# Hierarchical Actor-Critic

**Andrew Levy** [1] **Robert Platt** [2] **Kate Saenko** [1]

## Abstract

The ability to learn at different resolutions in time may help overcome one of the main challenges in deep reinforcement learning — sample efficiency. Hierarchical agents that operate at different levels of temporal abstraction can learn tasks more quickly because they can divide the work of learning behaviors among multiple policies and can also explore the environment at a higher level. In this paper, we present a novel approach to hierarchical reinforcement learning called Hierarchical Actor-Critic (HAC) that enables agents to learn to break down problems involving continuous action spaces into simpler subproblems belonging to different time scales. HAC has two key advantages over most existing hierarchical learning methods: (i) the potential for faster learning as agents learn short policies at each level of the hierarchy and (ii) an end-to-end approach. We demonstrate that HAC significantly accelerates learning in a series of tasks that require behavior over a relatively long time horizon and involve sparse rewards.

## 1. Introduction

Despite major successes in both simulated and real-world tasks, a key problem with many deep reinforcement learning (RL) algorithms is that they are slow. Learning is particularly slow when the rewards granted to agents are sparse. One major reason for reinforcement learning's poor sample efficiency is that many existing algorithms force agents to learn at the lowest level of temporal abstraction. For instance, if a simulated robot agent is given a task involving locomotion, the agent will need to learn the entire sequence of joint torques to accomplish the task (Lillicrap et al., 2015) instead of trying to break the problem down at a higher level. Learning exclusively at low levels of abstraction slows down learning for two key reasons. First, agents must learn longer sequences of actions in order to achieve the desired behavior. This is problematic because policies involving longer sequences of actions are more difficult to learn, particularly when rewards are sparse. The process of propagating back Q-values from the actions that produce the sparse reward to the preceding actions takes longer. The issue of long-term credit assignment also becomes more severe as the action-value function needs to learn Q-values for a larger portion of the state-action space. Second, learning at the lowest level restrains exploration. Agents that can propose higher level subgoals can more quickly determine the distant states that are helpful in achieving certain behavior goals. A faster exploration of the state space of the environment may speed up the process of learning a robust policy.

Yet most existing hierarchical RL methods do not provide an approach for breaking down tasks involving continuous action spaces that guarantees shorter policies at each level of abstraction and is end-to-end. Most current hierarchical approaches only enable agents to learn at higher levels if the action space is discrete (Dayan & Hinton, 1993) (Vezhnevets et al., 2017). Further, many existing hierarchical learning approaches do not implement hierarchical agents that equitably divide up the work of learning a behavior among the agent's multiple policies. For instance, many approaches choose to decompose problems into smaller state spaces rather than into smaller time scales (Dayan & Hinton, 1993). This can be problematic in continuous action space environments as a policy that acts within a small region of the state space may need a lengthy sequence of actions to escape that region. Most existing hierarchical approaches also require non-trivial manual work including designing non-sparse reward functions, preselecting the set of possible higher level subgoals rather than learning them from experience, and staggering the training of different policies (Sutton et al., 1999) (Kulkarni et al., 2016).

In this paper, we introduce a novel approach to hierarchical reinforcement learning called Hierarchical Actor-Critic (HAC). The algorithm enables agents to learn to divide tasks involving continuous state and action spaces into simpler problems belonging to different time scales. HAC achieves this objective by implementing agents that learn multiple policies in parallel. Each successive policy in the hierarchy is responsible for learning how to break down problems into subproblems with increasingly fine time resolutions. Figure 1 should provide some intuition on how HAC agents learn at different time scales. The figure shows an agent

[1]Department of Computer Science, Boston University, Boston, MA, USA [2]College of Information and Computer Science, Northeastern University, Boston, MA, USA.

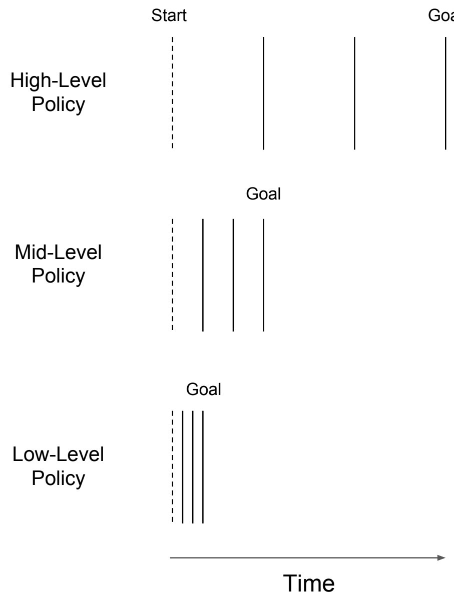

*Figure 1.* Example HAC Hierarchy. Agent in figure uses three policies to learn a behavior. Each policy specializes in breaking down problems into subproblems with finer time resolutions.

that uses three policies to accomplish some behavior. The solid vertical lines represent the time resolutions of subgoals output by each policy. The more distance there is between consecutive vertical lines, the more time the agent has to achieve each subgoal. The low-level policy outputs actual agent actions so the vertical lines for the low-level policy can be interpreted as subgoals requiring one action. In the figure, the high-level policy breaks down the end goal into three subgoals with relatively large time resolutions. The mid-level policy specializes in dividing each subgoal from the high-level policy into three subgoals belonging to shorter time scales. Finally, the low-level policy specializes in decomposing each subgoal from the mid-level policy into three agent actions, which represent the smallest time resolution. A crucial benefit of having each policy specialize in breaking down goals of particular time scale into subgoals of a certain smaller time scale is that the policies that are learned are limited in length. This is beneficial because shorter policies can be learned more quickly than longer ones. Further, having multiple policies that operate at different levels of temporal abstraction is helpful because it enables high-level exploration, which can also accelerate learning.

