Hierarchical Actor-Critic

Andrew Levy  
Boston University  
levya@bu.edu

Robert Platt  
Northeastern University  
rplatt@ccs.neu.edu

Kate Saenko  
Boston University  
saenko@bu.edu

Abstract

We present a novel approach to hierarchical reinforcement learning called Hierarchical Actor-Critic (HAC). HAC aims to make learning tasks with sparse binary rewards more efficient by enabling agents to learn how to break down tasks from scratch. The technique uses a set of actor-critic networks that learn to decompose tasks into a hierarchy of subgoals. We demonstrate that HAC significantly improves sample efficiency in a series of tasks that involve sparse binary rewards and require behavior over a long time horizon.

1 Introduction

The ability to break down problems into a set of simpler subproblems may help overcome one of the main challenges in deep reinforcement learning, sample efficiency. Despite major successes in both simulated and real-world tasks, a key problem with deep reinforcement learning algorithms is that they are slow. One major reason learning is slow is that reinforcement learning agents generally learn backwards. If the agent takes an action that produces a large positive reward, the agent will learn that taking that action in the current state was a favorable choice. Given that the last action was an advantageous decision, the agent will then learn that taking the prior action in the previous state was also a good decision, and so on. Learning complex behavior that requires a long sequence of actions can therefore be a lengthy process. Sparse and binary reward functions in which the reward is granted only after the last action further exacerbate this problem by forcing the agent to learn action sequences from end to beginning. Yet sparse and binary reward functions are desirable because they require less time and domain expertise and are less likely to produce unintended behavior \cite{1}. In this paper, we introduce a novel approach to hierarchical reinforcement learning called Hierarchical Actor-Critic that aims to improve sample efficiency when the reward function is sparse and binary. The technique speeds up learning by enabling agents to learn how to break down tasks into easier subtasks using only their own experience.

We believe there are two main reasons why the ability to break down complex problems into increasingly simple subproblems may improve sample efficiency. First, dividing tasks into easier subtasks results in shorter sequences that need to be learned backwards. For instance, breaking down a task that generally requires 100 actions into 10 blocks consisting of 10 actions each results in two sequences that need to be learned: (i) the sequence of 10 subgoal blocks to achieve the end goal and (ii) the sequence of 10 actions to achieve a subgoal block. However, each of these sequences is significantly shorter than the original sequence. For actor-critic approaches that derive a policy from a Q-function, it should take less time to shape a Q-function for a policy that requires a shorter sequence of actions. Second, the ability of an agent to explore over different time scales should improve learning. In addition to exploring at the individual action level, agents using Hierarchical Actor-Critic can explore the state space at a higher-level by adding noise to subgoals. This enables agents to more quickly determine the distant states that are helpful in achieving certain end goals. A faster and more thorough exploration of the state space of the environment should speed up the process of learning a robust policy.

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 task environments include pick-and-place and

\cite{1}
reach environments of different levels of complexity. All environments involved continuous action
spaces. 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. Links to videos of our experiments can be found
in the Experiments section.

2 Background

Hierarchical Actor-Critic builds off two techniques from the deep reinforcement learning literature:
(i) the Deep Deterministic Policy Gradient (“DDPG”) learning algorithm [7], and (ii) Hindsight
Experience Replay (HER) [1].

DDPG serves as the key learning infrastructure within Hierarchical Actor-Critic, enabling agents to
learn by trial and error. DDPG is an actor-critic model and thus uses two neural networks to learn
from experience. The actor network represents the agent’s policy and maps from states to actions.
Our agents will be trying to learn to achieve goals similar to the universal policies introduced by
Schaul et al. [9]. As a result, the actor function for our agents will map from the state and goal
spaces to actions (π : S × G → A) or lower-level subgoals, as described later. The critic network
is responsible for helping the agent better understand the long-term reward of taking an action or
proposing a subgoal in a given state and with a given goal. In our case, the critic network will map
from (state, goal, action or subgoal) tuples to expected long-term reward (Q : S × G × A → R).

Agents using an off-policy, actor-critic model, such as DDPG, learn in two phases. The first phase is
exploration. Agents will explore by adding noise to the actions prescribed by their actor function.
They will then store their experiences, typically tuples of (state, action, reward, next state) in a
replay buffer. The second phase is when the agent learns from experience. Agents first update their
assessment of the long-term value of taking an action, a_t, in a state, s_t, based on their past experience.
This step is completed by sampling transitions from the replay buffer and updating the Q-function,
Q(s_t, a_t), in the direction of the target y_t = r_t + γQ(s_{t+1}, π(s_{t+1})). In other words, the agent
will modify its previous assessment of the long-term value of taking an action in a certain state,
Q(s_t, a_t), so that it is closer to the sum of (i) the immediate reward, r_t, of taking the action in the
present state and (ii) the long-term value of taking the action prescribed by the policy in the next
state, Q(s_t, π(s_{t+1})). This 1-step Q-update is the mechanism by which the agent learns backwards.
If the action prescribed by the policy in the next state, π(s_{t+1}), produces the sparse, binary reward,
the 1-step Q-update will enable the agent to learn that the second-to-last action is also beneficial.
After updating the Q-function to gain a better understanding of the long-term values of taking certain
actions in certain states, the agent will next update its policy to promote actions in states with higher
Q-values. The actor function is trained by moving its parameters in the direction of the gradient of Q
w.r.t. the actor’s parameters.

