Final Project Report of Reinforcement Learning – An Empirically Study of Hierarchical Deep Reinforcement Learning

Qianli Shen
1500017740
Yuanpei College, Peking University

Mo Zhou
1500017814
Yuanpei College, Peking University

Abstract

Sparse reward is always a difficult problem for reinforcement learning. To solve this problem, researchers proposed several hierarchical reinforcement learning algorithms. With the help of deep neural networks, we give our hierarchical deep reinforcement learning algorithms h-DQN and h-DQNm to tackle the sparse reward problem. We empirically show that our method is more stable and faster than DQN in Discrete Stochastic Decision Process (DSDP) and Maze Game.

1 Introduction

Learning with sparse reward in a complex environment is very difficult for classical reinforcement learning. The main reason is that it requires the agent to explore the environment efficiently. However, there exists a kind of methods, called hierarchical reinforcement learning[1][5], could have better performance on reinforcement learning problems with sparse reward environment. In general, hierarchical reinforcement learning decomposes the task into several small but easy ones. By completing these subgoals, the agent could solve the original task. Hence, hierarchical reinforcement learning could learn the optimal policy in principle.

Recently, deep reinforcement learning, especially deep Q-Networks(DQN[7]), successfully learns to play Atari Games and Go. By using deep neural networks as nonlinear approximate function, deep reinforcement learning shows its power. But it still could not overcome the problem left over by classical reinforcement learning, like sparse reward problem.

In this project, we empirically study the performance of hierarchical deep reinforcement learning (h-DQN), which combines the hierarchical reinforcement learning and deep reinforcement learning. In this way, h-DQN could solve the sparse reward problem and benefits from the power of deep reinforcement learning. We also slightly modified h-DQN to get h-DQNm, which is more stable and faster in training. We measure the performances of our h-DQN and h-DQNm on Discrete Stochastic Decision Process (DSDP) and Maze Game. The experiment results show that both h-DQN and h-DQNm behave better than DQN, and h-DQNm converges faster than h-DQN in Maze Game.

1.1 Outline

In Section 2, we will clarify our notations throughout the report. We will give a brief summary for hierarchical reinforcement learning and deep reinforcement learning in Section 3 and 4. In Section 5, we will give our algorithms for hierarchical reinforcement learning. At last, the experiment results will be given at Section 6.

2 Preliminary

In this section, we will briefly introduce few notations that will be frequently used in the following sections. We consider an infinite-horizon discounted Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is a (finite) set of states, $\mathcal{A}$ is a set of actions (discrete or continuous action space), $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor.

In reinforcement learning, we aim to find a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ or a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ to maximize the expected discounted reward

$$\eta(\pi) = \mathbb{E}_{s \sim \rho_\pi, a \sim \pi} r(s, a)$$

At last, we denote the state-action value function Q and value function V as the following,

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots \sim \pi} \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k)$$

$$V^\pi(s_t) = \mathbb{E}_{a_t; s_{t+1}, \ldots \sim \pi} \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k)$$

3 Hierarchical Reinforcement Learning

For reinforcement learning, one of its most important challenges is the sparsity of the reward. All reinforcement learning methods require the environment to give a feedback about their actions. But if its feedback, or in other word, reward, is very sparse, it will be difficult for agent to sample useful experience for learning. This may lead to the failure of the reinforcement learning system. However, let us consider what human would do in this situation. Usually, people would like to decompose the original tasks into small but easy-achieved goals. We will finally finish the original task by completing every subgoals. Inspired by this simple idea, many researchers develop methods called hierarchical reinforcement learning.

For hierarchical reinforcement learning, one way is to consider goals directly. Schaul et al.[8] estimates the universal value function $V(s, g)$ to give the optimal action. Kulkarni et al.[5] proposed the following h-DQN framework. Consider a meta controller to select a proper goal according to the state and a controller to achieve the goal that meta controller selected. Let $\mathcal{G}$ denotes the goal spaces. Then, meta controller would be a MDP $(\mathcal{S}, \mathcal{G}, P_\mathcal{G}, F, \rho_0^\mathcal{G}, \gamma_\mathcal{G})$ and controller would be a MDP $(\mathcal{S} \times \mathcal{G}, \mathcal{A}, P_\mathcal{A}, F, \rho_0^\mathcal{A}, \gamma_\mathcal{A})$. By learning meta controller and controller together, we could have learn the optimal policy. Another way is to decompose the original task. The MAXQ framework[1] decompose the MDP into several small MDPs. Latent Space Policy Learning[2] solves the task layer by layer with the information of latent space. In the following section, we will use hierarchical reinforcement learning to improve the performance of deep reinforcement learning in environments with sparse rewards.