Hierarchical Actor-Critic helps agents learn a hierarchy of policies similar to Figure 1 using a set of actor-critic networks. Each actor-critic network is responsible for learning one of the policies within the hierarchy. The policies or actor networks that are learned are goal-based, meaning that they take as input the current state and a goal and output an action. Each goal-based actor network learns limited length policies that operate at different time resolutions due to a critical feature of the algorithm — time limits. Each actor network has only a certain number of actions to achieve its higher level input goal. Section 3 explains how time limits enable each actor network to specialize in a different time scale.

Another key advantage of HAC is that it provides an end-to-end hierarchical learning approach. HAC learns to separate goals into subgoals using just the agent's experience and the algorithm only requires sparse reward functions. The hierarchical policies are also learned in parallel and do not need to be learned in different phases.

For this paper, we ran a series of experiments comparing the performance of agents that did and did not use the Hierarchical Actor-Critic algorithm. The tasks examined include pendulum, reacher, cartpole, and pick-and-place environments. In each task, agents that used Hierarchical Actor-Critic significantly outperformed those that did not. In some tasks, the use of Hierarchical Actor-Critic appears to be the difference between consistently solving a task and rarely solving a task. A video showing the results of our experiments is available at `https://www.youtube.com/watch?v=m3EYeBpGepo`.

## 2. Background

Hierarchical Actor-Critic builds off three techniques from the reinforcement learning literature: (i) the Deep Deterministic Policy Gradient (DDPG) learning algorithm (Lillicrap et al., 2015), (ii) Universal Value Function Approximators (UVFA) (Schaul et al., 2015), and (iii) Hindsight Experience Replay (HER) (Andrychowicz et al., 2017).

DDPG serves as the key learning infrastructure within Hierarchical Actor-Critic. DDPG is an actor-critic algorithm and thus uses two neural networks to enable agents to learn from experience. The actor network learns a deterministic policy that maps from states to actions $\pi : S \rightarrow A$. The critic network approximates the Q-function or the action-value function of the current policy $Q^{\pi}(s_t, a_t) = E[R_t | s_t, a_t]$, in which $R_t$ is the discounted sum of future rewards $\sum_{i=t}^{\infty} \gamma^{i-t} r_i$. Thus, the critic network maps from (state, action) pairs to expected long-term reward $Q : S \times A \rightarrow R$. In order to learn a near-optimal policy that results in large expected long-term reward, DDPG follows a cyclical process composed of two steps: (i) policy evaluation and (ii) policy improvement. In the policy evaluation phase, the agent first interacts with the environment for a period of time using a noisy policy $\pi(s) + N(0, 1)$, in which $N(\cdot)$ is some normal distribution. The transitions experienced are stored as $(s_t, a_t, r_t, s_{t+1})$ tuples in a replay buffer. The agent then updates its approximation of the Q-function of the current policy by performing mini-batch gradient descent on the loss function $L = (Q(s_t, a_t) - y_t)^2$, in which the target $y_t$ is the Bellman estimate of the Q-function $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$. In the policy improvement phase, the agent modifies its policy based on the updated approximation of the action-value function. The actor function is trained by moving its parameters in the direction of the gradient of $Q$ w.r.t. the actors parameters.

Universal Value Function Approximators is a second idea that is critical to HAC. UVFA extends the action-value function to incorporate goals. The Q-function now represents the expected long-term reward of taking an action given the current state and goal $Q^{\pi}(s_t, a_t, g_t) = E[R_t|s_t, a_t, g_t]$. Each goal has its own reward function $r_g(s_t, a_t, s_{t+1})$ and discount function $\gamma_g(s)$. $\gamma_g(s)$ = 0 when the agent is in a state that achieves the prescribed goal as the current state can be viewed as a terminating one. Goals are critical to HAC because goals are often hierarchical and can be broken down into subgoals. Goals are also useful because they can be used easily with sparse and binary reward functions.

Hindsight Experience Replay is another component from the reinforcement learning literature that is integral to Hierarchical Actor-Critic. HER helps agents learn goal-based policies more quickly when sparse reward functions are used. The idea behind HER is that even though an agent may have failed to achieve its given goal in an episode, the agent did learn a sequence of actions to achieve a different objective in hindsight — the state in which the agent finished. Learning how to achieve different goals in the goal space should help the agent better determine how to achieve the original goal. Hindsight Experience Replay is implemented by creating a separate copy of the transitions $(s_t, a_t, r_t, s_{t+1}, g)$ that occurred in an episode and replacing (i) the original goal with the goal achieved in hindsight and (ii) the original reward with the appropriate value given the new goal.

