SOLVING SPARSE REWARD SIGNAL PROBLEMS : DEEP Q-NETWORK-BASED APPROACH

Kangil Lee*, **Kunwoo Na***, **Dusol Lee***
Seoul National University
Seoul, Korea
{4bkang, jamongna, dusollee}@snu.ac.kr

ABSTRACT

Reinforcement learning problems with sparse reward signals are considered to be a challenging task. We solve two particular reinforcement learning problems with sparse reward signals by Deep Q-Network based approaches. We apply bootstrapped DQN and Random Network Distillation to balance exploration and exploitation.

***K*eywords** Exploration and Exploitation · Random network distillation · Prioritized replay buffer · Deep Q-network

1 Introduction

In reinforcement learning, an agent aims to maximize the cumulative reward. An agent needs to explore the unknown states and actions and choose the best-known actions. However, such best-known action might not be an optimal choice, i.e., the agent's action selection might be sub-optimal rather than optimal. In this sense, reinforcement learning problems with sparse reward signals are considered to be very challenging. The emergence of Deep Q-Network(DQN)[1] has lead to tremendous empirical success of some problems; however, there are specific problems where vanilla-DQN algorithm fails.
Thankfully, the various extent of DQN-based algorithms has emerged and achieved great empirical success. For example, bootstrapped-DQN[2], dueling-DQN[3] and other extents of DQN have provided efficient exploration schemes. Moreover, mathematical and statistical methods have been proposed to solve the optimal balance between exploration and exploitation. For instance, UCB-based approaches[4, 5, 6, 7] encourage the agent to explore and systematically reduce the uncertainty by introducing a bonus term which generally decreases as the timestep grows.
We solve two particular reinforcement learning problems, namely *chain-MDP* and *lava gridworld*, where the reward signal is sparse. We apply a DQN-based approach. Illustration of chain-MDP and lava gridworld is shown in Figure 1.

2 Preliminaries and Backgrounds

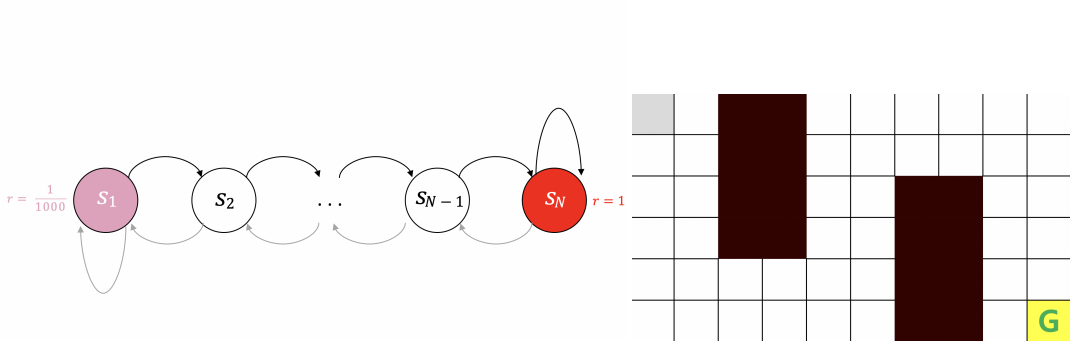
Deep Q-Networks(DQN). Deep Q-Network [1] is a function approximation based algorithm that approximates Q-value function by a neural network $Q(\cdot, \cdot\,; \theta)$ parametrized by $\theta \in \mathbb{R}^d$. The function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined as

$$Q(s,a) := \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a\Big]. \tag{1}$$

DQN applies TD(0)-error to train the RL agent, i.e., the optimization objective of DQN is

$$\underset{\theta \in \mathbb{R}^d}{\text{minimize}} \sum_{(s,a,r,s') \sim \mathcal{D}} \Big(r(s,a) + \gamma \max_{a'} Q(s', a'; \theta) - Q(s,a;\theta)\Big)^2, \tag{2}$$

*Equal contribution



(a) (b)

Figure 1: (a) Chain-MDP (b) Lava-gridworld

where $\mathcal{D}$ denotes the replay buffer. Due to this replay buffer, DQN was able to achieve a great sample efficiency. Double-DQN[8] modifies (2) as

$$\underset{\theta\in\mathbb{R}^d}{\text{minimize}} \sum_{(s,a,r,s')\sim\mathcal{D}} \big(r(s,a)+\gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta)\big)^2, \tag{3}$$

i.e., it uses two neural networks $Q(\cdot,\cdot\,;\theta^-)$ and $Q(\cdot,\cdot\,;\,\theta)$ where θ^- is a bootstrapped parameter of θ.

