J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

오프 폴리시 강화학습에서 몬테 칼로와 시간차 학습의 균형을 사용한 적은 샘플 복잡도☆

Random Balance between Monte Carlo and Temporal Difference in off-policy Reinforcement Learning for Less Sample-Complexity

김 차 영[1] 박 서 희[2] 이 우 식[3*]
Chayoung Kim Seohee Park Woosik Lee

요 약

강화학습에서 근사함수로써 사용되는 딥 인공 신경망은 이론적으로도 실제와 같은 근접한 결과를 나타낸다. 다양한 실질적인 성공 사례에서 시간차 학습(TD) 은 몬테-칼로 학습(MC) 보다 더 나은 결과를 보여주고 있다. 하지만, 일부 선행 연구 중에서 리워드가 매우 드문드문 발생하는 환경이거나, 딜레이가 생기는 경우, MC 가 TD 보다 더 나음을 보여주고 있다. 또한, 에이전트가 환경으로부터 받는 정보가 부분적일 때에, MC가 TD보다 우수함을 나타낸다. 이러한 환경들은 대부분 5-스텝 큐-러닝이나 20-스텝 큐-러닝으로 볼 수 있는데, 이러한 환경들은 성능-퇴보를 낮추는데 도움 되는 긴 롤-아웃 없이도 실험이 계속 진행될 수 있는 환경들이다. 즉, 긴 롤-아웃에 상관없는 노이지가 있는 네트웍이 대표적인데, 이때에는 TD 보다는 시간적 에러에 견고한 MC 이거나 MC와 거의 동일한 학습이 더 나은 결과를 보여주고 있다. 이러한 해당 선행 연구들은 TD가 MC보다 낫다고 하는 기존의 통념에 위배되는 것이다. 다시 말하면, 해당 연구들은 TD만의 사용이 아니라, MC와 TD의 병합된 사용이 더 나음을 이론적이기 보다 경험적 예시로써 보여주고 있다. 따라서, 본 연구에서는 선행 연구들에서 보여준 결과를 바탕으로 하고, 해당 연구들에서 사용했던 특별한 리워드에 의한 복잡한 함수 없이, MC와 TD의 밸런스를 랜덤하게 맞추는 좀 더 간단한 방법으로 MC와 TD를 병합하고자 한다. 본 연구의 MC와 TD의 랜덤 병합에 의한 DQN과 TD-학습만을 사용한 이미 잘 알려진 DQN과 비교하여, 본 연구에서 제안한 MC와 TD의 랜덤 병합이 우수한 학습 방법임을 OpenAI Gym의 시뮬레이션을 통하여 증명하였다.

☞ 주제어 : 온- 앤 오프-폴리시, 시간차 학습, 몬테 칼로 학습, 강화학습, 분산과 편차의 균형

ABSTRACT

Deep neural networks(DNN), which are used as approximation functions in reinforcement learning (RN), theoretically can be attributed to realistic results. In empirical benchmark works, time difference learning (TD) shows better results than Monte-Carlo learning (MC). However, among some previous works show that MC is better than TD when the reward is very rare or delayed. Also, another recent research shows when the information observed by the agent from the environment is partial on complex control works, it indicates that the MC prediction is superior to the TD-based methods. Most of these environments can be regarded as 5-step Q-learning or 20-step Q-learning, where the experiment continues without long roll-outs for alleviating reduce performance degradation. In other words, for networks with a noise, a representative network that is regardless of the controlled roll-outs, it is better to learn MC, which is robust to noisy rewards than TD, or almost identical to MC. These studies provide a break with that TD is better than MC. These recent research results show that the way combining MC and TD is better than the theoretical one. Therefore, in this study, based on the results shown in previous studies, we attempt to exploit a random balance with a mixture of TD and MC in RL without any complicated formulas by rewards used in those studies do. Compared to the DQN using the MC and TD random mixture and the well-known DQN using only the TD-based learning, we demonstrate that a well-performed TD learning are also granted special favor of the mixture of TD and MC through an experiments in OpenAI Gym.