4 Deep Reinforcement Learning

With the huge success of deep learning in computer vision[3] and speech recognizing[4] in the recent years, researchers in the reinforcement learning community found out that neural networks improve the performance of reinforcement learning a lot. Combing the classical reinforcement learning and modern deep learning together, we could get what we called deep reinforcement learning. In fact, deep reinforment learning is a classical reinforcement learning using neural networks to approximate value function V, state-action function Q, or policy π.

Perhaps Tesauro et al.[13] is the first paper that attempt to combine neural networks and reinforcement learning, and defeated human experts. And recently, Mnih et al.[7] developed a method called Deep Q-Learning (DQN), which could let agent to learn playing Atari games from pixels and achieve average human level. This work indeed starts the recent works on deep reinforcement learning. Just like classical reinforcement learning, there are two main trends in deep reinforcement learning. The first is value-based algorithm. There are many works focus on learning state-action function Q, like DQN[7], Dueling-DQN[15], Double-DQN[14]. And second is policy-based method, which models the policy π directly. In this line of work, there are several practical algorithms like Trust Region Policy Optimization (TRPO)[10], Deterministic Policy Gradient (DPG)[12], Deep Deterministic Policy Gradient (DDPG)[6] and Proximal Policy Optimization (PPO)[11]. All these methods above show the strong ability of deep reinforcement learning. However, deep reinforcement learning still has the drawbacks of classical reinforcement learning, i.e. poor performance on sparse reward. We will later see that when combing hierarchical reinforcement learning and deep reinforcement learning, it will become better than before.

5 Hierarchical Deep Reinforcement Learning

In this section, we will introduce our Hierarchical Deep Reinforcement Learning (h-DQN, h-DQNm) algorithms. In fact, they are both modified versions of h-DQN in [5].

5.1 h-DQN and h-DQNm

Our h-DQN and h-DQNm contain two important parts, meta controller and controller. The meta controller tries to learn a MDP $(\mathcal{S}, \mathcal{G}, P_{\mathcal{G}}, F, \rho_0^{\mathcal{G}}, \gamma_{\mathcal{G}})$ while controller tries to learn another MDP $(\mathcal{S} \times \mathcal{G}, \mathcal{A}, P_{\mathcal{A}}, F, \rho_0^{\mathcal{A}}, \gamma_{\mathcal{A}})$.

The basic procedure for h-DQN and h-DQNm is the following: First, let meta controller select a goal and then train controller to complete the goal. With the goal completed, the meta controller will select another goal. The procedure loops until the agent reaches the terminal state.

By using the framework of hierarchical reinforcement learning and using deep neural networks to estimate state-action function, we could get our hierarchical deep reinforcement learning algorithms h-DQN and h-DQNm. In details, the controller estimates the following state-action function Q,

$$\begin{aligned} Q_c(\{s,g\},a) &= max_{\pi_c}\mathrm{E}[\sum_{k=t}^{\infty}\gamma_{\mathcal{A}}^{k-t}r_k|s_t=s;a_t=a;g_t=g,\pi_c] \\ &= max_{\pi_c}E[r_t+\gamma_{\mathcal{A}}max_{a_{t+1}}Q_c(\{s_{t+1},g\},a_{t+1})|s_t=s;a_t=a;g_t=g,\pi_c] \end{aligned} \tag{1}$$

Similarly, for the meta-controller, we have

$$Q_{mc}(s,g) = max_{\pi_{mc}}E[\sum_{k=t}^{t+N} f_k + \gamma_{\mathcal{G}} max_{sg'} Q_{mc}(s_{t+N},g')|s_t=s,g_t=g,\pi_{mc}], \tag{2}$$

where N is steps that controller complete the current goal g.

To use DQN to estimate the above Q functions, we define the following loss function,

$$L(\theta_c) = \mathrm{E}_{(\{s,g\},a,r,\{s',g\})}[r+\gamma_{\mathcal{A}}max_{a'}Q_c(\{s',g\},a';\tilde{\theta}_c) - Q_c(\{s,g\},a;\theta_c)]^2, \tag{3}$$

$$L(\theta_{mc}) = \mathrm{E}_{(s,g,r,s')}[r+\gamma_{\mathcal{G}}max_{g'}Q_{mc}(s',g';\tilde{\theta}_{mc}) - Q_{mc}(s,g;\theta_{mc})]^2, \tag{4}$$

where $\tilde{\theta}_{mc}$ and $\tilde{\theta}_c$ are the parameters of Q target networks of meta controller and controller. By minimizing the loss function above, our meta controller and controller could converge and give the optimal policy. Algorithm1 gives the detailed description of h-DQN and h-DQNm.