Hindsight Experience Replay is a second idea from the deep reinforcement learning literature that
is integral to Hierarchical Actor-Critic. HER is a technique that aims to speed up the learning of
goal-based policies when the reward function is sparse and binary. A key issue with sparse, binary
reward functions is that they can be problematic when the desired goal is difficult to achieve. If the
desired behavior is complex and the agent is not able to achieve the goal with a certain regularity,
the agent may not receive enough reward signal to eventually learn how to achieve the task. More
specifically, it will be difficult for the agent to propagate Q-values from end of a successful sequence
to the beginning if there is only a small number of transitions in the replay buffer that contain the
positive binary reward. HER tries to overcome this issue of weak reward signals by enabling the agent
to learn more from its mistakes. 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. The following example should show
why this is helpful. Consider an agent in a reacher environment in which the goal is to touch a certain
point in space. The agent has failed to achieve its goal in several consecutive episodes, missing high,
low, left and right. Without hindsight experience replay, the agent will only learn that these actions
were not helpful in achieving its goal. However, with hindsight experience replay, the agent learns
the sequences of actions required to achieve goals that are above, below, left, and right of the original
goal. With this better understanding of how to achieve surrounding goals, the agent’s policy should
be better-tuned to achieve the original goal. Hindsight Experience Replay is implemented by creating
3 Hierarchical Actor-Critic

In traditional deep reinforcement learning approaches, a single neural network is tasked with learning the lengthy sequence of actions that can solve a problem. The idea behind Hierarchical Actor-Critic is that a stack of neural networks, in which each network is responsible for learning how to break down problems belonging to a different time scale, may learn lengthy sequences more efficiently. Further, HAC attempts to learn this hierarchical policy solely from experience.

3.1 Motivating Example

The following example should provide some intuition for how Hierarchical Actor-Critic can improve sample efficiency. The example involves an environment with a discrete action space. Our experiments for this paper involved only environments with continuous action spaces, but the algorithm should apply equally well to environments with discrete action spaces.

Consider a grid world environment in which the agent’s goal is to reach a certain \((x,y)\) block in the environment, and the agent’s reward function is sparse and binary. After any action, the agent receives 10 points if the goal is achieved and 0 points otherwise. Figure 1 (Left) shows a successful sequence of actions, in which the agent achieves its goal in 100 steps.

A reinforcement learning algorithm would generally learn from this experience by first updating its Q-function. At a high-level, updating the Q-function typically follows a backwards sequential process like that shown in bottom row of Figure 1 (Left). Beginning from the end of the successful sequence, the agent would learn that given that the last action was beneficial, taking the second-to-the-last action in the current state and with the current goal was also helpful and would increment the Q-value for that \((\text{state}, \text{goal}, \text{action})\) tuple. Given that the second-to-the-last action was successful, the agent would then learn that taking the third-to-last action was also useful and increment the Q-value for that respective tuple, and so on. The problem with this approach is that for lengthy sequences, it can take a while to propagate the Q-values backwards. The longer it takes to properly mold the Q-function, the longer it will take to learn a robust policy function.

The Hierarchical Actor-Critic algorithm attempts to overcome the sample efficiency issue by replacing the single policy mapping from states and goals to actions with a compositional policy. A simple compositional policy may have two components: a “high-level” component and a “low-level” component. Given an initial state and goal, the high-level component of the compositional policy would output a subgoal that should be first reached in order to eventually hit the end goal. This subgoal would then be passed to the low-level policy, which would subsequently execute the actions to reach that subgoal. With a compositional policy, the low-level policy no longer needs to learn the original lengthy sequence of actions. Instead, the low-level policy can focus on learning the shorter sequences of actions that achieve easier subgoals. The sequence of subgoals required to achieve the

Figure 1: (Left) Example Grid World Sequence (Right) Faster Q-update.

a separate copy of the transitions that occurred in an episode and replacing the original goal with the goal that was achieved in hindsight.
end goal can then be delegated to the higher-level component. Figure 1 (Right) shows why this is important for improving sample efficiency. Consider a sequence of 30 actions and an agent that uses hindsight experience replay, meaning the agent will learn regardless of whether the original goals or subgoals were achieved or not. Since the low-level network does not need to learn entire sequences, we may want the low-level network to learn sequences of 10 actions instead. As a result, the sequence of 30 actions can be fed to the low-level component in 3 batches of 10. Now, instead of having to propagate the ending positive reward back about 30 steps, the positive rewards only need to be pushed back about 10 steps. The high-level component would then be provided with the short sequence of 3 subgoals, in which each subgoal is the ending state of each of the 3 batches, that resulted in achieving the end goal. The high-level component can then update its own Q-function with just a few steps back. Thus, the Hierarchical Actor-Critic algorithm may be able to improve sample efficiency by enabling the Q-function update process to occur more quickly.

