# Progress Part 1

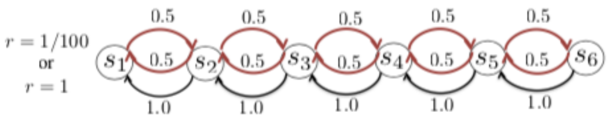
Yiheng Lin, Zhihao Jiang

## Introduction

It is widely accepted that h-DQN (Hierarchy Deep Q-Learning) is developed to tackle the efficiency problem of $\epsilon-$greedy Q-Learning when the reward is sparce. However, after we study the problem setting and reimplement the method proposed in (ref 1), we find another big advantage of h-DQN over traditional Q-Learning and DQN (Deep Q-Learning) (ref 2) is that h-DQN can potentially discover the hidden state which is not observable by the agent but is crucial for receiving reward. To make our idea more clear, let us consider the following game (proposed in (1)):

### Game Setting

The state set is $S = \{1, 2, \cdots, 6\}$ and the action set is $A = \{1, 2\}$. The agent start 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 reward 0.01.




The authors of (ref 1) claim that traditional Q-Learning cannot learn to take action 2 in order to move towards state 6 even after a long period of training (200 epochs) and converges to a sub-optimal policy. Our experiment shows the same result.

Actually, despite the length of the path of the optimal policy and the probability setting on state transition, we think the major difficulty in this game is the hidden state of whether the agent has reached state 6. At state i ($2 \leq i \leq 5$), when state 6 has not been reached, the agent should take action 2 in order to reach 6, which means $Q(i, 2) > Q(i, 1)$; when state 6 has already been reached, the agent should take action 1, which means $Q(i, 2) < Q(i, 1)$ (Although keep taking action 2 is a possible optimal policy, we think it is almost impossible for the agent to keep taking action 2 after state 6 has been reached, since it cannot see the difference in the final reward.). In traditional Q-Learning and DQN, QValue function should converge as learning proceed, thus it is difficult for them to learn the optimal policy even unlimited time is given.

However, if the hidden state is given to tell the agent whether it has reached state 6, (for example, 0 represents it has not visited state 6, 1 represents it has visited state 6) then the problem is converted to a MDP of 12 states. Then at state i, the agent can seperately learn $Q((i, 0), 2) > Q((i, 0), 1)$ and $Q((i, 1), 2) < Q((i, 1), 1)$. We think this will make the problem much easier for traditional Q-Learning. We will do more experiments to verify this claim. The subgoal in (ref 1) operates similarly, the meta-controller should first set the subgoal to state 6 and then set the subgoal to state 1 after state 6 is visited.

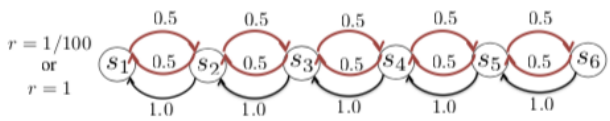
In the real world, it is very common that the final outcome of a game actually depends on some hidden state that the agent can not observe. So this problem, although not very complicated, can provide important insight to solve real problems with hidden states. h-DQN algorithm provides an potentially efficient way to let the agent, which combines controller and meta-controller, learn about the important hidden states.

## H-DQN Algorithm

We carefully studied algorithm 1 proposed by (ref 1) (please find the pseudocode in appendix 1) and reimplement this algorithm on the game setting described in the last section based on (ref 3, github). We can see salient improvement compared with traditional Q-Learning method, although the performance is worse than the authors of (ref 1) claimed. After supervising the training process, we think there are some potential improvements we can do to judge the effectiveness of h-DQN algorithm.

Roughly speaking, the h-DQN algorothm works as following: the meta-controller (higher hierarchy) sets a subgoal state for controller (lower hierarchy); then the controller tries to move to the subgoal state since it will get an intrinsic reward of 1 if it succeed in reaching the subgoal and no intrinsic reward if it fails to reach the subgoal; the meta-controller collects the extrinsic reward gotten by the agent for different (state, subgoal) pairs and decide which state should be the next subgoal.

As for the performance metric, the authors of (ref 1) uses the number of visits to each state during 1000 runs to judge the agents tendency of taking action 2. However, a lot of visits to state 6 is resulted by keep taking action 2 after arriving at state 6. So the agents can visit state 6 multiple times in one run. Thus we think we should instead evaluate the probability the agent arrives at state 6 in each run.




**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 $(1 \le i \le 6)$, then we have $x_1 = 0, x_6 = 1$, and

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

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

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

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

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

This theorem shows that the probability the agent arrives at state 6 in each run should be near 0.128 if the policy is near optimal when $\epsilon = 0.1$.

Another observation is that when the subgoal is set to state 2, the controller (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 (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. This result is unexpected, and not robust, since the intrinsic reward is given to the controller only when it arrives at state 2, we can never promise that the agent will always try to go to state 6 before arrives at state 2 during long period of learning. We think this phenomena occurs because the experience provided to the meta-controller is highly unstable during the training. The controller may has 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 been 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. So we think in the further experiments, we should measure the meta-controller's ability of setting reasonable subgoals; and the actions the controller takes to achieve them.

# Appendix

**Algorithm 1** Learning algorithm for h-DQN

Initialize experience replay memories $\{\mathcal{D}_1, \mathcal{D}_2\}$ and parameters $\{\theta_1, \theta_2\}$ for the controller and meta-controller respectively.
Initialize exploration probability $\epsilon_{1,g} = 1$ for the controller for all goals $g$ and $\epsilon_2 = 1$ for the meta-controller.
**for** $i = 1, num\_episodes$ **do**
  Initialize game and get start state description $s$
  $g \leftarrow \text{EPSGREEDY}(s, \mathcal{G}, \epsilon_2, Q_2)$
  **while** $s$ is **not** terminal **do**
    $F \leftarrow 0$
    $s_0 \leftarrow s$
    **while** **not** ($s$ is terminal **or** goal $g$ reached) **do**
      $a \leftarrow \text{EPSGREEDY}(\{s, g\}, \mathcal{A}, \epsilon_{1,g}, Q_1)$
      Execute $a$ and obtain next state $s'$ and extrinsic reward $f$ from environment
      Obtain intrinsic reward $r(s, a, s')$ from internal critic
      Store transition $(\{s, g\}, a, r, \{s', g\})$ in $\mathcal{D}_1$
      $\text{UPDATEPARAMS}(\mathcal{L}_1(\theta_{1,i}), \mathcal{D}_1)$
      $\text{UPDATEPARAMS}(\mathcal{L}_2(\theta_{2,i}), \mathcal{D}_2)$
      $F \leftarrow F + f$
      $s \leftarrow s'$
    **end while**
    Store transition $(s_0, g, F, s')$ in $\mathcal{D}_2$
    **if** $s$ is **not** terminal **then**
      $g \leftarrow \text{EPSGREEDY}(s, \mathcal{G}, \epsilon_2, Q_2)$
    **end if**
  **end while**
  Anneal $\epsilon_2$ and adaptively anneal $\epsilon_{1,g}$ using average success rate of reaching goal $g$.
**end for**

**Acknowledgement:**