Reward Function For reward function of meta controller, or we called extrinsic reward, we simply use the environment reward. In fact, there could be more complex extrinsic reward function, which may contain more prior human knowledge. However, we would like our method to become more general. So we only use the original reward. For reward function of controller, or we called intrinsic reward, we could have different choices. One example is 0-1 reward. It means that the controller only get reward when it complete the goal. Another example is the distance reward. It means that the intrinsic reward is an non-increasing function f of the distance between the agent and the goal. For example, $f(x) = -x$ or $f(x) = 1/x$. Note that sometimes it could not easily get the distance, so we use the 0-1 reward function. We use Algorithm2 as our intrinsic reward function.

5.2 Differences Between original h-DQN, h-DQN and h-DQNm

The original h-DQN updates the meta controller along with the controller all the time. However, this may cause the unstable behavior of meta controller. Since the controller is not converged, the reward meta controller gets may not be true and confuse the meta controller. But if we separate the update step of meta controller and controller, and let meta controller start training at later time. In this way when meta controller starts learning, the controller has already learned something, which will help to stabilize the training procedure.

During the training procedure, the meta controller of h-DQN stores many useless transitions due to the controller's mistakes, just like what we have stated above. But for meta controller of h-DQNm, it could learn to choose the right goal based on the useful transitions. What makes a difference is the line 18 of Algorithm1. Meta controller of h-DQNm only stores the transition when the controller achieves the goal. This help the meta controller learn more efficiently. Hence, we could expect that h-DQNm converges faster than h-DQN.

Algorithm 1 Hierarchical Deep Reinforcement Learning (h-DQN/h-DQNm)

```
Require: controller start step $controller\ start$, meta controller start step $meta\ controller\ start$
    and $max\ episode$
Require: meta controller $DQN_{mc}$ and $controller\ DQN_c$.
 1: for i = 1 to $max\ episode$ do
 2:     $Step \leftarrow 0$
 3:     Initialize environment and get start state $s$
 4:     Obtain goal from meta controller: g $\leftarrow DQN_{mc}(s)$
 5:     while $s$ in not terminal state do
 6:         $F \leftarrow 0$; $s_0 \leftarrow s$
 7:         while $s$ in not terminal state and goal $g$ is not reached do
 8:             Obtain action from controller: $a \leftarrow DQN_c(\{s, g\})$
 9:             Execute $a$ and get next state $s'$ and extrinsic reward $f$ from environment
10:             Obtain intrinsic reward: $r \leftarrow$ IntrinsicReward($g$, $s'$)
11:             StoreTransition(($\{s, g\}$, $a$, $r$, $\{s', g\}$), $DQN_c$)
12:             if $Step$ > $controller\ start$ then
13:                 Update controller $DQN_c$
14:             end if
15:             $F \leftarrow F + f$; $s \leftarrow s'$
16:             $Step$ = $Step + 1$
17:         end while
18:         if goal $g$ is reached then                # the only difference between h-DQN and h-DQNm
19:             StoreTransition(($s_0$, $a$, $r$, $s$), $DQN_{mc}$)
20:         end if
21:         if $Step$ > $meta\ controller\ start$ then
22:             Update meta controller $DQN_{mc}$
23:         end if
24:         Obtain new goal from meta controller: $g \leftarrow DQN_{mc}(s)$
25:     end while
26: end for
```

6 Experiments

We experiment our algorithms on different tasks to measure their performances. The first problem is Discrete Stochastic Decision Process (DSDP) problem. And the second problem is a Maze game. In our experiments, we run each algorithm 5 times with different random seeds.

Algorithm 2 IntrinsicReward

```
Require: $g$, $s$
  if ReachedGoal($g$, $s$) then
    return 1
  else
    return -1
  end if
```

6.1 Discrete Stochastic Decision Process

6.1.1 Environment

Discrete Stochastic Decision Process (DSDP) is a stochastic decision process with 6 possible states and 2 actions. The agent always starts at s_2. When the agent chooses 'left' action, it moves left deterministically. However, the action 'right' could only succeed 50% of the time. Otherwise, the agent moves to the left. The reward at the terminal state s_1 depends on whether s_6 is visited ($r = 1$) or not ($r = 1/100$). This DSDP is originally used in [5] and we simply adopt it. The process is illustrated Figure1.