☞ keyword : Deep Q-Network, Temporal Difference, Monte Carlo, Reinforcement Learning, Variation and Bias Balance

1 Division of General Studies, Kyonggi University, 154-42, Gwanggyosan-ro, Yeongton-gu, Suwon, Korea.
2 KT, 151, Taebong-ro, Seocho-gu, Seoul, Korea.
3 SSiS, 173, Toegye-ro, Jung-gu, Seoul, Korea.
* Corresponding author (wslee@ssis.or.kr)

[Received 25 May 2020, Reviewed 15 June 2020(R2 24 July 2020), Accepted 9 August 2020]
☆ A preliminary version of this paper was presented at ICONI 2019 and was selected as an outstanding paper.

1. Introduction

Deep neural networks (DNN) as function approximators in reinforcement learning (RL) [1, 2] has significantly enlarged dealing with real environment. Theoretical results could be helped mainly on linear function approximators within a set of narrow environment. However, their theoretical assumptions do not consider the genuine real-world application domains of deep RL, such as high input dimensionality of feature patterns or non-linear function approximators. Such expressive parameterization in the DNN also brings up thousands of practical issues. It tends to be sensitive to hyper-parameter. Moreover, poor hyper-parameter settings lead to unstable or non-convergent training or diverge to infinity. Also, Deep RL [3] is more likely to exhibit high sample complexity, which is impractical to the real-world problems like DNN. It is regarded that batch policy gradient RL offers stability of learning than Deep RL. However, it leads to the high-variance requiring large batches because their estimations are continually growing. So, TD-style [4] techniques, for example Q-learning or actor-critic, are regarded helpful for sample-efficient but still biased, and require expensive hyper-parameters setting for better stabilization. Also, MC of RL [1, 2] is regarded as that it can offer nearly unbiased because MC policy gradient is common for on-policy techniques. However, MC might be occasionally suffering from high variance. Therefore, to deal with high variance gradient and such a difficult optimal parameterization, there are some previous valuable research works, such as mixing value-based back-ups in MC [4]. However, most those works require huge amount of samples and some complicated formular of rewards for dealing with these real-world problems, which is very intensive in terms of Big-Data. Off-policy Q-learning [3, 5] and off-policy actor-critic [4] can use all samples by TD-learning combined with experience replay in deep neural network for sample-efficient. Such architectures are significantly useful in terms of Big-Data samples. Still, there are some issues on a convergence of TD-learning, which is not guaranteed with non-linear function approximators. Non-convergence and instability issues still require extensive hyper-parameter tuning and human-interactions[6].

In some benchmark works, TD based on a combination of MC theory and dynamic programming (DP) [2, 4] theory has been used for better empirical results rather than theoretical results. Moreover, some recent works show that finite-horizon MC is a little superior than TD, when it comes to sparse or delayed rewards. Moreover, recent researches show that a technique based on MC prediction might outperform TD-based methods on complex control works in partially observables. Most of these environments can be regarded as 5-step Q-learning or 20-step Q-learning, where the experiment continues without long roll-outs for alleviating deterioration of performance results. In other words, for networks with a noise, a representative network that is regardless of the controlled roll-outs, it is better to learn MC, which is robust to noisy rewards than TD, or almost identical to MC. These studies provide a break with that TD is better than MC. The key point of recent research results is to suggest the ways combining MC and TD [4]. In value-based deep RL architectures with bootstrapping samples of Big-Data such as DQN [3], TD have been regarded as superior than MC. However, recent empirical researches have shown that MC is more stable to noisy and sparse rewards or a balance of TD and MC [4] are more practical for training an AI agent. Our study focus on discrete action sets and algorithms involving a prediction of value function, which can be learned via a combination of TD and MC to make value-based methods perform better.