## 3. Hierarchical Actor-Critic

We introduce a new hierarchical RL approach called Hierarchical Actor-Critic. The algorithm helps agents learn long time horizon tasks involving continuous action spaces and sparse rewards more quickly by enabling agents to learn to break down those tasks into easier subtasks belonging to different time scales. HAC directly addresses the issue of lengthy policies that hinder many existing non-hierarchical and hierarchical approaches as HAC agents learn limited policies at each level of temporal abstraction. The approach is also end-to-end as it learns subgoal policies at different levels of temporal abstraction on its own and in parallel and only requires sparse reward functions.

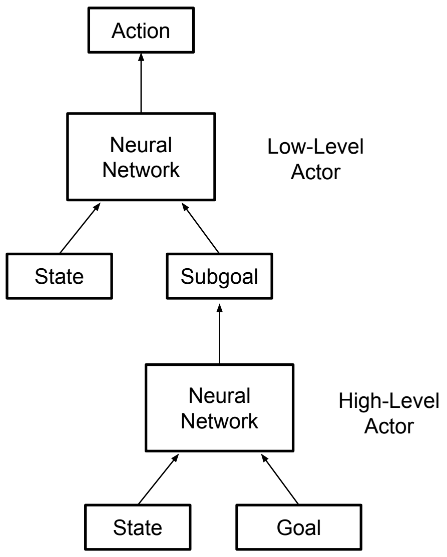

*Figure 2.* Hierarchical policy with 1 subgoal layer

### 3.1. Architecture

The objective of the algorithm is to learn a hierarchical policy like the one shown in Figure 2. The hierarchical policy is composed of multiple goal-based policies or actor networks. Each actor network takes as input the current state and higher level goal and outputs an action belonging to a particular time scale. For the subgoal actor networks, such as the bottom network in Figure 2, this action is a proposed subgoal. The proposed subgoal is a desired future state or set of future states for the agent. For the actor network operating at the lowest level of abstraction, such as the top network in Figure 2, the action is the agent's actual output. In our experiments, we trained agents that used hierarchical policies composed of two and three actor networks. Additional layers can be easily added.

Each actor network has its own critic network and replay buffer to learn a near-optimal policy. The actor networks from Figure 2 are shown connected to their respective critic networks in Figure 3. Each critic network approximates the Q-function for its associated policy $Q^{\pi_i}(s_t, a_t, g_t) = E[R_t|s_t, a_t, g_t]$ using the Bellman equation as a target $y_t = r_g + \gamma_g Q^{\pi_i}(s_{t+1}, \pi_i(s_{t+1}), g_t)$. $r_g$ is sparse and binary and is granted when the agent has reached a state within a certain distance of the goal. As described in (Schaul et al., 2015), $\gamma_g = 0$ when the prescribed goal has been achieved as a terminating state has been reached.

### 3.2. Temporal Abstraction via Limited Policies

Each actor network within the hierarchical policy learns a limited length policy as a result of time limits. Actor net-

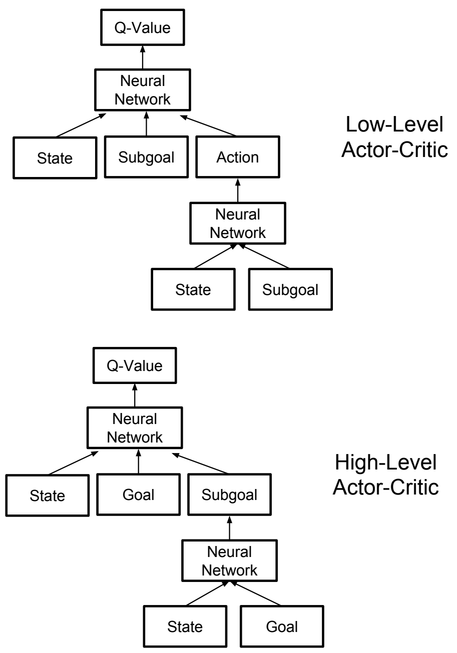

*Figure 3.* Actor-Critic networks for hierarchical policy with 1 subgoal layer.

works can only take a certain number of actions to achieve their higher level goal. For the lowest level actor network, this means it can only execute a certain number of actual agent actions to achieve a subgoal. For higher level subgoal actor networks, the policy limit means the network must achieve its higher level goal within a maximum number of subgoals. The policy length limit for actor network $i$ is controlled by a hyper parameter $T_i$. In our experiments, we generally used the same value for $T_i$ for each actor network within the hierarchical policy. This limit thus restricts each actor network to only learning how to achieve goals that can be accomplished within a certain number of actions, which shortens the goal-based policy that is learned.

The combination of time limits and Hindsight Experience Replay enables each actor network to specialize in different time resolutions. Actor networks learn to operate at different time scales largely as a result of hindsight learning, in which the agent learns how to achieve the goal states the agent actually reached during an episode with the help of HER. The following example should provide some clarity. Consider an agent using a two layer hierarchical policy with $T_0 = T_1 = 10$, meaning that the agent must achieve each subgoal in no more than 10 agent actions and achieve the end goal in no more than 10 subgoals. Even in the worst case scenario in which the agent fails after 100 agent actions to achieve any of the 10 subgoals and the end goal, the agent will still be able to learn how it could have divided up the task of achieving the final state into problems belonging to different time scales. In order to achieve the last state reached on the 100th action, the agent could have chosen every 10th state to be a subgoal state. This is a valid breakdown because the end goal is reached in no more than 10 subgoals and each subgoal is achieved in no more than 10 actions by the agent. From this breakdown, the higher level network's replay buffer receives a sequence of 10 transitions showing how it can use subgoals belonging to larger time resolutions to achieve the hindsight end goal. Similarly, the low-level network's replay buffer receives 10 sequences of 10 transitions each showing how it can use actions belonging to the smallest time scale to achieve each of the 10 subgoals. Over the course of many episodes, each actor network learns to achieve goals with actions belonging to its respective time scale. Thus, time limits are critical for helping each actor network specialize in a different time resolution because they provide a simple way to divide up a sequence of actions that achieved some goal into multiple sequences belonging to different time scales.

