# A research report on the frontier work of Hierarchical Reinforcement Learning

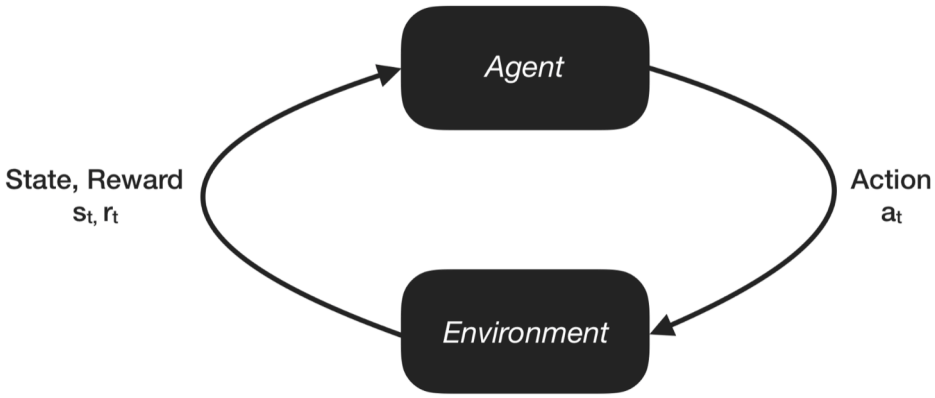
18340146 Computer Science and Technology Yujie Song

## Abstract

Hierarchical reinforcement learning (HRL) is a promising method that extends traditional reinforcement learning (RL) methods to solve more complex tasks. As for the research of hierarchical reinforcement learning, there are many papers presented at top conferences every year. In this report, I will research several of the top conference papers in recent years to understand the cutting-edge work done by them. In addition, I will give priority to my own understanding and retelling, supplemented by the interpretation of the papers, and make a research report on these works.

## 1 background

In recent years, reinforcement learning (RL) methods have yielded good results in many fields, such as learning to play Atari games from pixels. The most basic reinforcement learning model is as follows:

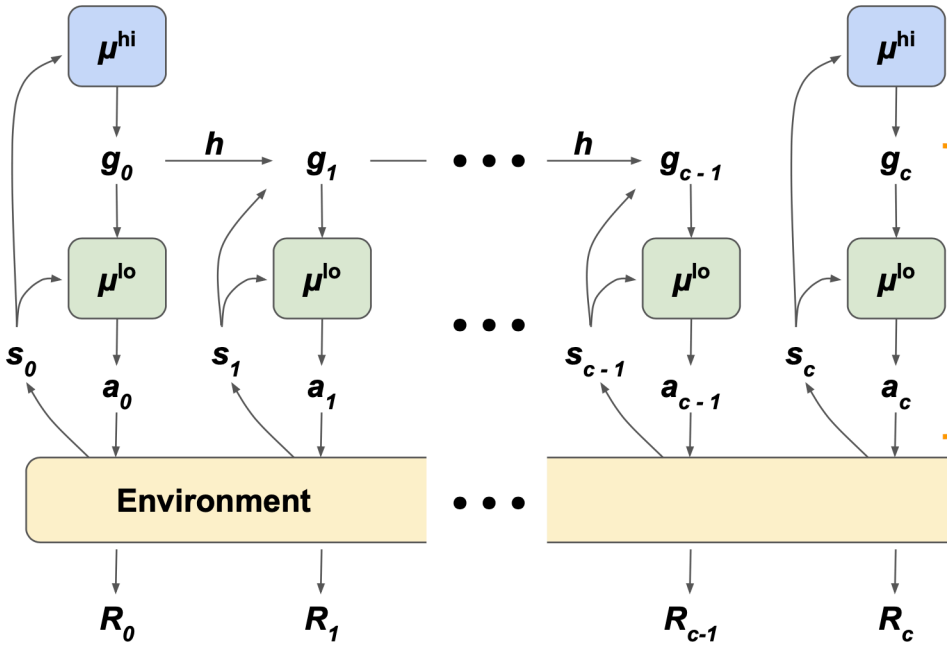



There is an agent and an environment. The agent makes an action $a_t$ through the current state $s_t$, and the environment gives the next state $s_{t+1}$ and the reward $r_t$ from this action. The above process is repeated, and the agent learns from these sample tuples $(s_t, a_t, r_t, s_{t+1})$ through some algorithms (such as Q-learning).

However, the basic RL approach does not apply perfectly to all problems, first of whom is the problem of having a large action or state space. The basic method creates a dimensional disaster. The next is the multi-time problem: one time period of the game has a goal corresponding to one time period, which the basic method is difficult to accommodate.

Hierarchical reinforcement learning (HRL), in which multiple layers of policies are trained to perform decision-making and control at successively higher levels of temporal and behavioral abstraction, has long held the promise to learn such difficult tasks. Hierarchy mainly solves the problem of sparse reward. In the actual reinforcement learning problem, the environmental reward is usually very sparse, coupled with the huge combination of state space and action space, leading to the direct training is often unable. The idea of hierarchy comes from the fact that we humans tend to solve a complex problem by breaking it down into several subproblems that are easy to solve.

The basic framework of hierarchical reinforcement learning is as follows:




Each time $t_0$ the higher-level policy $\mu^{hi}$ proposes a target $g_0$ based on the current state $s_0$, The lower-level policy $\mu^{lo}$ outputs an action $a_0$ according to the current state $s_0$ and target $g_0$, and obtains an internal incentive $R_0$ and the next state $s_1$, in which the reward $R_0$ is proportional to the degree of proximity to the target state, and the closer to the target, the greater the reward. At the next time $t_1$, the target transfer function $h$ will output the target $g_1$, and the lower-level policy will output the action $a_1$ according to the state $s_1$ and target $g_1$. The higher-level policy reproposes a target every $c$ time steps.