Therefore, in this study, based on the results shown in previous researches, we attempt to exploit a random balance with a mixture of TD and MC in RL without any complicated formulas by rewards used in those researches do. We also demonstrate that DQN with a well-performed TD leaning are also granted special favor of the mixture of TD and MC at random. Moreover, our proposed algorithm goes through experimental comparison with the well-known DQN using only the TD-based learning. The result shows that our proposed algorithm has shorter training time than the well-known DQN.

2. BackGrounds

Reinforcement learning (RL) consists of an artificial

intelligence (AI) agent acting in an environment over discrete time-steps. An environment is defined by states s, actions a, a reward function r : s×a→r, a transition probability $p(s_{t+1}|s_t,a_t)$ and a discount factor $\gamma \in [0,1]$. Let π^* denote an optimal policy such as $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$ for every $s \in S$, $a \in A$ and any policy π In terms of Q-learning, the policy $\pi$is a maximum in every update of Big-Data. The objective of the equation is to find a policy $\pi(a_t|s_t)$ to maximize the expected sum of all future rewards through the episode such as $R_t = \sum T_{i=t}\ r_i$. (1). To avoid divergence for long episodes, long-distant rewards can be decayed by a discount factor γ or truncated until the explicit steps τ (horizon) such as $R_\gamma^t = \sum T_{i=t}\ \gamma^{i-t} r_i = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$; $R_t^\tau = \sum^{t+\tau}_{i=t} r_i$. (2). For a given policy π, the value function and the action-value function are defined as expected returns that are conditioned on observation or the observation-action pair respectively such as $V^{\pi}(s_t) = E_\pi[R_t|st]$, $Q^{\pi}(s_t,a_t)=E_\pi[R_t|s_t,a_t]$. (3). Optimal value and action-value functions are defined such as $V^*(s_t) = \max_\pi V^{\pi}(s_t)$, $Q^*(s_t,a_t) = \max_\pi Q^{\pi}(s_t,a_t)$. (4). In value-based RL such as Q-learning, the value or action-value are estimated by a function approximator V with parameters θ. The function approximation is trained by minimizing a loss between the current estimate and a target value such as $L(\theta) = (V(s_t;\theta) - V_{target})^2$.(5). Updates on the target each step makes the value or action-value stable.

Monte Carlo (MC) trains the AI agent with the formular, $V_{target} = R_t^\gamma$ or $V_{target} = R_t^\tau$. [0,1] This target requires propagation of the forward before a training step can take place, for example by the step τ for finite-horizon return R_t^τ or the end of the episode for the discounted return R_t^γ. It might increase the variance of the target value. However, it cannot be biased because it is not approximated. An alternative to MC training is temporal difference (TD), which estimates the return by bootstrapping samples of Big-Data from the function approximators, after acting for a fixed number of steps n such as $V_{target} = \sum^{t+n-1}_{i=t} \gamma^{i-t} r_i + \gamma^n V(s_t+n;\theta)$. (6). TD learning is used within finite-horizon returns. TD applied to the action-value function is the well-known Q-learning.

Usually, the classic Q-learning algorithms are used in synergy with deep neural networks, which can oscillate or diverge because the Q-value estimated by Q-learning algorithm are approximated. This limitation is caused by correlated Big-Data or continuously repeated updates. Therefore, some benchmark works [1, 2, 3] use an experience replay memories which samples at random from the mini-batch (s_t, a_t, r_t, s_{t+1}) from the memory buffers, D. So, the training are smoothly over many experience Big-Data. It is explicit to take advantage of the deep neural network (DNN) in RL because the all experiences of the trajectories, is buffered in D for $Q(s_{t+1}, a_{t+1}, \theta)$ at random and also reused by MC training. Figure 1 shows that Q-learning goes into DNN. So, it is called DQN, which can take advantage of the experience replay memories.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

(그림 1) DQN 알고리즘 [3]
(Figure 1) The DQN algorithm [3]

The Deep Neural Network (DNN) architecture is based on multiple stacked layers of neurons. A neuron is a non-linear transformation architecture of the linear sum of Big-Data sample inputs. Normally, the first layer models the data itself. Stacked hidden layers in the Neural Network(NN) is constructed as arrays of neurons receiving the inputs from the previous layer. The a neuron activator as a function on the top of the stacked layers in the NN is using composite functions, which show that supervised training of a DNN with non-linearities architecture is faster. So, most stacked hidden layers are composed of the activators such as Rectified Linear Units (ReLU) [7].