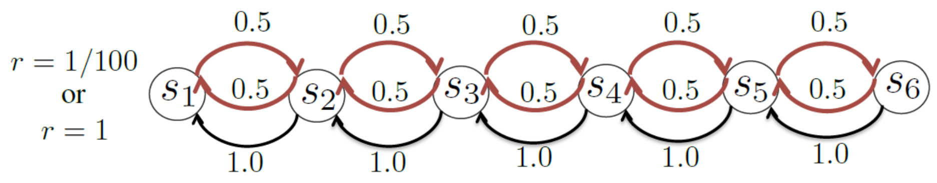

Figure 1: Discrete Stochastic Decision Process (DSDP)

6.1.2 Expected Reward of Optimal Policy

DSDP is a model that we could easily analysis mathematically. Here, we give the reward that a optimal agent could get in expectation. Through in the experiment, we stop the game after 100 steps if the agent does not arrive the terminal state s_1, we still consider an infinite-horizon MDP to simplify our analysis.

The optimal policy is obvious. If the agent does not arrive at s_6 before, choose 'right' action. If agent arrived at s_6, choose 'left' action. Let r_{ij} denotes the expected reward for the optimal agent when starting at state s_i (i=1,2,...,6). Note that $j = 0$ means the agent does not arrive at s_6 before and $j = 1$ means the agent arrived at s_6 before. Thus, We could have the following equations,

$$\begin{cases} r_1 = 0.01 \\ r_{20} = 0.5r_1 + 0.5r_{30} \\ r_{30} = 0.5r_{20} + 0.5r_{40} \\ r_{40} = 0.5r_{30} + 0.5r_{50} \\ r_{50} = 0.5r_{40} + 0.5r_6 \\ r_6 = r_{51} = r_{41} = r_{31} = r_{21} = 1.00 \end{cases} \tag{5}$$

Solving the above equations, we get $r_{20} = 0.204$. It means that even if the agent learns an optimal policy, it could only get an average reward 0.204. We will later see that our h-DQN almost achieves this expected reward 0.204, while DQN not.

6.1.3 Results

For DQN, we used the standard structure, which is one layer fully connected neural network, and standard hyper parameters with prioritized experience replay[9]. For h-DQN and h-DQNm, we use almost the same structure as original DQN. The only difference is we let controller and meta

Table 1: Hyper Parameters for h-DQN and DQN in DSDP Task

	controller	meta controller	DQN
n goals	6	NA	NA
max episode	20000	20000	20000
learn start	5000	10000	5000
optimizer	rmsprop	rmsprop	rmsprop
momentum	0.9	0.9	0.9
learning rate	0.00025	0.00025	0.00025
learning rate decay	0.99	0.99	0.99
γ	0.99	0.99	0.99
ϵ	1	1	1
ϵ_{max}	0.01	0.01	0.01
ϵ iters	5000	5000	5000
replace target iters	200	200	200
memory size	10000	10000	10000
prioritized replay alpha	0.6	0.6	0.6
prioritized replay beta0	0.4	0.4	0.4
prioritized replay beta iters	100000	100000	100000
prioritized replay eps	1e-6	1e-6	1e-6

controller start their learning at different time. All the hyper parameters are shown in Table1. Note that the hyper parameters for h-DQN and h-DQNm are the same.

The results showing in the Figure2 are average rewards, and the dash region shows its standard error. Due to the randomness of the environment, our testing rewards are also averaged over five different runs. We could see that there exits huge difference between the performance of h-DQN(m) and DQN. It seems that DQN is only able to get the minimum reward 0.01 for the most of the time, while h-DQN(m) indeed learns a better policy. As the previous section shows, the expected reward of optimal policy is 0.204. Our h-DQN and h-DQNm almost achieve this averaged reward. Hence, we could say that our h-DQN and h-DQNm actually learned the optimal policy, while DQN failed on this task. Comparing h-DQN and h-DQNm, we could see that h-DQNm is slightly stable and converged slightly more quickly than h-DQN, even though it is not a significant difference.