Learning limited policies at different time resolutions presents significant benefits as well as new challenges for the agent. The key benefit is that it should be easier to learn multiple shorter policies in parallel than one long policy. Credit assignment is less of a problem as the critic function for each policy only needs to learn Q-values for a more limited region of the state-action-goal space. Learning is faster because reinforcement learning agents essentially learn from end to beginning when sparse rewards are used. This backwards learning process occurs more quickly if the policy requires fewer actions. However, the use of limited policies also results in a significant new dilemma — subgoal actor networks now have conflicting missions. Subgoal actor networks need to learn a policy that can simultaneously (i) achieve its higher level goal in as few actions (i.e., subgoals) as possible but also (ii) output subgoals that can be achieved by the lower-level actor network in a limited number of steps. Producing subgoals that are too ambitious may not be achievable by lower level layers as they specialize in limited policies. Overly ambitious subgoals could thus result in the failure of the higher level actor network to achieve its own goal. To simultaneously solve both of its conflicting objectives, the upper and lower level layers need to coordinate as the upper level needs to understand the types of subgoals the lower level can accomplish.

We take two steps to incentivize subgoal actor network $i$ to output subgoals that can be achieved by actor network $i - 1$ in no more than $T_{i-1}$ actions. First, as in the example discussed above, all experience transitions passed to the replay buffers of subgoal actor networks contain actions (i.e. subgoals) that were actually achieved by the succeeding, lower level actor network within the maximum number of actions. Second, HAC penalizes proposed subgoals that were not achieved. A certain percentage of the time defined by a hy-

**Algorithm 1** Hierarchical Actor-Critic

```
Initialize Actor-Critic networks $[(\pi_0, Q_0), \ldots, (\pi_n, Q_n)]$
Initialize Replay Buffers $(R_0, \ldots, R_n)$
**for** $episode = 1$ **to** $M$ **do**
    Sample actual goal, $G$, and initial state, $s_0$
    **for** $t_n = 1$ **to** $T_n$ **do**
        Testing Boolean $B \leftarrow \{0, 1\}$
        Sample layer $n$ subgoal from $\pi_n$
            $g_{t_n} \leftarrow \pi_n(s(t_n), G) + \overline{B} \cdot N(0, 1)$
            . . .
        **for** $t_0 = 1$ **to** $T_0$ **do**
            Sample action from $\pi_0$
                $a_{t_0} \leftarrow \pi_0(s(t_0), g_1) + \overline{B} \cdot N(0, 1)$
            ProcessTrans($s_{t_0}$,$a_{t_0}$,$r_{t_0+1}$,$s_{t_0+1}$,$g_{t_1}$)
            **if** $g_{t_1}, ..., G$ achieved **or** $t_0 = T_0$ **then**
                Perform HER on layer 0 transitions
                break
            **end if**
        **end for**
            . . .
        ProcessTrans($s_{t_n}$,$g_{t_n}$,$r_{t_n+1}$,$s_{t_n+1}$,$G$)
        **if** $G$ achieved **or** $t_n = T_n$ **then**
            Perform HER on layer n transitions
            break
        **end if**
    **end for**
    Update Actor-Critic Networks
**end for**
```

**Algorithm 2** Transition Processing

```
**Function** ProcessTrans($s_{t_i}$,$a_{t_i}$,$r_{t_i+1}$,$s_{t_i+1}$,$g_{t_{i+1}}$)
    **if** subgoal layer **and** $a_{t_i}$ not achieved **then**
        $R_i \leftarrow (s_{t_i}, g'_{t_i}, r_{t_i+1}, s_{t_i+1}, g_{t_{i+1}})$
        **if** Testing **then**
            $R_i \leftarrow (s_{t_i}, a_{t_i}, -T_i, s_{t_i+1}, g_{t_{i+1}})$
        **end if**
    **else**
        $R_i \leftarrow (s_{t_i}, a_{t_i}, r_{t_i+1}, s_{t_i+1}, g_{t_{i+1}})$
        **if** subgoal layer **then**
            Store HER trans $(s_{t_i}, g'_{t_i}, TBD, s_{t_i+1}, TBD)$
        **else**
            Store HER trans $(s_{t_i}, a_{t_i}, TBD, s_{t_i+1}, TBD)$
    **end if**
**end function**
```

per parameter, agents will test subgoals by not adding noise to the subgoals and actions prescribed by its hierarchical policy. Noise needs to be removed when testing subgoals because an agent may miss a subgoal due to noise added to lower level actions. Subgoals that cannot be achieved with the agent's current, noise-free policy will be penalized. In our experiments, if layer $i$ proposed a subgoal that was not achieved, layer $i$ received a reward of $-T_i$. In addition, we set $\gamma^g = 0$ when a subgoal is missed during testing as the Q-value of a missed proposed subgoal should not depend on the Q-value of a different state. If $\gamma^g = 0$, then the reward for proposing the subgoal must be $-T_i$ or less. Otherwise, proposing the missed subgoal may have a larger Q-value than some subgoals that do efficiently guide the agent to the goal, and the subgoal actor network may not update its policy as a result. We observed that the extra penalty for proposing a missed subgoal works as intended as the subgoal layer would adjust to proposing more conservative subgoals that the lower-level layer can achieve.