Bootstrapped DQN. Extending the heuristic of double-DQN which uses two bootstrapped parameters, bootstrapped-DQN[2] applies K-distinct neural networks that approximate Q-value. To be specific, if $K=1$, it is identical with vanilla-DQN, and if $K=2$, it is identical with double-DQN. In specific, bootstrapped-DQN applies

$$\mathbf{1}_k \odot \Big(r(s,a)+\gamma \max_{a'} Q\big(s', \operatorname{argmax} Q(s',a';\theta);\theta^-\big) - Q_k(s,a;\theta)\Big)\nabla_\theta Q_k(s,a;\theta), \quad (k=1,\cdots,K) \tag{4}$$

as a gradient of k-th neural network. Here, $\mathbf{1}_k$ denotes the mask tensor.

Prioritized Experience Replay(PER). Memory replay buffer in DQN is one of the key ingredient of great empirical success of DQN, since it dramatically enhances the sampling efficiency. Prioritized experience replay(PER)[9] is a replay buffer that is designed to sample past experience with high priority(e.g., TD(0)-error). To be concrete, it samples transition $\tau_i = (s_i, a_i, r_i, s_{i+1})$ with probability

$$\mathbb{P}\{\tau_i\} = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \quad (i=1,\cdots,k) \tag{5}$$

where p_k denotes the priority of k-th transition and $\alpha\in[0,1]$ is a hyperparameter. Prioritized sampling yields a bias which can encourage the RL agent to explore; however, in the long term, it is general to correct the sampling bias via multiplying importance sampling weight $w\in\mathbb{R}^N$ to the gradient. The sampling bias is corrected with

$$\theta \leftarrow \theta + \eta\cdot w \odot \nabla_\theta Q(s,a;\theta), \tag{6}$$

where η denotes the learning rate. Here, $w\in\mathbb{R}^N$ is calculated by

$$w_i = (N\cdot\mathbb{P}\{\tau_i\})^{-\beta}, \quad (i=1,\cdots,N) \tag{7}$$

where N denotes the batch size and $\beta\in[0,1]$ is a hyperparameter. If $\beta=0$, then the replay buffer does not corrects the sampling bias, and if $\beta=1$ then it fully corrects the sampling bias.

Random Network Distillation(RND). Random Network Distillation[7] is the first method that surpassed human performance on the infamous environment *Montezuma's Revenge*. RND can effectively predict the novelty of s_t, an observation at time step t, by MSE error between fixed target network f and prediction network $\hat{f}$,

$$i_t = \|\hat{f}(s_t;\theta) - f(s_t)\|^2. \tag{8}$$

The prediction network is trained with samples to minimize the expected MSE error. From a Bayesian point of view, the MSE error could be regarded as an estimate of predictive variance. Thus it can quantify the degree of uncertainty, i.e., it can be used as an intrinsic reward for promoting exploration. To be concrete, total reward r_t is computed by the sum of intrinsic reward i_t and external reward e_t given by the environment.

NoisyNet. Vanilla-DQN with an $\epsilon-$greedy policy tends to exhibit poor performance in environments where deep-exploration is required or high stochasticity involves. NoisyNet[10] injects the random noise(e.g., Gaussian random noise) to the Q-network, i.e., it modifies (3) as

$$\underset{\theta\in\mathbb{R}^d}{\text{minimize}} \sum_{(s,a,r,s')\sim\mathcal{D}} \left(r(s,a)+\gamma \max_{a'} Q(s',a',\varepsilon^-;\theta^-) - Q(s,a,\varepsilon;\theta)\right)^2, \tag{9}$$

where ε^- and ε are random noises. Due to the noise injection, agent well-explores the uncertain states and actions. Also, the effect of ϵ is learnable in the sense that the Q-network can learn $\theta \in \mathbb{R}^d$ that neutralize the effect of the random noise in the long term.