# 2 h-DQN

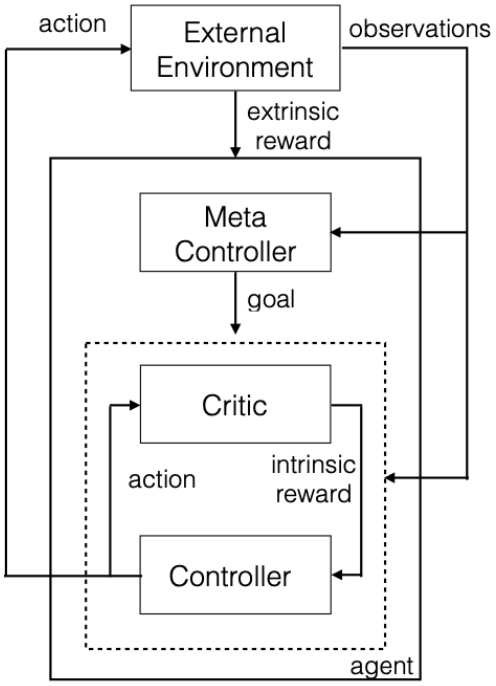
The first frontier paper I researched is Hierarchical Deep Reinforcement Learning : Integrating Temporal Abstraction andIntrinsic Motivation, published in 2016

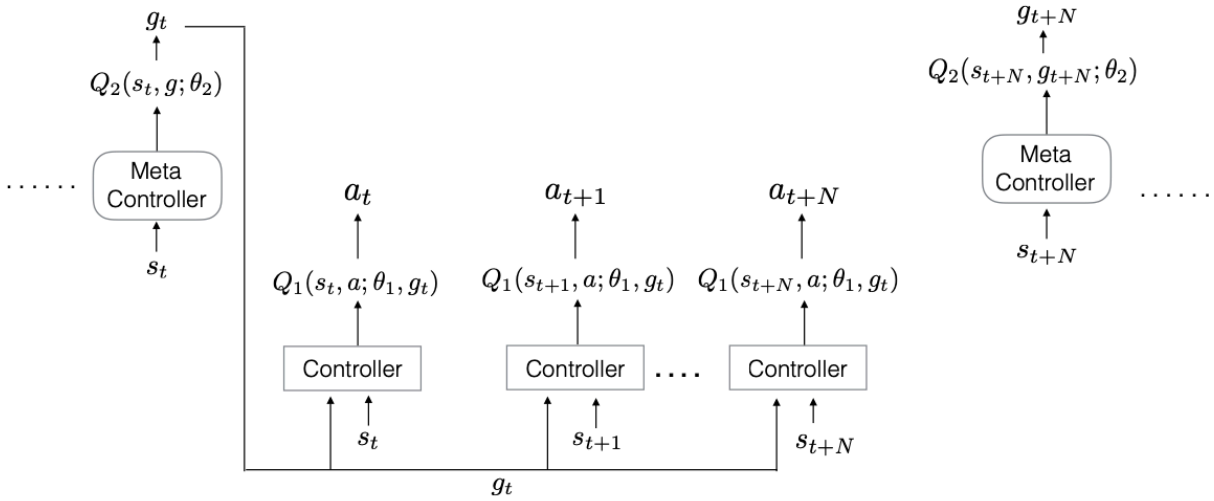
## 2.1 H-DQN's model

In the famous Atari game "Montezuma's Revenge", the player has to control the little man to get the key and then open the door to success. Therefore, in this game, there is clearly a multi-objective situation: to success you need to go to the gate, but to open the gate you need to get the key first. In this paper, the author mainly compares DQN with his own method, and points out the

shortcomings of DQN: "Deep Q-Networks and its variants have been successfully applied to various domains including Atari games and Go, but still perform poorly on environments with sparse, delayed reward signals."

Therefore, the author proposed a framework with hierarchically organized deep reinforcement learning modules working at different time-scales. The model is divided into two hierarchies:




(a) The higher-level module (Meta-Controller) receives the status and selects a new goal: It is responsible for obtaining the current status $s_t$, and then selecting a sub-task $g_t$ from a list of possible sub-tasks to assign to the lower-level module (Controller) to complete.

(b) The lower-level module (Controller) uses the state and selected goal to select the action until the goal is reached or the event ends: It is responsible for receiving the subtask $g_t$ of the higher-level and the current state $s_t$, and then selecting one or a series of possible actions $a_t$~$a_{t+N}$ to execute until the goal or end state is reached. Internal Critic gives the Controller a positive reward if and only if the goal is met.

The Meta-Controller then selects another target and repeats the steps (A~B).

## 2.2 H-DQN's training

This paper uses the framework of DQN to learn the strategy of Meta-Controller and Controller. Specifically, the Q value function of Controller is as follows:

$$\begin{aligned} Q_1^*(s,a;g) &= \max_{\pi_{ag}} \mathrm{E}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \mid s_t = s, a_t = a, g_t = g, \pi_{ag}] \\ &= \max_{\pi_{ag}} \mathrm{E}[r_t + \gamma \max_{a_{t+1}} Q_1^*(s_{t+1}, a_{t+1}; g) \mid s_t = s, a_t = a, g_t = g, \pi_{ag}] \end{aligned} \tag{1}$$

where $g$ is the agent's goal in state $s$ and $\pi_{ag} = P(a|s,g)$ is the action policy.

