Artificial Intelligence Project – Final Report

Yiheng Lin, Zhihao Jiang

1 Introduction and Background

The foundation of h-DQN should be deep Q-learning method proposed by the authors of [**?**]. They proposed two important methods to fix the instability when applying deep learning to reinforcement learning, which is memory replay and fixing the parameters of the target network. Their works make it possible to apply the deep Q-learning to solve problems like playing simple games. But the limitation that hinders deep Q-learning to be applied in the real world is that the rewards are often too sparse for the agent to learn good policies efficiently.

The intuition of h-DQN (Hierarchy Deep Q-Learning) may come from the thought that a big goal can be divided into a sequence of small goals, which are much easier to reach for the agent than the big goal. By reaching a small goal at a time, the agent may be able to handle the situations in which rewards are sparse. So the authors of [**?**] proposed a structure as following: a meta-controller (the higher hierarchy) is responsible for choosing a goal for the controller (the lower hierarchy) given the sequence of steps and states the controller experiences and the extrinsic reward they get during the process of trying to reach the subgoal; a controller, who tries to reach the goals set by the meta-controller, takes actual actions in the environment and receiving the intrinsic reward designed based on the goal.

Despite tackling the efficiency problem when rewards are sparse, another two advantages of h-DQN are: it can reveal a hidden state that is crucial for deciding the final reward; it can also encourage the agent to explore unknown states more actively. To make it clear, consider the following game proposed in [**?**]:

1.1 Experiment Game Setting

The state set is $S = \{1, 2, \cdots, 6\}$ and the action set is $A = \{1, 2\}$. The agent starts at state 2 and state 1 is the terminal state. At state i, if the agent takes action 1, it will go to state $i - 1$ with probability 1; if the agent takes action 2,

it will go to state $i+1$ with probability 0.5;otherwise, it will go to state $i-1$. The reward is received at state 1: if the agent has been to state 6, it will receive reward 1; otherwise, it will receive a reward 0.01.

During the presentation, our mentor points out that this is not an MDP (Markov Decision Process) since the reward depends on the whole history rather than the current state and action. But actually this implies a common case we meet in the real situation, that is, we may fail to add some crucial hidden states that decide the final reward to the state space. As we mentioned in the progress report, by adding an additional dimension to the state space which indicates whether we have already visited state 6, the problem will become much easier. But without this additional state, the problem is almost intractable for traditional Q-learning and DQN which tries to approximate the Q function. However, h-DQN, in which the controller is given both state and goal, intrinsically has more dimensions in new state space than the original state space. This may help h-DQN to perform better than traditional Q-learning in this problem.

Another important function of subgoals are encouraging exploration. Notice that in our problem's setting, random actions can hardly reach state 6. By setting a remote state as a goal, the agent is pushed to explore that state rather than try aimlessly and give up.

1.2 Setting Goal and Intrinsic Reward

The authors of [**?**] proposed a way to set goals and intrinsic rewards. In their method, each goal is set to a specific state and intrinsic reward is the opposite number of the l2-norm of the difference of current state and goal state (i.e. $r(s_t, g_t, a_t, s_{t+1}) = -||s_t + g_t - s_{t+1}||$, here $s_t + g_t$ is the goal observation and s_{t+1} is the current observation). As they mentioned, this method works will in the ant grid game setting and can be easilly generallized to tackle other game settings.

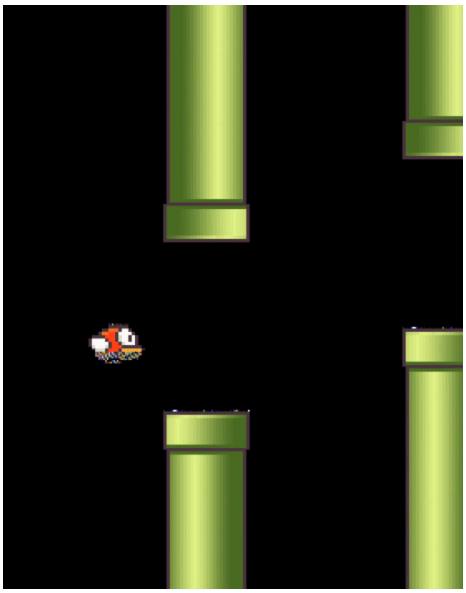
In our experiment, we use the same method to set goals and intrinsic rewards (that is, approaching the goal incurs higher intrinsic rewards). Although our mentor said after our presentation that the distance between two states cannot be easily found, we think the Lp-norm of the difference between the goal and the current state is at least a good heuristic of the real distance (in action steps). And we find this method can significantly accelerate the training process by encouraging the controller (the lower hierarchy) to achieve the goal and give more accurate feedback to the meta-controller (the higher hierarchy).