## 3.3. Algorithm

The main part of the HAC algorithm, shown in Algorithm 1, is concerned with ensuring the correct experience transitions are passed to each actor network's replay buffer. In order to learn policies of a certain length, actor networks need to be passed sequences of transitions of the same length. Also, as discussed above, in order for each subgoal actor network to learn to output effective subgoals that belong to its respective time resolution, certain transitions need to be passed to each actor network.

HAC proceeds as follows. The initial state and end goal are passed to the actor network $n$, which then triggers the passing of a series of subgoals from higher level actor networks to lower level actor networks. If the agent is in a subgoal testing phase, noise will not be added to these subgoals. The cascade of subgoals terminates when the last subgoal is passed to the lowest level actor function, which is located in the innermost for loop. The lowest level actor then has $T_0$ attempts to try to achieve the provided subgoal. After each action by the lowest level actor network, two copies of transitions are created as a result of the call to the *ProcessTrans* function shown in Algorithm 2. The first transition $(s_{t_0}, a_{t_0}, r_{t_0+1}, s_{t_0+1}, g_{t_1})$ is placed in replay buffer $R_0$. This transition indicates whether the action $a_{t_0}$ taken in state $s_{t_0}$ was able to achieve goal $g_{t_1}$. The second transition $(s_{t_0}, a_{t_0}, TBD, s_{t_0+1}, TBD)$ awaits further processing during Hindsight Experience Replay. Once the lowest level actor has run out of $T_0$ attempts or achieves any of the higher-level goals $g_{t_1}$,...,$G$, Hindsight Experience Replay is executed. During HER, a few random transitions are selected to serve as hindsight goals. As a result, for the selected transition and all preceding transitions, the TBD goal component is replaced with $g'_{t_1} = f(s_{t_1+1})$, in which $f(\cdot)$ is a function that transforms an array from the state space to the the goal space. $g'_{t_1}$ can thus be interpreted as the subgoal achieved in hindsight. The TBD reward component is then replaced with its appropriate value given the updated goal. The purpose of these HER transitions is thus to help the agents learn the sequences of actions that were used to achieve hindsight goals, even though these hindsight goals may be different than the original target goals. After HER is executed for the lowest level actor network, control is then passed to the preceding higher level actor network.

The completion of the innermost for loop consists of 1 subgoal action for actor network 1. Different transitions will be created depending on (i) whether its original proposed subgoal $g_{t_1}$ was achieved by actor network 0 and (ii) whether the agent is currently in a testing phase. If the proposed subgoal was not achieved, the transition $(s_{t_1}, g'_{t_1}, r_{t_1+1}, s_{t_1+1}, g_{t_2})$ is passed to replay buffer $R_1$. This transition indicates whether the subgoal achieved in hindsight, $g'_{t_1}$, was able to achieve the higher level goal $g_{t_2}$. If the proposed subgoal was missed and the agent is currently testing subgoals, the actor network 1 is penalized with the transition $(s_{t_1}, g_{t_1}, -T_1, s_{t_1+1}, g_{t_2})$. On the other hand, if subgoal $g_{t_1}$ was achieved by actor network 0, the transition $(s_{t_1}, g_{t_1}, r_{t_1+1}, s_{t_1+1}, g_{t_2})$ is passed to $R_1$. Finally, an additional transition $(s_{t_1}, g'_{t_1}, TBD, s_{t_i+1}, TBD)$ is created for Hindsight Experience Replay. Actor network 1 then proposes a second subgoal for actor network 0 and the process begins again. The key insight here is that the second component or the "action" component in every transition passed to the subgoal actor networks identifies a subgoal that has been achieved by the lower level actor network in no more than $T_{i-1}$ steps, except for the transitions that penalize bad subgoals. As a result, each subgoal actor network learns to output actions that both belong to a a particular time resolution and are helpful towards achieving the actor network's higher level goal.

After interacting with the environment for one or more episodes and filling its replay buffers with transitions, the agent now learns from the experience by updating its actor-critic functions. Mini-batch gradient descent is performed on the critic network to push the Q-function closer to the Bellman estimates. Next, the parameters of the actor function $\theta_i$ are moved in the direction of the gradient of the Q-function w.r.t $\theta_i$. For the lowest level actor network, updating its actor-critic network should enable the actor to find more efficient routes to achieving its shorter term goals. For each subgoal actor-critic layer, updating its actor-critic network means finding a better way to balance its conflicting objectives of finding the sequence of subgoals that can (i) most quickly solve the higher level goal and (ii) be achieved in no more than $T_{i-1}$ steps by the succeeding actor layer.