Similarly, for the Meta-controller, it has:

$$Q_2^*(s,g) = \max_{\pi_g} \mathrm{E}[\sum_{t'=t}^{t+N} f_{t'} + \gamma \max_{g'} Q_2^*(s_{t+N}, g') \mid s_t = s, g_t = g, \pi_g] \tag{2}$$

where $N$ denotes the number of time steps until the controller halts given the current goal, $g'$ is the agent's goal in state $s_{t+N}$ , and $\pi_g = P(g|s)$ is the policy over goals.

And then is to define $Q_1, Q_2$'s loss function: This paper stored $Q_1$'s experience $(s_t, a_t, g_t, r_t, s_{t+1})$ and $Q_2$'s experience $(s_t, g_t, f_t, s_{t+N})$ respectively in two experience pool $D_1$ and $D_2$, and then define the loss function:

$$L_1(\theta_{1,i}) = E_{(s,a,g,r,s') \sim D_1}[(y_{1,i} - Q_1(s,a,\theta_{1,i},g))^2] \tag{3}$$

$$L_2(\theta_{2,i}) = E_{(s,g,f,s') \sim D_2}[(y_{2,i} - Q_2(s,\theta_{2,i},g))^2] \tag{4}$$

where $i$ denotes the training iteration number and $y_{1,i} = r + \gamma \max_{a'} Q_1(s', a'; \theta_{1,i-1}, g)$, $y_{2,i} = r + \gamma \max_{a'} Q_2(s'; \theta_{2,i-1}, g)$

In particular, this paper does not directly give the calculation formula of $L_2(\theta_{2,i})$. I have written down the formula based on my own understanding, which may be wrong.

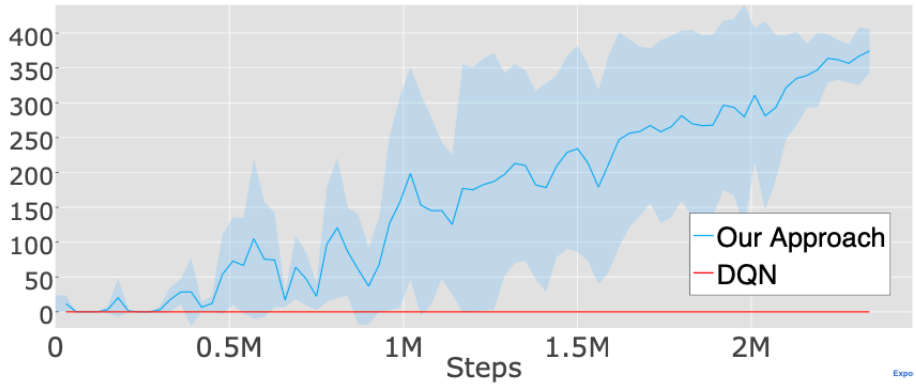
Then, the stochastic gradient descent method common to neural networks is used to train and optimize the parameters of DQN.

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

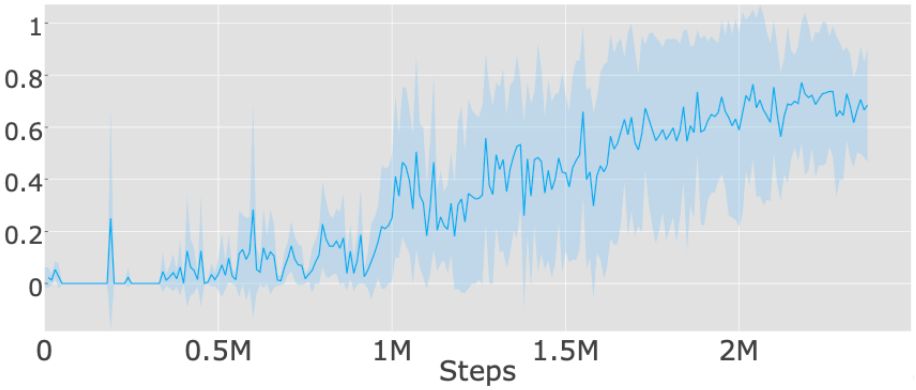
Initialize experience replay memories $\{\mathcal{D}_1, \mathcal{D}_2\}$ and parameters $\{\theta_1, \theta_2\}$ for the controller and meta-controller respectively.
Initialize exploration probability $\epsilon_{1,g} = 1$ for the controller for all goals $g$ and $\epsilon_2 = 1$ for the meta-controller.
**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**

## 2.3 h-DQN's effect and disadvantage

**Effect:** h-DQN was tested in the Atari game "Montezuma's Revenge". As can be seen from the figure below, h-DQN can achieve higher scores in continuous iterative training (while normal DQN has no effect), and the success rate of reaching the goal 'Key' is gradually increasing.




(a) Total extrinsic reward




(b) Success ratio for reaching the goal 'key'

**Disadvantage:** The paper artificially sets certain situations. For example, in the Atari game "Montezuma's Revenge", the paper artificially sets the Critic, which defines a yes-or-no condition like "whether the little man has reached a certain position". This makes the model dependent on artificial knowledge, which isn't universally applicable for other tasks.

At the same time, since this paper was published in 2016, DQN was basically the only frontier research at that time, so this paper could only compare it. And then, after more experts developed better hierarchical reinforcement learning models in the next few years, H-DQN seems a little more simple.

