms targeting versatility such as DQN use Reward Clipping that clips rewards gained from games to $[-1, 1]$. It is a technique that we enables an agent to learn without changing other parameters. However, Reward Clipping may make it impossible to distinguish between a high reward and a small reward with a large absolute value, so there is a possibility that it will not try to obtain high rewards [17]. We do not consider it in this research because the Pong's reward is $r \in R$.

Like the DQN, we process one frame of the game such as Figure 3 to 84×84 and convert to grayscale and normalize it. Then the state is set with the consecutive 4 frames together. We show the main parameters and the network parameters setting for the experiment in Table II and Table III as Appendix.

We evaluate the learning speed of NEC2DQN, N-step DQN and Double DQN. Also we observe the learning result by the

difference in the size of Replay Buffer. We test 5 times every 50,000 learning steps, and we use greedy policy ($\varepsilon = 0$) in every test.

VI. RESULT

Figure 4 is a comparison with NEC2DQN, N-step DQN ($N = 10$) and Double DQN. Although Double DQN required more than 30M steps to earn more than 10 points, NEC2DQN achieved it in about 3M steps. Furthermore, NEC2DQN has better learning efficiency and performance than N-step DQN.

Figure 5 and Figure 6 show the difference in results depending on Replay Buffer size. The size affects the stability of sampling. The larger it is, the better the performance is. However, since the target value is also stored in Replay Buffer, the target value is too old to proceed well if it is too large like $size = 500,000$. Looking at Figure 6, the smaller size is, the faster learning is due to the newness of the target value at the 2M frame. But as learning progresses, it turns out that the

game score is not obtained well in the case the size is small like $size = 100,000$. NEC2DQN needs considering this balance.

VII. CONCLUSION

In this research, we have verified DQN is possible to use the good sampling efficiency of NEC in the beginning of learning using a Pong example. We have showed that the learning speed is faster than only DQN by using the same and better target values for NEC and DQN. Indeed, we observed a significant learning speed improvement by using NEC during 2M steps only.

However it is necessary to confirm whether this method succeeds also in other games. Particularly, although NEC does not need to Reward Clipping, DQN is better to do Reward Clipping. Thus, there is a possibility that we cannot use the NEC's advantage.

Moreover, Since NEC and DQN do not connect as the same network, updating weights of the networks do not directly affect each other. Therefore, it is easy to replace it with a network other deep Q-learning algorithm. It is necessary to verify whether other deep Q-learning algorithms also works as well as DQN by changing Q_{DQN} used for Q_{N2D} to the Q-value of others.

The most important issue is that multiple networks learn while choosing appropriate target values. In this paper, we have used NEC since it is excellent in Q-value estimation at the beginning of learning, but NEC is not necessarily useful for estimating Q-value. Deep Q-learning from Demonstrations (DQfD) [18] is used to estimate the Q-value with reference to human play. It is sometimes hard to go well in tasks that humans cannot do very well (such as Pong), but if it is a task that humans can do well, it is good for learning faster than NEC. However, it is difficult to use human demos when we make our agent learn more complicated and time-consuming tasks. It is important that various networks of deep Q-learning cooperate by learning while choosing an appropriate target value automatically.

REFERENCES

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

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A Brief Survey of Deep Reinforcement Learning. arXiv:1708.05866, 2017.

[3] Long-Ji Lin. Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. Machine Learning, 8(3-4):293-321,1992.

[4] Schaul, Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. In ICLR, 2016.

[5] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria' Puigdome'nech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural Episodic Control. arXiv:1703.01988, 2017.

[6] Richard Bellman. On the Theory of Dynamic Programming. In PNAS, 38(8):716-719, 1952.

[7] Christopher JCH Watkins and Peter Dayan. Q-Learning. Machine Learning, 8(3-4):279-292, 1992.

[8] Alex Krizhevsky, Ilya Sutskever, Geoffrey E,Hinton. ImageNet classification with deep convolutional neural networks. In ANIPS, 2012.

[9] Hado van Hasselt. Double Q-Learning. In NIPS, 2010.

[10] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning. In AAAI, 2016.

[11] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-Free Episodic Control. arXiv:1606.04460, 2016.

[12] Bentley, Jon Louis. Multidimensional binary search trees used for associative searching. Commun. ACM, 18(9): 509-517, 1975.

[13] Jing Peng and Ronald Williams. Incremental multi-step Q-learning. Machine Learning, 22:283- 290, 1996.

[14] Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G Bellemare. Safe and Efficient Off-Policy Reinforcement Learning. In NIPS, 2016.

[15] Shoya Matsumori, Takuma Seno, Toshiki Kikuchi, Yusuke Takimoto, Masahiko Osawa, Michita Imai. Embedding Cognitive Map in Neural Episodic Control. In SIG-AGI, 2017

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. arXiv:1606.01540, 2016.

[17] Hado van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, David Silver. Learning values across many orders of magnitudes. In NIPS, 2016.

[18] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. Deep Q-learning from Demonstrations. arXiv:1704.03732, 2017.

APPENDIX

TABLE II
HYPER PARAMETER

Parameter	Value
Optimizer	RMSProp($\varepsilon = 0.01$)[†]
Optimizer learning rate	0.000025
Optimizer momentum	0.95[†]
Explore ε	$1 \rightarrow 0.01$ over 1M steps
Replay buffer size	300,000
DND learning rate	0.1
DND size	500,000 per action[††]
p for KDTree	50[††]
N-step returns N	10
NEC embedding size	64
DQN embedding size	512[†]
Replay period	every 4 learning steps[†]
Minibatch size	32[†]
Discount rate	0.99[†]
NEC2DQN change step CS	2M steps

[†]same as Double DQN [10] [††]same as NEC [5]

TABLE III
NETWORK

Parameter	Value
NEC,DQN: CNN channels	32, 64, 64
NEC,DQN: CNN filter size	$8 \times 8, 4 \times 4, 3 \times 3$
NEC,DQN: CNN stride	4, 2, 1
NEC : embedding size	64
DQN : hidden layer	512
DQN : output units	Number of actions