## 4. Experiments

We evaluated the Hierarchical Actor-Critic approach on a total of 5 tasks: Pendulum-Climb, Pendulum-Balance, UR5 Reacher, Cartpole, and 1-Object Pick-and-Place. In each environment, we compared the performance of agents using 0, 1, and 2 subgoal layers. Agents using 0 subgoal layers were thereby only using DDPG and HER. A video showing the results of our experiments is available at https://www.youtube.com/watch?v=m3EYeBpGepo.

### 4.1. Environments

We assessed HAC on the following 5 tasks. Each of these simulations was built using the Mujoco physics engine (Todorov et al., 2012).

1. Pendulum - Climb
   The goal of this task is for the agent to swing the pendulum to its maximum height, marked by a yellow cube. The agent only needs to touch the yellow sphere located at the peak and does not need to try to balance the pole. We found that an efficient policy could solve this task in around 100 low-level actions.

2. Pendulum - Balance
   The goal for this environment is to balance the pendulum at its peak near the yellow sphere. Thus, to achieve the goal the pole must be located near the peak and have angular velocity near 0. Figure 4 shows a few frames from a successful episode. We found that an efficient policy could solve this task in around 150 low-level actions.

3. UR5 Reacher
   The goal of this task is for the agent to learn to move to a randomly designated point, marked by a yellow cube. The agent in this task is a simulated UR5, a 6 DOF robotic arm. To make the task require a longer time horizon, the goal location is always in the quadrant in front and opposite the starting location of the gripper. We found that an efficient policy could solve this task in around 60 individual actions.

4. Cartpole Swingup
   The goal for this task is to swing the pole up to the yellow cube. In order to achieve the goal, the angular velocity of the pole must also be near 0 and the position of the cart must be below the yellow cube. We found that an efficient policy could solve this task in around 170 low-level actions.

5. 1-Object Pick-and-Place
   The idea for this task was to assess how Hierarchical Actor-Critic would perform in a task with natural hierarchy. The objective in this task is to pick up the blue rod and move it to the yellow rod. The agent is a 2 joint robot worm. Our more efficient agents can solve this task in around 110 steps.

**States, Actions, Rewards**
The state space in all environments include joints positions and joint velocities. The actions are joint torques. Further,

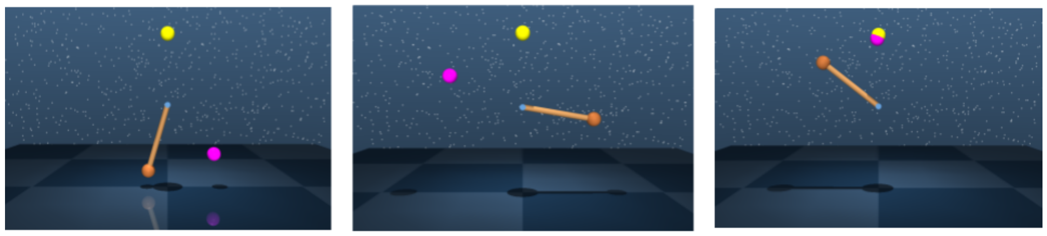
*Figure 4.* Three frames from a successful episode of the Pendulum - Balance task. The agent learns to reach the goal (touch the yellow sphere) by breaking down the task into subgoals (touch the purple spheres).

*Table 1.* Environment subgoal and goal descriptions

| TASK | SUBGOAL | GOAL |
|---|---|---|
| PENDULUM-CLIMB | $\theta_{xy}$ | $\theta_{xy}$ |
| PENDULUM-BALANCE | $\theta_{xy}, \dot{\theta}$ | $\theta_{xy}, \dot{\theta}$ |
| UR5 REACHER | $\theta_{effector_{xyz}}$ | $\theta_{effector_{xyz}}$ |
| CARTPOLE | $\theta_{xy}, \dot{\theta}$ | $\theta_{xy}, \dot{\theta}$ |
| PICK-AND-PLACE | $\theta_{xy}$ | $\theta_{xy}$ |

each network within the stack has its own sparse reward function. In our experiment, a reward of -1 was granted if an actor network took an action that did not complete its goal and a reward of 0 if the action was successful. Also, a larger negative reward was issued for subgoal networks that proposed a subgoal that the lower level layers could not achieve. Rewards are granted by a layer if the agent has moved within a certain distance of the layer's goal. Further, the rewards granted by each network are independent of each other. For instance, an agent may miss the low-level subgoal but achieve a high-level subgoal. In this case, the transition passed to the low-level replay buffer will still list the negative reward for taking that action given the current state and subgoal because the action was unsuccessful.

**Goals and Subgoals**

Table 1 shows the subgoals and end goals for each task. $\theta_{xy}$ and $\dot{\theta}$ represent Cartesian joint positions and joint velocities, respectively. The subgoals in our experiments were essentially desired future states that the agent learns to achieve along the way to reaching the end goal. For some of our experiments, we used a lower dimensional version of the state as the subgoal.

The end goal shown in the last column of Table 1 depends on the behavior the user would like the agent to learn. The more specific the desired behavior, the more similar the end goal should be to the state. For instance, in our Pendulum - Climb task, the agent just needed to learn to swing up to the peak in as few actions as possible. The agent did not need to try to balance the pole. Thus, for this task the end goal was 2-dimensional and includes the (x,y) coordinates of maximum height of the pendulum. On the other hand, in the Pendulum - Balance task, the agent needed to both swing the pole to its peak and maintain a near 0 angular velocity when the pole was upright. Thus, the goal for this task was 3-dimensional and included the (x,y) coordinates of maximum height of the pendulum and the desired angular velocity of 0.