3. THE PROPOSED ALGORITHM

For better exploration and bias-variance balance, some researchers suggests a mixture of TD and MC [4], which is

applicable to high-dimensional discrete environments, partially observable or sparse by using DQN with replay memories. TD-based AI training has been used for more frequently and efficiently because of empirical reasons since the breaking results ATARI-BREAKOUT [3]. Based on the research [4], we propose the technique to exploit a random balance with a mixture of TD and MC in RL training, specifically DQN. Figure 2 shows our proposed algorithm of off-policy RL, DQN with Monte Carlo and Temporal Difference Balance at random. Mixture of MC and TD is accomplished using β_1 and β_2 random probabilities in [0, 1] for TD and MD, respectively.

Algorithm DQN with Monte Carlo and Temporal Difference Balance
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode =1 , M do
  Initialize sequence s_1 = {x_1} and preprocessed sequenced $\varphi_1 = \varphi(s_1)$
  for time=1, T do
    With probability ε select a random action a_t
      Otherwise select $a_t = \max_a Q^*(\varphi(s_t, a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1}=s_t, a_t, x_{t+1}$ and preprocess $\varphi_{t+1}=\varphi(s_{t+1})$
    Store transition ($\varphi_t, a_t, r_t, \varphi_{t+1}$) in D
    Sample random minibatch of transitions ($\varphi_j, a_j, r_j, \varphi_{j+1}$) from D
    Set $y_j = r_j$ for terminal φ_{j+1}
      Otherwise set $y_j = r_j + \gamma \max_a Q(\varphi_{j+1}, a'; \theta)$ for non-terminal φ_{j+1}
    With a random probability β_1 perform a gradient descent step
      on $(y_j - Q(\varphi_j, a_j; \theta)^2$
  end for
  With a random probability β_2 perform a gradient descent step
      on $(y_j - Q(\varphi_j, a_j; \theta)^2$
end for

(그림 2) 본 논문에서 제안하는 알고리즘
(Figure 2) The proposed algorithm

We attempt to combine Monte Carlo and Temporal Difference with the truncated-steps (horizon) through the whole roll-out. Our proposed algorithm is different from the results of the previous research [4]. It is simpler and random because we are targeting the goal without any complicated formula for the reward of the truncated-steps of exploration. We follow a random probability β_1 for performing a gradient descent of TD and β_2 for performing a gradient descent of MC. β_2 is (1 - β_1). So, the expression is y_j = β_1* y_TD + β_2 * y_MC. It is fundamentally based on a random policy with β_1 and β_2, respectively, for TD and MC. We demonstrate that both TD and MC methods benefit from our method through experimental comparison of the classic DQN using only TD on high-dimensional discrete action environments, such as the well-known environment OpenAI Gym [8].