# 3 FuN

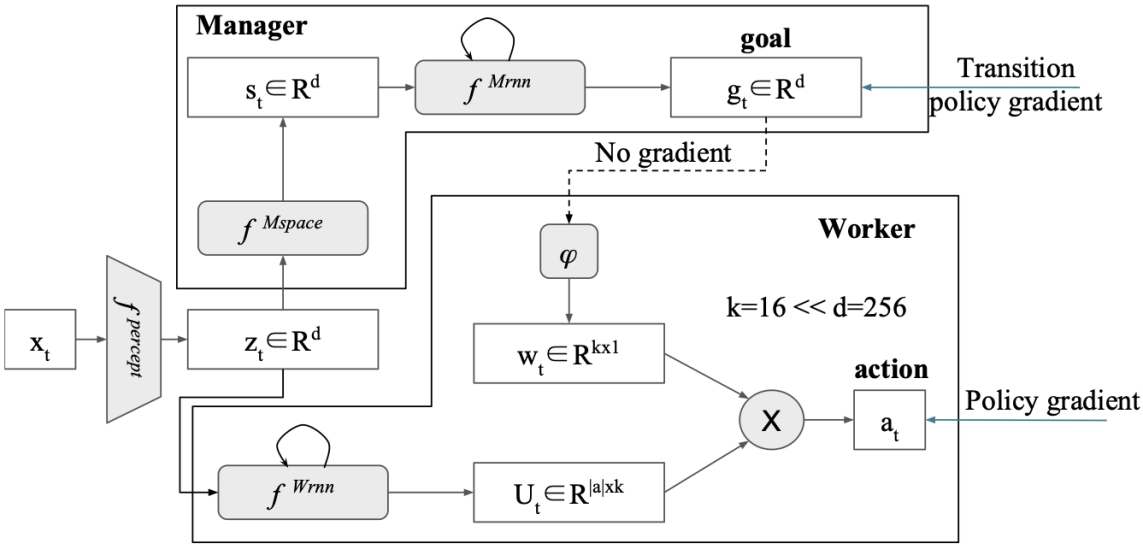
The second frontier paper I researched is FeUdal Networks for Hierarchical Reinforcement Learning, published in 2017

## 3.1 FuN's model

The model FuN is based on a 1993 paper Feudal Reinforcement Learning. The idea of this paper comes from the feudal hierarchy, whose control level is divided into three levels: The current level is the Manager, the upper level is the Super-Manager, and the lower level is the Sub-Manager, which belongs to the typical multi-level control. This paper also puts forward two principles: (1) reward hiding. It is rewarded as long as the Manager is satisfied, no matter whether the Super-Manager is satisfied or not; (2) Information hiding. The Sub-Manager does not need to know the goal set by the Super-Manager, and the Super-Manager does not need to know what the Manager

does.

The significance of the above paper is that it proposes the idea of multi-level task division, and the main work of FuN is to apply it to hierarchical reinforcement learning. FuN is also composed of two modules - the Worker and the Manager. The Manager internally computes a potential state $s_t$ and outputs a goal vector $g_t$. Worker generates actions based on external observations, its own state, and the Manager's goals. The specific model structure is as follows:




The model first converts the input $x_t$ from a high-dimensional image to a low-dimensional vector $z_t$ via Convolutional Neural Network ($f^{percept}$). On the one hand, $z_t$ is passed into the Manager module and becomes the Manager's state space $s_t$ through a full connection layer ($f^{Mspace}$), then through the Dilated LSTM carefully designed by the author, the goal $g_t$ set by the Manager layer for the Worker layer is obtained, and finally through the linear transformation $\phi$ nearly $c$ step, the goal $w_t$ is passed to the Worker layer. On the other hand, $z_t$ is passed into the Worker module, and through a common LSTM, the transformation matrix $U_t$ (i.e. the coefficient matrix of the final neural network $X$) of the Worker layer is obtained. Finally, it is combined with the target $w_t$, and the final action $a_t$ is obtained through the final neural network $X$.

$$z_t = f^{percept}(x_t) \tag{5}$$
$$s_t = f^{Mspace}(z_t) \tag{6}$$
$$h_t^M, \hat{g}_t = f^{Mrnn}(s_t, h_{t-1}^M);\ g_t = \hat{g}_t / ||\hat{g}_t||; \tag{7}$$
$$w_t = \phi(\sum_{i=t-c}^{t} g_i) \tag{8}$$
$$h_W, U_t = f^{Wrnn}(z_t, h_{t-1}^W) \tag{9}$$
$$a_t = SoftMax(U_t w_t) \tag{10}$$

## 3.2 FuN's training

For hierarchical reinforcement learning, the paper also adopts the method of separate training for the Manager layer and the Worker layer. For the Manager layer, the first thing to do is to give the action (goal) that the Manager outputs a definite meaning, rather than making it just an implicit variable. "We propose instead to independently train Manager to predict advantageous directions (transitions) in state space and to intrinsically reward the Worker to follow these directions."

The method to train the Manager layer is the transition policy gradient designed in this paper. In this method, each continuous $c$ step is regarded as one step of the Manager, and the state change generated by Worker's continuous $c$ step is regarded as one transition, i.e:

$$\pi^{TP}(s_{t+c}|s_t) = p(s_{t+c}|s_t, \mu(s_t, \theta)) \tag{11}$$

The corresponding strategy gradient formula is

$$\nabla_\theta \pi_t^{TP} = E[(R_t - V(s_t))\nabla_\theta \log p(s_{t+c}|s_t, \mu(s_t, \theta))] \tag{12}$$

Because $p(s_{t+c}|s_t, g_t) \propto \exp d_{\cos}(s_{t+c} - s_t, g_t)$, so the above equation can be converted to

$$\nabla g_t = A_t^M \nabla_\theta d_{\cos}(s_{t+c} - s_t, g_t(\theta)) \tag{13}$$

Where $A_t^M = R_t - V_t^M(x_t, \theta)$ is the Manager's advantage function, and $d_{\cos}$ is the cosine similarity. Then we can use strategy gradient method to train the Manager. This paper use A3C method.