## 4.2. Results

The accompanying video and Figure 4.1 show the results of our experiments. In Figure 4.1, each chart plots the percentage of goals achieved by agents using 0, 1, and 2 subgoal layers in each testing period. Testing periods are separated by about 300 episodes and each testing period consists of 64 episodes. Each plot represents the average performance over 7-10 runs of each task. Agents learn each task from scratch and thus begin with no pre-training. In episode 0, agents are thereby using random policies.

The key result from our work is that the use of Hierarchical Actor-Critic resulted in a substantial improvement in performance. In all 5 environments, agents that used HAC learned a robust policy significantly faster. Indeed, for many tasks, agents that used no subgoal layers were not able to consistently solve the task.

The video and the frames in Figure 4 also demonstrate that the algorithm is working as intended – the agents are learning how to break problems down into easier subproblems. In the Pendulum - Balance environment, the subgoals represented by purple spheres show that the agent has learned that if it wants to balance the pole upright it first needs to swing back and forth. In the UR5 Reacher environment, if the goal location marked by the yellow cube is far away from an agent using 2 subgoal layers, the agent will often put the high-level subgoal marker, the green cube, about halfway to the end goal. The low-level subgoals, marked by the purple cubes, then appear to guide agent toward the high-level subgoal. This indicates the agent has learned how to effectively separate high-level goals into easier low-level goals. Similar hierarchical behavior is evident in the pick-and-place environments. In the 1-Object Pick-and-Place, the high-level subgoal, represented by the green sphere and

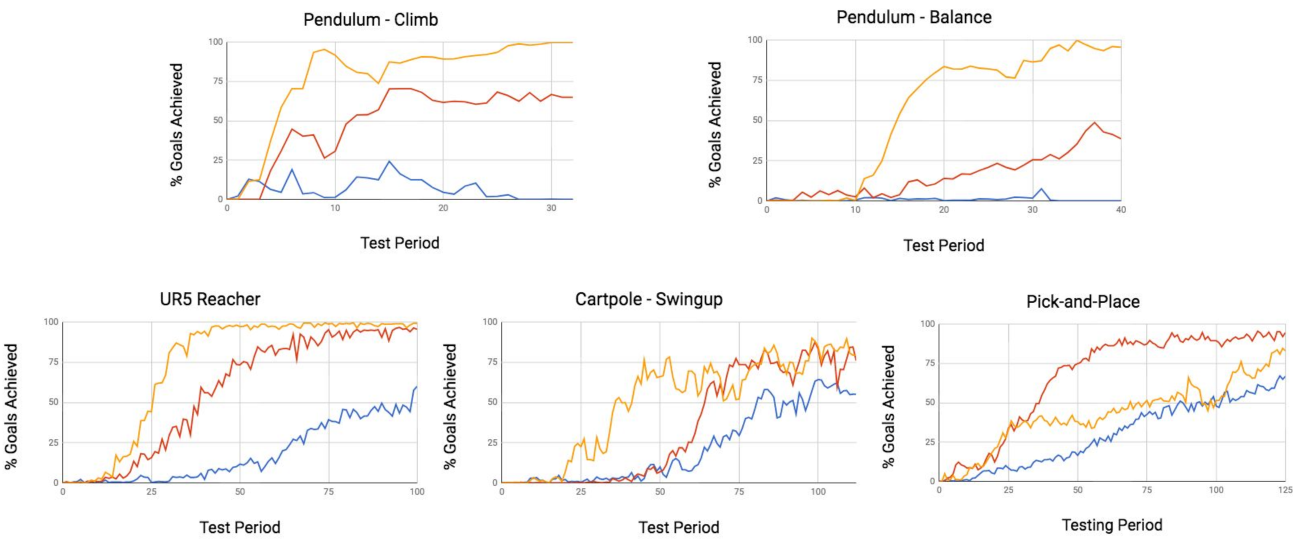

*Figure 5.* The above plots compare the performance of agents that use the baseline DDPG+HER method with 0 subgoal layers (blue), and our HAC method with 1 (red) and 2 (yellow) subgoal layers, for each of the five tasks. Performance is measured by the percentage of end goals achieved in each testing period, which consists of 64 episodes.

rod, will often appear in the vicinity of the blue rod when the blue rod has not yet been picked up. Once the blue rod has been attached, the high-level subgoal moves to the yellow rod. The low-level subgoal, represented by the purple sphere and rod, again guide the agent to the green high-level subgoal.

We also observed that for most trials in each task except for the pick-and-place task, agents using 2 subgoal layers learned significantly faster than agents using 1 subgoal layer. Agent using 2 subgoal layers learn an extra policy but each of the three policies specialize in learning shorter policies than the 2 policies learned by agents using 1 subgoal layer. This result further supports the main premise of the algorithm — hierarchical agents that learn more shorter policies in parallel can outperform agents learning fewer longer policies.