3.1 Algorithm Description

<off-policy RL, DQN with Monte Carlo and Temporal Difference Balance>

1. Initialize replay memory D and action-value function Q with random weights
2. Initialize the sequence with the start state, s.
3. The agent learns the policy $\max_a Q^*(\phi(s_t),a); \theta)$ or follows another policy with probability ε
4. For better exploration, an experience composed of a tuple, such as (state $\phi(s)$, action a, reward r, new state $\phi(s')$) is in D selected randomly at every training step.
5. Sample a random mini-batch of transitions from D, such as (state $\phi(s)$, action a, reward r, new state $\phi(s')$)
6. For non-terminal state, reward based on $\max_a Q^*((\phi(s_t),a'); \theta)$ is decayed by γ or for terminal state, reward is the current reward, r.
7. The weights for performing the gradient descent $(r_j + \gamma \max_a Q^*(\phi(s_{j+1}), a; \theta) - Q(\phi(s_j), a; \theta))$ for a target DQN with replay memories with probability random β_1.
8. Steps 3 – 7 are repeated for training.
9. Before the next episode, the weights for performing the gradient descent $(r_j + \gamma \max_a Q^*(\phi(s_{j+1}), a; \theta) - Q(\phi(s_j), a; \theta))$ for a target DQN with the whole memories in every step with probability random β_2.
10. Steps 2 – 9 are repeated for training.

```
beta_1 = random()          # random function
beta_2 = 1 - beta_1        # beta_1 and beta_2
Q_target = reward + GAMMA * argmax(values)  # for TD-learning
        # reward from the environment
        # GAMMA is the discount_factor
        # Q_target is TD-learning
        # values are the estimated Q-values in every state
For every episode:                          # for MC-learning
    G_target = GAMMA * (reward * G_target)
    value = (value + ALPHA * (G_target - value) )
        # G_target is MC-learning
        # ALPHA is the learning rate
        # value is the estimated Q-value in a state
Y_target = beta_1 * Q_target + beta_2 * G_target
        # Y_target is a mixture of TD and MC
```

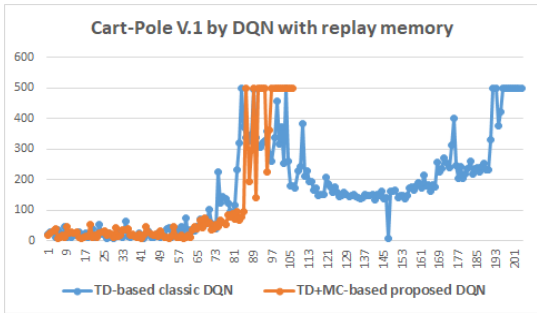
(그림 3) 본 논문에서 제안하는 알고리즘의 의사코드
(Figure 3) The pseudo code of the proposed algorithm

4. Performance Evaluation

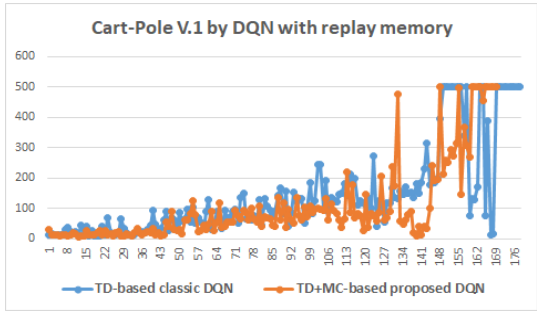
We exploit OpenAI Gym [8] for our proposed algorithms, DQN with TD and MC balance at random. We consider classic control environments in OpenAI Gym, such as CartPole-V0 [9]. With benchmark approaches [1, 3], we exploit an experience replay for better exploration. Moreover, in our proposed method, Q-learning stores past experiences at each time step in a buffer D, which is known as a replay memory. Our emulators from OpenAI Gym [8] can apply mini-batch updates in D. After the experience replay memory D, the agent's actions of the emulator follow εgreedy policy. In terms of DQN with the replay memory D for better exploration, we also follow the theory of the target Q-network of the breakthrough researches in [1, 3]. The Q-learning agent calculates the TD-error with the current estimated Q-value[2]. Updates based on the target network is slower than those on the current network.