1.3 Limitations of the previous works

We have carefully studied the h-DQN algorithm proposed in [**?**] and reimplement their method based on [**?**] to solve the problem we mentioned in subsection 1. A significant weakness of this method is the performance is too volatile and seems very hard to converge. And as we observed, in the actual training process, the learning of meta-controller and controller did not follow the authors' expectation. When the subgoal is set to state 2, the controller (the lower hierarchy) opts to choose action 2 frequently even when it is at state 3, 4, 5 and consequently arrives at state 6 with high probability. This encourages the meta-controller (the higher hierarchy) to set the subgoal to state 2 since the meta-controller will see setting subgoal 2 often results in a good extrinsic reward. The reason may be that the experience provided to the meta-controller is highly unstable during the training. The controller may have learned a suboptimal policy to reach its subgoal, but this suboptimal policy results in a good extrinsic reward, which is favored by the meta-controller. When we see good average rewards, the model may have not converged. And maybe at some time the controller may find a new way to reach its subgoal, then the average reward collapses dramatically, as we see when running experiment.

To choose goal space, the authors of [**?**] said we can find (manually or by some digital image processing skill) some things that are very possible to be important for this game. They argued that labeling objects in a given image (for example, find the key, ladders etc. in ATARI game) is an open problem. So we do not consider how to find the goal set when studying h-DQN. And the authors of [**?**], claims that letting the goal states as the state space can more easily be generallized. However, this method might only be useful in games that have a very good and thorough abstraction. If the agent can only discover a small part of the whole environment, we cannnot easily apply their method.

And we find this weakness of generalization when we try to apply the h-DQN method to learn a control policy in the flappy bird game. We first do the experiment using DQN method based on the code on [**?**].



Although we can observe the improvement through time, the agent learns very slow. This may because the (extrinsic) reward is given only when the bird has successfully passed through a door. At many times, although the bird has reached the height that can pass the door or very close to the door, it crashes into the wall due to some small disturb. Then the agent cannot distinguish this case with the totally bad policy like always flying upward because it does not receive any additional reward for getting more closer to the door. If we use the h-DQN method, we can set a goal in the door and give intrinsic to the controller for getting closer to this goal. Then this may speed up the training process. But since the scene is changing and moving forward, we cannot fix a location or a specific image as the goal for controller. Even reading the current height of the bird requires a new neural network and additional training, and requires specific human knowledge of the rules of this flappy bird game.

2 Our Result

2.1 First Experiment

We implement our first experiment on the 6-state problem we mentioned in the Section 1.1. In this game setting, we have the following theorem:

Theorem 1. *If the optimal policy is chosen (i.e. keep taking action 2 until state 6 is arrived), the probability of arriving at state 6 is* $\frac{1}{5}$.

Proof. Since we only consider whether state 6 can be reached, in the proof, we can let state 6 be a terminal state. Let x_i be the probability of arriving state 6

from state i ($1 \leq i \leq 6$), then we have $x_1 = 0, x_6 = 1$, and

$$x_i = 0.5x_{i-1} + 0.5x_{i+1}, (2 \leq i \leq 5) \tag{1}$$

Solving the equations and we can get $x_2 = \frac{1}{5}$.

Notice that if we adopt $\epsilon-$greedy policy, equation (1) will become

$$x_i = \frac{1+\epsilon}{2}x_{i-1} + \frac{1-\epsilon}{2}x_{i+1}, (2 \leq i \leq 5) \tag{2}$$

When $\epsilon = 0.1$, we can get $x_2 = 0.128645$. □

This means if the probability of reaching state 6 gets near 0.2, the policy is near optimal.

2.1.1 Intuition

Draw subgoal randomly is a new method to set subgoal. Because it urge the agent to explore new states rather than stay in local optimal states. In the game we just mentioned, the agent tends to go left if using $\epsilon-$greedy because of the transport probability. However, in hierarchical model, the agent tends to the subgoal. If we draw subgoal randomly, the agent will explore efficiently.

Based on this method, we attempt to modify the method a little bit.

To explore new states, besides draw subgoal randomly, we can also draw the state which we have least information about.

There are several ways to implement this idea such as drawing the farthest state away the current state, drawing randomly from states which are not arrived before. Another idea is, define the distance between two states in some way, and draw a subgoal such that the subgoal is "close" to lots of states which are not arrived before. We will attempt to implement this idea in models which is more complicate than the model we just mentioned.