**Algorithm S3** Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

*// Assume global shared parameter vectors $\theta$ and $\theta_v$ and global shared counter $T = 0$*
*// Assume thread-specific parameter vectors $\theta'$ and $\theta'_v$*
Initialize thread step counter $t \leftarrow 1$
**repeat**
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state $s_t$
  **repeat**
    Perform $a_t$ according to policy $\pi(a_t|s_t;\theta')$
    Receive reward $r_t$ and new state $s_{t+1}$
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  **until** terminal $s_t$ **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  **for** $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i;\theta'_v))^2 / \partial \theta'_v$
  **end for**
  Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
**until** $T > T_{max}$

For the Worker layer, its reward is defined firstly: the goal set by the Manager needs to be completed by the Worker, so the Worker needs to be motivated not only by external reward, but also by internal reward, that is, $R_t + \alpha R_t^I$. The definition of internal rewards needs to be related to the goals of the Manager, and the definition of the paper is

$$r_t^I = 1/c \sum_{i=1}^{c} d_{\cos}(s_t - s_{t-i}, g_{t-i}) \tag{14}$$

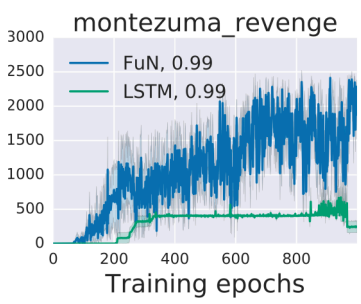
The corresponding strategy gradient formula is

$$\nabla_{\pi_t} = A_t^D \nabla_\theta \log \pi(a_t|x_t;\theta) \tag{15}$$

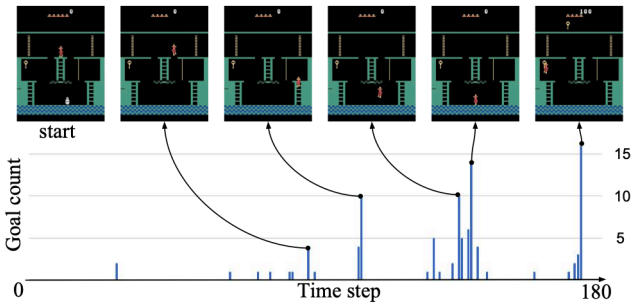
Where $A_t^D = (R_t + \alpha R_t^I - V_t^D(x_t;\theta))$ is the Worker's advantage function. Next, the Worker can be trained with the strategy gradient method, and the A3C method is also used in this paper.

### 3.3 FuN's effect and disadvantage

**Effect:** The goal of its experiments is to demonstrate that FuN learns non-trivial, helpful, and interpretable sub-policies and sub-goals. At the same time, the paper also conducted experimental comparison with the baseline LSTM model, and the results are as follows:




(a)




(b)

An experiment with Montezuma's Revenge is shown above. FuN and baseline LSTM are shown at left; The Goal count in the right figure indicates how many previous moments of this frame of image are considered as the Goal, that is, the game situation corresponding to such a high count position can be understood as the situation that the Manager expects to achieve. As you can see, these frames are basically the critical path to the key.

**Disadvantage:** For some specific games (such as Space_Invaders, Hero, Seaquest), Fun performs little or worse than the benchmark LSTM. This suggests that in these games long-term credit assignment is not important and the agent is better off optimising more immediate rewards in a greedy fashion.

## 4 HIRO

The third frontier paper I researched is Data-Efficient Hierarchical Reinforcement Learning, published in 2018

### 4.1 HIRO's model

Hierarchical Reinforcement Learning (HRL) method is an extension of traditional Reinforcement Learning (RL) method. Since then, however, there have been three important difficulties： How should one train the lower-level policy to induce semantically distinct behavior? How should the high-level policy actions be defined? How should the multiple policies be trained without incurring an inordinate amount of experience collection?

Yet, the majority of current HRL methods require careful task-specific design and on-policy training, making them difficult to apply in real-world scenarios. Therefore, this paper proposes a new pattern: HIRO, a method for applying off-policy algorithms to HRL frameworks, enabling HRL to be generic, requiring no artificial onerous assumptions, data duplication, and efficient utilization. So the key part of this paper is using off-policy correction in the higher-level policy training process.

In the previous HRL framework, the higher-level policy could only be used with on-policy, and the use of off-policy would cause instability of the policy, which was caused by the repeated use of historical empirical data: From the perspective of the lower-level policy $\mu^{hi}$, under the state $s_t$, take the action $a_t$, and the reward is $\sum R_{t:t+c-1}$ and the next state is $s_{t+c}$, which is obtained after $\mu^{lo}$ interacts with the environment by $c$ time steps. So it's related to the action $a$ output from $\mu^{lo}$. And then, For $\mu^{hi}$, $\mu^{lo}$ is a part of the environment, because it will affect $\mu^{hi}$'s next state $s_{t+c}$.

However, $\mu^{lo}$ is constantly updated, so under the same state $s_t$, $\mu^{lo}$ proposes the same target $g_t$, the reward $\sum R_{t:t+c-1}$ and the next state $s_{t+c}$ are constantly changing, not regular, completely random. Therefore, if only through ordinary off-policy, it may not learn any experience at all.

Therefore, this paper proposes off-policy correction: since $\mu^{hi}$ proposes a target $g_t$ under $s_t$, due to the constant updating of $\mu^{lo}$, the next state is no longer transferred to $s_{t+c}$, which becomes irregular. If $\mu^{hi}$ proposes another target $\tilde{g}_t$, which makes the current $\mu^{lo}$ take the same action $a$ as when it was not updated before, then in the $c$ steps, the lower-level policy $\mu^{lo}$ stays the same with the environment, so that $s_{t+c}$ will not change.