In CartPole-V0 [9], a pole is attached with an unactuated joint to a cart moving along a frictionless track. It is controlled by forcing +1 or −1 to the cart. The pole starts upright, and the goal is to prevent it from falling over [9]. A +1 reward is given to every time step in which the pole remains upright [9]. The episode stops when the pole is more than 15° from the vertical direction or the cart moves more than 2.4 units from the center [9]. In our simulation, CartPole-V0 defines "solving" as if the average reward is more than 490 or equal to 500 over 10 consecutive runs [10]. The agent of CartPole-V0 receives −100 reward if it falls over prior to the max-length of the episode [10]. Our proposed algorithm with MC and TD balance at random is implemented using TensorFlow [11] and Keras [12]. For the proposed algorithm, we follow similar previous studies [10]. Figure 3 shows the pseudo code of the proposed algorithm. For CartPole-V0 [9] by OpenAI Gym [8], the discount-factor, GAMMA = 0.95, the learning rate, ALPHA = 0.001, the size of replay buffer, D = 10,000, the size of mini-bach = 64, the maximum of exploration = 1.0, the minimum of exploration = 0.01, and the random epsilon decay = 0.995 are the same as with the previous DQN studies [10].

In Figure 3, we consider that Q_target for TD-learning is trained every episode, but G_target for MC-learning is trained in the end of whole episodes. The mixture update are accomplished based on the probabilities β_1 and β_2. Followed by the observables of the environment, β_1 and β_2 are able to be marginally interacted by system designers. For G_target, we should check the length of the whole episodes. Moreover, we suggest that when the roll-out is toward to the maximum length, training of the AI agent with MC-based will be finished earlier than the maximum length because the value function can be approximately close to the best prediction. Therefore, we do not need the whole episodes any more.



(그림 4) 결과 중 가장 최고의 경우
(Figure 4) The best case



(그림 5) 결과 중 가장 최악의 경우
(Figure 5) The worst case

Figure 4 and Figure 5 display most of the results for best and worst cases, respectively. The proposed algorithm, off-policy RL, DQN with MD and TD balance at random can yield better results than the class DQN using only TD-learning in most cases. In the best case, our proposed

algorithm reaches the maximum reward earlier than the classic DQN. However, in the worst case, our proposed algorithm is almost same with the class DQN. Therefore, we can attempt a different type of deep neural network (DNN) or hyper-parameter settings to demonstrate that our proposed model can enhance the exploration with only MC and TD balance at random. We are convinced that we can enhance the behavior policy. For the purpose, we have more runs. Table 1 shows that the quantitative comparison between our proposed algorithm and the class DQN using only TD-learning. Actually, most cases are similar. However, in therms of "in score 150", our proposed algorithm is better than the classic DQN using only TD-learning. After this we consider that the sample-efficient policy gradient such as DDPG would be possible to have better results. Our next step is about to combine MC and TD not only off-policy RL, DQN but also policy gradient methods, such as DDPG [6, 13, 14]. Moreover, we will attempt to exploit the different type of deep neural networks such as CNN. [14]

(표 1) 정량적 비교 결과
(Table 1) Quantitative Comparison

The Best Cases (in between 100&200)	The Average Cases (in between 300&500)	The worst Cases (Never Ending until MAX Score 500)
(a)20%, (b)20(%)	(a)79%, (b)79(%)	(a)1%, (b)1(%)
In 150 (a)60%, (b)40%		

(a) The proposed TD+MC based algorithm (b) The class DQN using only TD-learning

5. Conclusion

In this paper, we have suggested the technique to exploit a random balance with a mixture of TD and MC in off-policy RL, representative DQN. We demonstrate DQN with TD and MC balance at random, which is trained with a random probability β_1 for performing a gradient descent of TD and β_2 for performing a gradient descent of MC. We attempt to exploit a random balance with a mixture of TD and MC in RL without any complicated formulas for better exploration and easier deployment. We also demonstrate that a well-performed TD learning are also granted special favor of the mixture of TD and MC through an experiments in OpenAI Gym. Our proposed method goes through experimental comparison with the classic DQN using only TD-learning. The result shows that our proposed algorithm has shorter training time than the classic DQN using only TD-learning. We will attempt to exploit a random balance with a mixture of TD and MC in policy gradient of RL and the different type of deep neural network.