3 Results

3.1 Chain-MDP

Chain-MDP is an environment with discrete action space $\mathcal{A}$ ={left, right}. Agent receives $+1$ reward if it reaches the right end, $+0.001$ reward if it reaches the left end, and 0 reward elsewhere. We have applied various RL algorithms, including DQN-based approaches as well as policy gradient-based algorithm, to compare the performance. Figure2(a) demonstrates average cumulative rewards of distinct RL algorithms.

It turns out that DQN-based approach generally outperforms SAC-discrete[11], which is a policy gradient-based algorithm.[2] SAC-discrete tends to achieve maximum cumulative reward in early episodes; however, its performance fluctuates a lot. Vanilla-RND exhibits poor performance; however, together with bootstrapped DQN, it tends to perform well.

The most noticeable result in terms of performance improvement is that of bootstrapped-DQN. Agents equipped with bootstrapped-DQN generally perform very well, i.e., they converge to the maximum reward quickly and exhibit slight fluctuation after convergence. Most importantly, the performance of bootstrapped-DQN is robust toward random initialization of networks. Among K-different Q networks, at least one has "good" initialization in most cases, which leads to robustness.

In chain-MDP, deceptive small rewards force other methods without deep exploration to fail. For example, vanilla-DQN with an ϵ-greedy policy sometimes reaches the rightmost state and gets maximum reward quite quickly, but an ϵ-greedy policy hinders exploitation. Using a small ϵ value cannot be a solution as the agent never gets to the right end in almost every trial.

One notable result is that RND does not help in the chain-MDP environment. This result is surprising to some extent because count-based-UCB enhances the performance of an RL agent. We expect RND would be more effective if the state space gets larger. As discussed in [13], overfitting can hugely affect the performance of RL agents negatively, which we consider as the main reason for the poor performance of RND in the chain-MDP setting.

The performance gap between the algorithm using bootstrapped-DQN together with UCB and that using bootstrapped-DQN only is negligible. In order to build our model more computationally efficient and avoid using the tabular method, we select bootstrapped-DQN.

3.2 Lava-gridworld

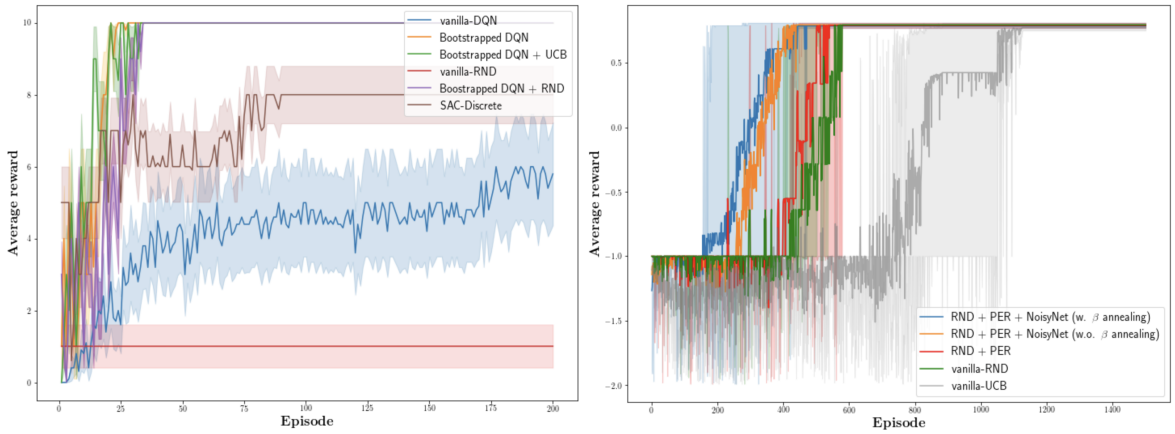
Lava-grdiworld is an environment with discrete action space $\mathcal{A}$ ={left, right, up, down}. Agent receives $+1$ reward if it reaches the goal state, -1 reward if it steps on the lava, and -0.01 otherwise. It turns out that SAC-discrete and vanilla-DQN tends to converge to the sub-optimal, i.e., the agent tends to receive -0.01 rewards successively by walking on the circles and not stepping on the lava. In order to achieve better exploration schemes, we have applied UCB-based algorithms. We did not apply policy gradient methods since those algorithms tend to converge easily to the sub-optimal. Figure 2(b) demonstrates the performance of various algorithms.