Therefore, the next step is to find the $\tilde{g}_t$. The method of this paper is: take $s_{t+c} - s_t$ as gaussian sampling center, take 8 samples, then add the sampling center $s_{t+c} - s_t$ and the original $\mu^{lo}$'s $g_t$ when not updated, a total of 10 reference values.

This paper choose $\tilde{g}_t$ to make the probability $\mu^{lo}_{new}(a_{t:t+c-1}|s_{t:t+c-1}, \tilde{g}_{t:t+c-1})$ bigger, so substitute these 10 reference values into the following formula for calculation:

$$\log \mu^{lo}(a_{t:t+c-1}|s_{t:t+c-1}, \tilde{g}_{t:t+c-1}) \propto -\frac{1}{2}\sum_{i=t}^{t+c-1} ||a_i - \mu^{lo}(s_i, \tilde{g}_i)||_2^2 + \text{const} \qquad (16)$$

Take the reference value with the maximum value of the above equation as $\tilde{g}_t$.

## 4.2 HIRO's training

This paper takes off-policy correction, so it also takes the corresponding TD3 (Twin of Deep Deterministic policy Gradient) method, which this paper hasn't introduce. So I consulted the introduction of various blogs on the Internet, and about my understanding on this.

The TD3 method is based on DDPG (Deep Deterministic Policy Gradient). Firstly, DQN use neural network instead of Q form, and the loss function is the gap between the current output of the neural network and the target, then take the derivative of the loss function to update the network parameters. The target is evaluated as $r + \gamma \max_{a'} \hat{q}(s', a', w)$, that is, from the next state, the Max function is used to select the largest action Q. Obviously, the Max operation cannot handle the case of continuous action, so DDPG took another neural network to replace $\text{Max}_{a'} \hat{q}(s', a', w)$.

DDPG takes the Actor-Critic method to update the parameters of the neural network, and at the same time, it adds the experience replay and double network structure of DQN. Since DDPG adds noise $\epsilon$, the maximum value estimate of the erroneous action value is usually larger than the true value in the real situation. In the TD3 algorithm, it uses two Critic networks to evaluate the Q values, and then selects the Q values of the smaller network to update, thus mitigating the Q overestimation. Meanwhile, Delayed Policy Update and Target Policy Smoothing Regularization scheme are adopted to optimize the results of TD3.

**Algorithm 1** TD3

Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\phi$ with random parameters $\theta_1$, $\theta_2$, $\phi$
Initialize target networks $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
**for** $t = 1$ **to** $T$ **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
  Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$

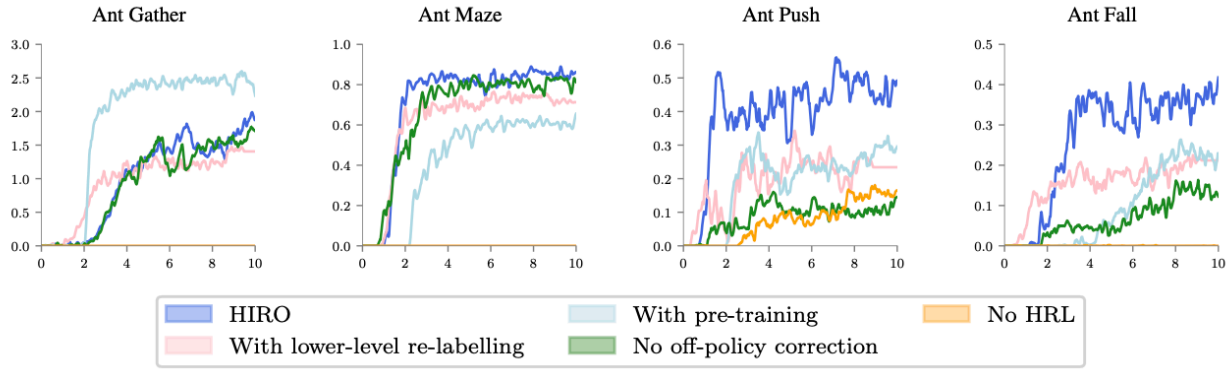
  Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  **if** $t$ mod $d$ **then**
    Update $\phi$ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i' \leftarrow \tau\theta_i + (1 - \tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  **end if**
**end for**

### 4.3 HIRO's effect and disadvantage

**Effect:** In this paper, they tested HIRO and other hierarchical reinforcement learning methods, such as Feudal Network and VIME, in Ant Gather, Ant Maze, Ant Push and Ant Fall environments. Finally they obtained that HIRO algorithm with far better effect than other algorithms.

| | **Ant Gather** | **Ant Maze** | **Ant Push** | **Ant Fall** |
|---|---|---|---|---|
| HIRO | **3.02±1.49** | **0.99±0.01** | **0.92±0.04** | **0.66±0.07** |
| FuN representation | $0.03 \pm 0.01$ | $0.0 \pm 0.0$ | $0.0 \pm 0.0$ | $0.0 \pm 0.0$ |
| FuN transition PG | $0.41 \pm 0.06$ | $0.0 \pm 0.0$ | $0.56 \pm 0.39$ | $0.01 \pm 0.02$ |
| FuN cos similarity | $0.85 \pm 1.17$ | $0.16 \pm 0.33$ | $0.06 \pm 0.17$ | $0.07 \pm 0.22$ |
| FuN | $0.01 \pm 0.01$ | $0.0 \pm 0.0$ | $0.0 \pm 0.0$ | $0.0 \pm 0.0$ |
| SNN4HRL | $1.92 \pm 0.52$ | $0.0 \pm 0.0$ | $0.02 \pm 0.01$ | $0.0 \pm 0.0$ |
| VIME | $1.42 \pm 0.90$ | $0.0 \pm 0.0$ | $0.02 \pm 0.02$ | $0.0 \pm 0.0$ |

Secondly, the paper also tested basic HIRO and its variants (Ablative Analysis) in several environments to understand the importance of various designs of the HIRO model:




**Disadvantage:** Insufficient support of HIRO on lower-level re-labelling and pre-training: Mentioned in other papers, the lower-level re-labelling technology allows the lower-level policy using experience, collected on specific goals for learning about the behavior of any alternative. But when joining in the experiments in this paper, although early training learning speed is significantly fast, its performance stable soon, the end result is not as good as basic HIRO. This paper has not done a further study on this issue but simply said "The benefit of re-labeling goals will require more research, and we encourage future work to investigate better ways to harness its benefits".