## 5. Related Work

Hierarchical RL is a topic of ongoing research (Sutton et al., 1999), (Dayan & Hinton, 1993), (Vezhnevets et al., 2017), (Dietterich, 1998). One popular hierarchical reinforcement learning approach is feudal reinforcement learning (Dayan & Hinton, 1993). In feudal reinforcement learning, the state space is divided into increasingly small regions at each level of abstraction. Dayan and Hinton (Dayan & Hinton, 1993) present a grid world example, in which a maze is continually divided into quarters at each level. Each level has a set of managers that are in charge of providing goals and rewards to the 4 sub-managers below. Sub-managers need to learn to complete their goals by learning how to give tasks to their own sub-managers. Dayan and Hinton (Dayan & Hinton, 1993) show that a feudal structure outperforms a non-hierarchical Q-learning approach. A key difference between HAC and feudal reinforcement learning is that the latter breaks down problems along the spatial dimension instead of the temporal dimension. This is problematic for two reasons. First, it is unclear how the state space would be divided for high-dimensional and continuous state spaces. Second, even if there was a way to divide a high-dimensional continuous state space, the feudal approach does not guarantee the hierarchical policies learned will each be short. There may be some small region of the continuous state space that is difficult to maneuver and may require many actions from a manager. HAC, on the other hand, motivates its actor networks to learn shorter policies, which can accelerate learning.

Another popular framework in hierarchical reinforcement learning is the options framework (Sutton et al., 1999). This approach generally uses a hierarchy of two layers to enable agents to break problems down. The low-level layer consists of multiple options, each of which is a policy that can solve a specific task. The high-level layer is responsible for learning the sequence of these specific policies that can achieve a task. HAC uses a different approach to breaking problems down. Instead of having the high-level policy select one of many specific low-level policies, the high-level network provides a subgoal to a single low-level network, which is trained to achieve a variety of subgoals as it learns a goal-based policy. Using one low-level goal-based policy network instead of several non-goal-based policies should provide some efficiency advantages because learning how to achieve one subgoal will often help in learning how to achieve different subgoals. For instance, in a pick-and-place task, learning how pick up and drop off an object in a certain location should help the agent learn to pick up and drop off the object in a different target location.

Kulkarni et al.(Kulkarni et al., 2016) proposed an approach with some similarities to both the options framework and HAC. The algorithm, named hierarchical-DQN (h-DQN), aims to help agents solve tasks in environments with discrete action spaces. Agents implemented with h-DQN break down tasks using two value functions. The high-level layer attempts to learn a sequence of subgoals that can accomplish a task. The low-level layer attempts to learn a sequence of individual actions that can achieve the provided subgoal. The low-level layer thus learns a goal-based policy and value functions similar to HAC. However, unlike the Hierarchical Actor-Critic method, h-DQN does not enable agents to learn the sequence of high-level subgoals from scratch while using only sparse reward functions. In the papers Montezumas Revenge example, the agent was provided with the set of the possible subgoals, which included objects in the game such as doors, ladders, and keys. The agent was then responsible for learning the order these items needed to be reached. An external reward function was also used to help the agent more quickly find the order of these subgoals. One key reason Hierarchical Actor-Critic does not need aids like sets of subgoals or manually-engineered reward functions is the use of Hindsight Experience Replay. With HER, as long as the agent can occasionally achieve goals that are nearby the intended goal, the agent should have a chance to learn the desired behavior.

## 6. Conclusion

We introduced a new technique called Hierarchical Actor-Critic that uses temporal abstraction to break down complex problems into easier subproblems. Our results indicate that only using one policy to learn a challenging behavior in an environment with sparse rewards can be problematic. A better approach may be to learn a set of policies operating at different time resolutions that work together to learn some behavior.

## Acknowledgements

This research was supported by NSF award IIS-1724237 and by DARPA.

## References

Andrychowicz, Marcin, Crow, Dwight, Ray, Alex, Schneider, Jonas, Fong, Rachel, Welinder, Peter, McGrew, Bob, Tobin, Josh, Pieter Abbeel, OpenAI, and Zaremba, Wojciech. Hindsight experience replay. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5055–5065. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7090-hindsight-experience-replay.pdf.

Dayan, Peter and Hinton, Geoffrey E. Feudal reinforcement learning. In Hanson, S. J., Cowan, J. D., and Giles, C. L. (eds.), *Advances in Neural Information Processing Systems 5*, pp. 271–278. Morgan-Kaufmann, 1993. URL http://papers.nips.cc/paper/714-feudal-reinforcement-learning.pdf.

Dietterich, Thomas G. The maxq method for hierarchical reinforcement learning. In *In Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 118–126. Morgan Kaufmann, 1998.

Kulkarni, Tejas D, Narasimhan, Karthik, Saeedi, Ardavan, and Tenenbaum, Josh. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3675–3683. Curran Associates, Inc., 2016.

Lillicrap, Timothy P., Hunt, Jonathan J., Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL http://arxiv.org/abs/1509.02971.

Schaul, Tom, Horgan, Daniel, Gregor, Karol, and Silver, David. Universal value function approximators. In Bach, Francis and Blei, David (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/schaul15.html.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence Journal*, 112:181–211, 1999.

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

Vezhnevets, Alexander Sasha, Osindero, Simon, Schaul, Tom, Heess, Nicolas, Jaderberg, Max, Silver, David, and Kavukcuoglu, Koray. Feudal networks for hierarchical reinforcement learning. *CoRR*, abs/1703.01161, 2017. URL http://arxiv.org/abs/1703.01161.