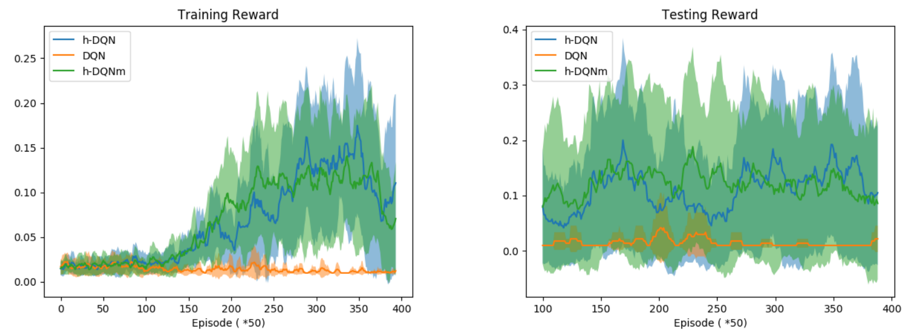

Figure 2: DSDP Results

6.2 Maze Game

6.2.1 Environment

Maze Game is shown in Figure3. The agent is represented by a red rectangle. The yellow circle is the key, the green rectangle is the door and the black rectangle is the hell. The agent's goal is to get the key, then open the door. The door could only be opened when the agent has the key. In the meantime, the agent could not step on the hell. Whenever the agent steps on the hell or opens the door, the game is over. The environment will return a 1 point reward when the agent gets the key and extra 2 points when agent successfully opens the door. However, the agent will receive -1 reward when it steps on the hell. Hence, for the optimal policy, we could expect it always get the reward 3.

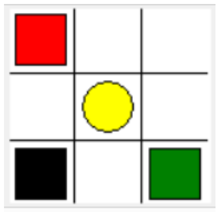

Figure 3: Maze Environment

6.2.2 Results

For DQN, h-DQN and h-DQNm, we all use the same structure as we use in the DSDP problem. The only difference is their hyper parameters. For detailed parameters, please see Table2. Note that the hyper parameters for h-DQN and h-DQNm are the same.

The results are shown in Figure4. We could have several observations about experiment results. First, for the final performance, we could see that h-DQN and h-DQNm are better than DQN. All h-DQN methods are able to get the averaged reward over 2.5 in both training and testing, while DQN are only able to get averaged reward 1. It means that DQN only success to learn to pick up the key and avoid the hell, but fails to learn to open the door. In fact, we could find that there exists a period of time that DQN's averaged reward achieved 2, but then went down. The main reason is that DQN could not remember the order to pick up the key and open the door due to the later inefficent sampling. However, with the help of our hierarchical reinforcement learning frameworks, both h-DQN and h-DQNm successfully learn the optimal policy and never forget. Second, for the convergence rate, we could see that h-DQNm converges to the optimal policy faster than h-DQN. This phenomenon confirms our analysis for the difference between h-DQN and h-DQNm in Section 5.

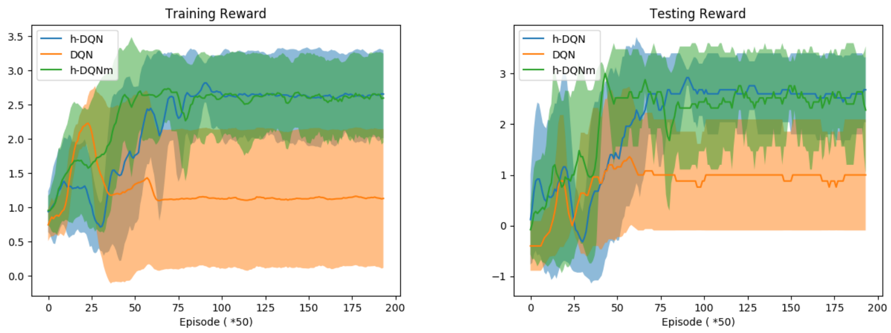

Figure 4: Maze Results

7 Conclusion

In this report, we empirically study the performances of h-DQN and h-DQNm on DSDP Problem and Maze Game. We find out that h-DQN benefits not only from the expressive power of DQN, but also from the general framework of hierarchical reinforcement learning. By changing the update frequency and transition store rule, we get a more stable and faster framework, h-DQNm.

For future work, we would like to apply our algorithms to more complex problems, like Montezuma's Revenge. Also, we would like to design better reward functions for controller and meta controller. In fact, during the training, we found that our model is sensitive to hyper parameters to some extent. Hence , it would be interesting to see if we could develop more robust models for hierarchical deep reinforcement learning.

References

[1] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[2] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[6] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[8] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[10] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[12] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[13] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[14] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[15] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

8 Appendix

Table 2: Hyper Parameters for h-DQN and DQN in Maze Game

	controller	meta controller	DQN
n goals	3	NA	NA
max episode	10000	10000	10000
learn start	200	20000	5000
optimizer	rmsprop	rmsprop	rmsprop
momentum	0.9	0.9	0.9
learning rate	0.001	0.001	0.00025
lr decay	0.99	0.99	0.99
γ	0.99	0.99	0.99
ϵ	1	1	1
ϵ_{max}	0.01	0.01	0.01
ϵ iters	1000	1000	5000
replace target iters	200	200	200
memory size	5000	500	10000
prioritized replay alpha	NA	0.6	0.6
prioritized replay beta0	NA	0.4	0.4
prioritized replay beta iters	NA	1000	5000
prioritized replay eps	NA	1e-6	1e-6