It turns out that RND-based approach outperforms the count-based-UCB. Also, it turns out that prioritized sampling using PER enhances the performance of RND, and if PER is combined with NoisyNet, it outperforms any other algorithms that we have experimented. The algorithm with β-annealing slightly outperforms the algorithm that uses constant $\beta = 0$. We select RND-based algorithm with PER with β-annealing and NoisyNet.[3] On the one hand, the prioritized sample selection could systematically encourage the deep exploration of the agent. On the other hand, it

[2] SAC(Soft-actor critic)[12] is originally designed for continuous action space. In [11], an extent of SAC algorithm that can be applied to discrete action space is proposed.

[3] For the specific β-annealing scheme and hyperparameter selection, see Appendix.

helps the agent minimize the intrinsic reward i_t of the state that has been observed, so that it can quickly find out the optimal policy. Our model selection is motivated by this heuristic.



(a) (b)

Figure 2: Performances of various algorithms in chain-MDP and lava-gridworld.

4 Conclusion and Further Works

We solved two particular reinforcement learning problems; chain-MDP and lava-gridworld, by using DQN-based approach. For chain-MDP, we observed that bootstrapped DQN with UCB algorithm outperforms SAC-discrete; however, we select bootstrapped DQN to lower the computational complexity. For lava-gridworld, we observed that prioritized replay buffer and NoisyNet could ennoble the performance of RND if hyperparameters and β-annealing scheme is well-tuned.

We have observed that poor hyperparameter tuning(e.g., $\beta = 1$ or poor annealing) abase the agent's performance. Since such tuning process is difficult, we look forward to design the algorithm that involves a neural network that takes bias correction weight $w \in \mathbb{R}^N$ or a minibatch of observations as an input and outputs an approximation of the weight vector. Also, we wish to design well-performing policy gradient method-based algorithms that hardly converge to the sub-optimal.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.

[3] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1995–2003, PMLR, 20–22 Jun 2016.

[4] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[5] J.-Y. Audibert, R. Munos, and C. Szepesvári, "Tuning bandit algorithms in stochastic environments," in *International conference on algorithmic learning theory*, pp. 150–165, Springer, 2007.

[6] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 2721–2730, JMLR.org, 2017.

[7] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[8] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[9] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[10] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, *et al.*, "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[11] P. Christodoulou, "Soft actor-critic for discrete action settings," *arXiv preprint arXiv:1910.07207*, 2019.

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870, PMLR, 10–15 Jul 2018.

[13] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," *arXiv preprint arXiv:1804.06893*, 2018.

Appendix

Here, we provide specific hyperparameter setting and annealing schemes.

For the chain-MDP, we used $K = 10$ as the number of ensemble networks. To clarify, we applied greedy action selection.

For the lava-gridworld environment, we select RND based approach with PER and NoisyNet. Details are shown below.

1. (*Selection of* α) Heuristically, an agent does not have a full knowledge on the environment, hence fully-greedy prioritization with $\alpha = 1$ might interrupt agent's deep search. Hence we use $\alpha = 0.5$, while other choice with $\alpha \in [0.4, 0.7]$ seems to perform well.
2. (β *annealing scheme*) As discussed in [9], sampling bias can encourage deep-exploration. In such sense, we use

$$\beta = 1 - \exp\Big(- \frac{\text{total step}}{1.3 \times 10^{10}} \Big),$$

   so that as total step increases, agent tends to exploit rather than to explore.
3. (*Initialization of NoisyNet)* We initialized each layer of NoisyNet with uniform distribution

$$\mathcal{U}\big[- \frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}} \big]$$

   where k denotes the number of inputs. Also, we initialized the entries of each layers with constant $\rho = \sigma/\sqrt{k}$ where σ is a hyperparameter. We used $\sigma = 0.5$. This setting is equivalent with that of [10]. Heuristically, $\sigma > 0$ might provide optimistic initialization which can encourage active searching of an agent, while large value of σ might increase the effect of random noise, which can cause a training inefficiency.