참고문헌(Reference)

[1] D. Silver, A. Huang, C. J. Maddison, A.Guez, L.t Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. Nature, Vol 529, No. 7587, pp. 484 - 489, 2016. https://doi.org/10.1038/nature16961

[2] R. S. Sutton, A. G. Barto. Reinforcement learning: An introduction, volume 1. MIT press Cambridge, 1998. https://doi.org/10.1016/S1364-6613(99)01331-5

[3] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." NIPS 2013. http://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

[4] A. Amiranashvili, A. Dosovitskiy, V. Koltun and T. Brox, TD OR NOT TD: Analyzing The Role Of Temporal Differencing In Deep Reinforcement Learning, ICLR 2018. http://arxiv.org/abs/1806.01175

[5] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, S. Levine, Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic, ICLR 2017. http://arxiv.org/abs/1611.02247

[6] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning, ICLR 2016. https://arxiv.org/abs/1509.02971

[7] V. Nair and G. E. Hinton, Rectified Linear Units Improve Restricted Boltzmann Machines, ICML 2010. https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf

[8] OpenAI Gym: https://gym.openai.com

[9] Cart-Pole-V0: https://github.com/openai/gym/wiki/Cart-Pole-v0

[10] Cart-Pole-DQN: https://github.com/rlcode/reinforcement-learning-kr/blob/master/2-cartpole/1-dqn/cartpole_dqn.py, 8 Jul. 2017.

[11] Tensorflow: https://github.com/tensorflow/tensorflow, 31 Oct. 2019.
[12] Keras : https://keras.io/api/ Oct. 2019.
[13] G. Sun, G. O. Boateng, H. Huang and W. Jiang, "A Reinforcement Learning Framework for Autonomous Cell Activation and Customized Energy-Efficient Resource Allocation in C-RANs," KSII Transactions on Internet and Information Systems, vol. 13, no. 8, pp. 3821-3841, 2019. https://doi.org/10.3837/tiis.2019.08.001
[14] R. Mu and X. Zeng, "A Review of Deep Learning Research," KSII Transactions on Internet and Information Systems, vol. 13, no. 4, pp. 1738-1764, 2019. https://doi.org/10.3837/tiis.2019.04.001

◐ 저 자 소 개 ◑

김 차 영(Chayoung Kim)
1996년 숙명여자대학교 전산학과(이학사)
1998년 숙명여자대학교 전산학과(이학석사)
2006년 고려대학교 컴퓨터학과(이학박사)
2005년~2008년 한국과학기술정보연구원 선임초청연구원
2008년~2017년 경기대학교, 컴퓨터과학과, 대우교수
2018년~현재 경기대학교, 융합교양대학, 조교수
관심분야 : 빅데이터, 머신러닝, 딥러닝 강화학습, IoT
E-mail : kimcha0@kgu.ac.kr

박 서 희(Seohee Park)
2017년 경기대학교 컴퓨터과학과 졸업(학사)
2018년 경기대학교 대학원 컴퓨터과학과 졸업(석사)
2018년 전자부품연구원(KETI) 휴먼케어시스템연구센터 위촉연구원
2018년 12월~현재 한국통신(KT) 융합기술원 AI연구소 전임연구원
관심분야 : 인간 자세 추정, 객체 탐지, 인공지능
E-mail : park.seohee@kt.com

이 우 식(Woosik Lee)
2003년 3월~2009년 2월 경기대학교 컴퓨터과학과 졸업(학사)
2009년 3월~2011년 2월 경기대학교 대학원 컴퓨터과학과 졸업(석사)
2011년 3월~2016년 2월 경기대학교 대학원 컴퓨터과학과 졸업(박사)
2016년 4월~2017년 2월 한국건설기술연구원 ICT융합연구소 박사후연구원
2017년 3월~2018년 4월 경기대학교 컴퓨터과학과 연구교수
2018년 4월~현재 사회보장정보원 사회보장데이터연구소 부연구위원
관심분야 : 빅데이터, 머신러닝, 데이터마이닝, 센서네트워크
E-mail : wslee@ssis.or.kr