For pre-training, this is a method to avoid non-stationary problems caused by off-policy in high-level policy training. After adding this method, it performs better in simple scenes such as Ant Gather, but produces negative gains in difficult scenes. The paper explains it as: "This suggests that our off-policy correction is still not perfect, and there is potentially significant benefit to be obtained by improving it."

# 5 HAC

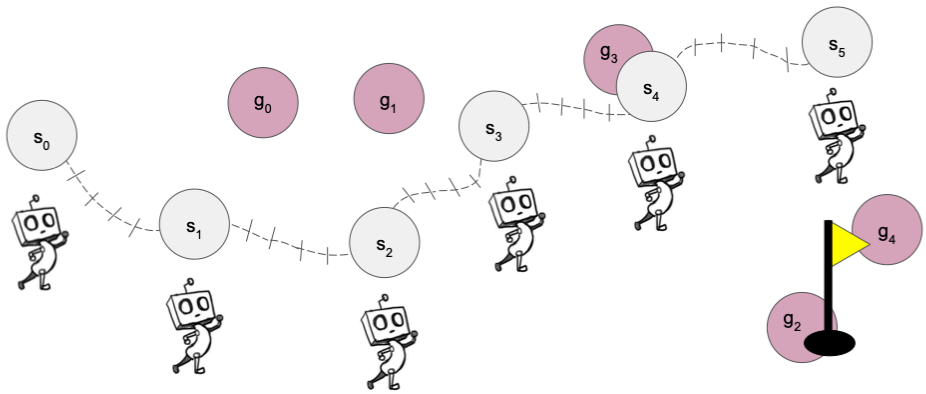
The last frontier paper I researched is Learning Multi-Level Hierarchies with Hindsight, published in 2019

## 5.1 HAC's principle

In the process of hierarchical reinforcement learning, there is a problem: if strategies at a certain level change in the hierarchical structure, it may lead to changes in the transition and reward part of strategies at a higher level, which is called non-stationary phenomenon. There are two main reasons why non-stationary occurs in hierarchical reinforcement learning. First, the state of the agent in $H$ time steps depends on the strategy of the lower layer. The upper layer proposes the same sub-goal in different stages and may reach different states eventually, resulting in that only the lowest strategy is stable. Second, the transition and reward functions at the higher level may change whenever the lower level's policy uses exploratory policies to complete subgoals. This makes it difficult to learn multiple levels of strategy together.

For this problem, the paper proposed a new model called Hierarchical Actor-Critic (HAC). The basic structure of HAC is three layers, The higher level reads the environment state $s_t$ and puts forward the sub-goal $g_t$ every $H$ time steps. The lower level receives the goal and then carries out $H$ step operation. Each step the lower level outputs an action $a_{t+i}, i \in [0, H)$, and the environment will change to $s_{t+i+1}$. After the sub-goal is reached or the $H$ steps is executed, the

higher level continues to propose new sub-goals... And what makes HAC so special is three transitions: Hindsight action transitions, Hindsight goal transitions and Subgoal testing transitions.




Hindsight action transitions: Simulates the transitions used by the higher level strategy when the lower layer uses the optimal strategy. There are two key components: The first part is using the sub-goal states realized after the fact as actions in Transition, rather than the sub-goal states originally proposed. The second key component is the reward function used at all sub-goal levels. The first requirement of this reward function is that it should motivate the shortest path to achieve the goal, because the shorter path allows faster learning. The second requirement is that it should be independent of the lower level paths. An example is given in the paper: the initial state of the agent is $s_0$, and the top-level strategy proposes the sub-goal $g_0$, and reaches the state $s_1$ after executing $H = 5$ steps. At this time, Highsight action transitions of the higher level strategy is [initial state=s0, action=s1, reward=-1, next state=s1, goal=yellow flag, discount rate=gamma]. Similarly, transitions of the second time step of the higher level strategy is [initial state=s1, action=s2, reward=-1, next state=s2, goal=yellow flag, discount rate=gamma].

Hindsight goal transitions: To complement transitions on all layers except the top layer, making each layer can study more sparse reward (the sparse reward) tasks effectively, and the method is to use Hindsight Experience Replay. Consider the example in the paper. Suppose the lower level is executing $H = 5$ step and producing 5 transitions, which are [initial state=s0, action=joint torques, reward=-1, next state=first tick mark, goal=g0, discount rate=gamma], ... , [initial state=4th tick mark, action=joint torques, reward=-1, next state=s1, goal=g0, discount rate=gamma]. Then use HER to modify the goal to a random next state (e.g. s1) in 5 periods, i.e. [initial state=s0, action=joint torques, reward=-1, next state=first tick mark, goal=s1, discount rate=gamma], ... , [Initial state=4th tick mark, action=joint torques, reward=0, next state=s1, goal=s1, discount rate=gamma] (Note that the reward is also modified to 0 if and only if next state = goal).