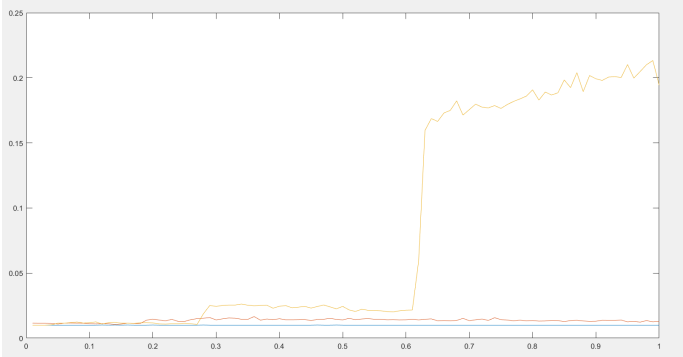
2.1.2 Our Attempt

Although the idea is not implemented in general model, it is easy to implement the idea in the simple model we mentioned. That is, set the farthest state as the subgoal.

In detail, the agent start at state 2. We always set subgoal as the farthest state (state 6 initially). The subgoal is not changed until the agent arrive the state. Once the agent arrive the subgoal, we reset the subgoal as the farthest state away the current state. For the lower controller, the closer to the subgoal, the larger the reward is, so the agent tends to the subgoal.

The chart below is the experiment result. The $x-$axis is time, and the $y-$axis is grade. The blue, red, yellow lines is corresponding to the method

$\epsilon-$greedy, random subgoal, farthest subgoal respectively.



We can see the yellow line raises abruptly when the process is 60% done. Maybe this is because the parameter should be adjusted, or there are some other reasons.

For more details, see our codes, there is README on the web site.

2.2 Second Experiment

To make our idea more general, we design a new 2-dimension continuous game setting as following: The map size is 20 by 20; The start state is at $(2, 2)$, the terminate state is at $(0, 0)$; The action set is $\{0(up), 1(down), 2(left), 3(right)\}$. Choose each action will result in moving to the corresponding direction for a random distance in the interval $[0.8, 1.2]$ (if hitting the wall, the agent will stop at the wall). The reward R is given at the terminal state. Let 'key' be at $(15, 15)$.

$$R = \begin{cases} 100 & \text{if state 'key' has been visited} \\ 1 & \text{otherwise} \end{cases}$$

In this continuous setting, we define a location $L = (x_0, y_0)$ has been visited if the agent has visited a point $P = (x, y)$ such that $||P - L||_2 < 1$.

Our implementation works as following: The controller (the lower hierarchy) will be given a subgoal state g. Suppose at time t, the goal is set at g_t, the intrinsic reward for the controller will be $r(s_t, g_t, a_t, s_{t+1}) = -||g_t - s_{t+1}||_2$. To learn which action to make, we use Q-Learning with Linear Approxiamators [**?**]. The meta-controller (the higher hierarchy) learns how to choose a goal for the controller based on current state and whether the key has been fetched. Here we add an additional state to indicate whether the key has been fetched because the meta-controller should be able to distinguish this state by using cameras and images. This skill will require training deep neural network for image classification, but we do not concern about the problem here. The meta-controller use traditional Q-Learning. But we will restrict its action zone to be within a range of $(5, 5)$, i.e., it will set a goal $g'_t = (i, j), -2 \leq i \leq 2, -2 \leq j \leq$

$2, i, j \in Z$ when a new goal is required. So the goal will be the $g_t = s_t + g'_t$. We use this new goal setting method instead of setting the furthest state because in reality, the agent can only observe the environment within a range.

To test the function of linear approximator of controller, we first set the goal to key position. After the key state is reached, we set the key to the terminal state. The agent learns to achieve an average reward of near 100 just after about 100 iterations. This proves the h-DQN's ability on encouraging the agent to search the given goal state much more efficiently than random walk in $\epsilon-$greedy method.

Then we use the Q-learning method we mentioned before for controller to test if it can figure out the key position. However, the agent failed to learn the optimal policy in 9999 iterations. The possible reason may be the iteration number is not large enough.

For more details, see our codes, there is README on the web site.

3 Conclusion and Further Research

In this project, we successfully test the h-DQN's ability on encouraging the agent to search far states. This propoerty is very useful when the rewards are sparce and requires efficient searching to find.

However, the ability of h-DQN method is still limited in the sense of generalization and requires specifically design for specific game settings. The distance of two states, although easy to figure out in some game settings, are not always easy to calculate in the real world problems. So maybe we should train a neural network to give intrinsic reward based on current state and goal to generalize the method in order to help in all problems DQN can solve.

But converging speed is another big burden of h-DQN, as we find in reimplement of [**?**]'s work. The unaccuracy of the lower hierarchy experience make it more hard to train the higher hierarchy. This problem will be more critical if we add more flexibility for giving intrinsic reward. The solution may need to find a balance between efficiency and generalization power.