Subgoal testing transitions: In Hindsight action transitions, it is assumed that lower level has reached the optimal level, so the higher level will only generate sub-goals close to the goal by action transition alone. However, these sub-goals maybe far away from the initial position, so we assume that lower level can be reached, but in fact they cannot be reached. Therefore, in order for the higher level to generate realistic sub-goals, we need to check from time to time whether this sub-goal is reachable, and if not, change its Q value. For example, [Initial state=s0, action=s1, reward=-H, next state=s1, goal=g0, discount rate=gamma].

So the essence of HAC is data enhancement.

**Algorithm 1** Hierarchical Actor-Critic (HAC)

**Input:**

- Key agent parameters: number of levels in hierarchy $k$, maximum subgoal horizon $H$, and subgoal testing frequency $\lambda$.

**Output:**

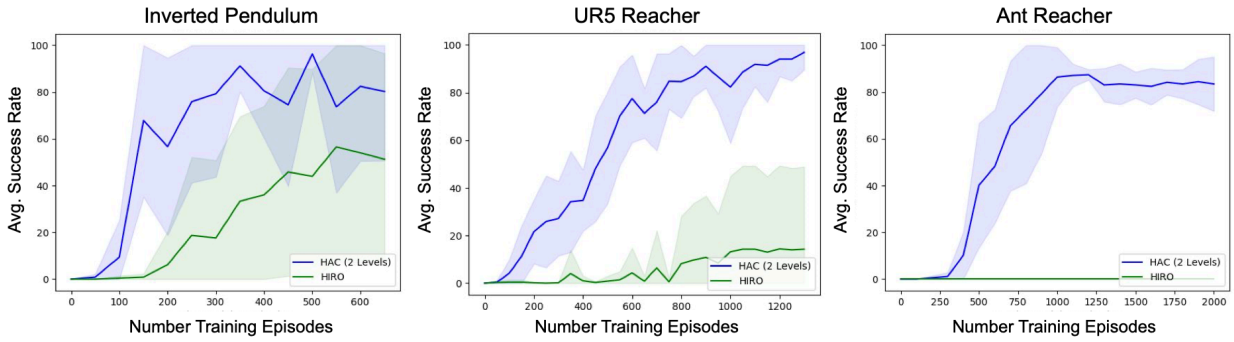
- $k$ trained actor and critic functions $\pi_0, ..., \pi_{k-1}, Q_0, ..., Q_{k-1}$

**for** $M$ episodes **do** ▷ Train for M episodes
  $s \leftarrow S_{init}$, $g \leftarrow G_{k-1}$ ▷ Sample initial state and task goal
  $train - level(k-1, s, g)$ ▷ Begin training
  Update all actor and critic networks
**end for**

**function** TRAIN-LEVEL($i :: level, s :: state, g :: goal$)
  $s_i \leftarrow s$, $g_i \leftarrow g$ ▷ Set current state and goal for level $i$
  **for** $H$ attempts or until $g_n$, $i \leq n < k$ achieved **do**
    $a_i \leftarrow \pi_i(s_i, g_i)$ + $noise$ (if not subgoal testing) ▷ Sample (noisy) action from policy
    **if** $i > 0$ **then**
      Determine whether to test subgoal $a_i$
      $s_i^{'} \leftarrow train - level(i-1, s_i, a_i)$ ▷ Train level $i-1$ using subgoal $a_i$
    **else**
      Execute primitive action $a_0$ and observe next state $s_0^{'}$
    **end if**
    ▷ Create replay transitions
    **if** $i > 0$ and $a_i$ missed **then**
      **if** $a_i$ was tested **then** ▷ Penalize subgoal $a_i$
        $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r = Penalty, s^{'} = s_i^{'}, g = g_i, \gamma = 0]$
      **end if**
      $a_i \leftarrow s_i^{'}$ ▷ Replace original action with action executed in hindsight
    **end if**
    ▷ Evaluate executed action on current goal and hindsight goals
    $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r \in \{-1, 0\}, s^{'} = s_i^{'}, g = g_i, \gamma \in \{\gamma, 0\}]$
    $HER\_Storage_i \leftarrow [s = s_i, a = a_i, r = TBD, s^{'} = s_i^{'}, g = TBD, \gamma = TBD]$
    $s_i \leftarrow s_i^{'}$
  **end for**
  $Replay\_Buffer_i \leftarrow$ Perform HER using $HER\_Storage_i$ transitions
  **return** $s_i^{'}$ ▷ Output current state
**end function**

## 5.2 HAC's effect

In this paper, the authors test the HAC model and HIRO model in three games, and the results showed that HAC is much better than HIRO in both training speed and training effect. In the paper, the authors also mention the disadvantages of HIRO: (i) HIRO does not use Hindsight Experience Replay at either of the 2 levels and (ii) HIRO uses a different approach for handling the non-stationary transition functions. In other words, HIRO values subgoal actions with respect to a transition function that essentially uses the current lower level policy hierarchy, not the optimal lower level policy hierarchy as in our approach. Consequently, HIRO may need to wait until the lower level policy converges before the higher level can learn a meaningful policy.




# 6 conclusion

In this research report on frontier work, I chose the topic "hierarchical reinforcement learning", researched and read four recent papers. On the basis of the DQN model I have learned in the course, I also learned H-DQN, FuN, HIRO and HAC four models of hierarchical reinforcement learning. At the same time, I also have a general understanding of AC, DDPG, TD3 and A3C model training methods. Of course, due to the limitation of time and ability, I did not repeat the selected papers, and as for these research papers, I cannot say that I have achieved a complete and thorough understanding, which is also a weakness of my final project.