3.2 Architecture

Hierarchical Actor-Critic is an algorithm that enables agents to learn from experience how to break down tasks into simpler subtasks. Similar to the traditional actor-critic approach used in goal-based learning, the ultimate aim is to find a robust policy function that maps from the state and goal space to the action space. However, whereas the traditional approach uses one neural network to approximate this function, Hierarchical Actor-Critic uses a stack of neural networks like the one shown in Figure 2 (Left). Each network is responsible for learning how to break down goals at a different time scale. For instance, the top actor network in Figure 2 (Left) has similar responsibilities to the traditional actor network. However, instead of learning the sequence of actions (e.g., joint torques) required to get to the end goal, the network is only responsible for learning the shorter sequence of actions to get to the input subgoal. The bottom actor network in Figure 2 (Left) is then responsible for learning the sequence of subgoals to get to the end goal. Figure 2 (Left) shows a hierarchical actor with 1 subgoal layer. Additional networks can be stacked on to the hierarchical actor enabling each network to be responsible for learning a shorter sequence of steps. In our experiments, we examine hierarchical actors with 1 and 2 subgoal layers.

The individual actor networks within the larger hierarchical actor are trained in the same manner as a traditional actor network. Each actor network has an accompanying critic network, which learns a Q-function that attempts to learn how good an action (or lower-level subgoal, if the actor network is
Algorithm 1 Hierarchical Actor-Critic algorithm

1: Given:
   Desired number of subgoal layers \( n \)
   Desired Sequence Length Per Layer: \( T_0(\text{no. actions}/g_1), \ldots, T_{n-1}(\text{no. actions}/g_{n-1}), T_n(\text{no. actions}/\text{Goal}) \)
   Subgoal Testing Strategy
   Function mapping state, \( s \), to hindsight goal achieved, \( g' : s \rightarrow g' \)

2: Initialize Actor-Critic networks \([ (\pi_0, Q_0), \ldots, (\pi_n, Q_n) ] \]
3: Initialize Replay Buffers \(( R_0, \ldots, R_n ) \)
4: for episode = 1,M
5:   Sample actual goal, \( G \), and initial state, \( s_0 \)
6:   for \( t_n = 1, T_n \)
7:     Determine whether subgoals should be tested
8:     Sample layer \( n \) subgoal from \( \pi_n \)
9:     \( g_n \leftarrow \pi_n(s(t_n), G) + \text{noise} \) (if not testing)
10:   for \( t_{n-1} = 1, T_{n-1} \)
11:     Sample layer \( n-1 \) subgoal from \( \pi_{n-1} \)
12:     \( g_{n-1} \leftarrow \pi_{n-1}(s(t_{n-1}), g_n) + \text{noise} \) (if not testing)
13:     \ldots
14:   for \( t_0 = 1, T_0 \)
15:     Sample action from \( \pi_0 \)
16:     \( a \leftarrow \pi_0(s(t_0), g_1) + \text{noise} \) (if not testing)
17:     Determine next state, \( s(t_0 + 1) \), hindsight goal achieved, \( g' \), and reward, \( r(t_0) \)
18:     Add non-HER transition to \( R_0 \)
19: end for
20: Add HER transitions to \( R_0 \)
21: \ldots
22: Determine next state, \( s(t_{n-1} + 1) \), hindsight goal, \( G' \), and reward, \( r(t_{n-1}) \)
23: Add non-HER transition to \( R_{n-1} \)
24: end for
25: Add HER transitions to \( R_{n-1} \)
26: Determine next state, \( s(t_n + 1) \), hindsight goal, \( G' \), and reward, \( r(t_n) \)
27: Add non-HER transition to \( R_n \)
28: end for
29: Update Actor-Critic Networks \([ (\pi_0, Q_0), \ldots, (\pi_n, Q_n) ] \) with off-policy RL Algorithm
30: end for

for a subgoal layer) given the input state and higher-level goal. The actor networks in Figure 2 (Left) are shown together with their respective critic networks in Figure 2 (Right). Each critic network has its own replay buffer, which stores the experience transitions that are used to learn the Q-function. Thus, each actor-critic layer is essentially trying to solve its own reinforcement learning problem – how to efficiently break down goals belonging to longer time scales into subgoals (or actions) belonging to shorter time scales.

3.3 Training

The aim of HAC is to train each of the networks within the stack to learn how to break down problems of different time scales using only the agent’s experience. At a high level, the idea is for the agent to learn the sequences of actions and subgoals that achieve higher-level goals by learning from past sequences of actions and subgoals and the higher-level goals these sequences did and did not achieve.

In order for each network within the stack to learn from experience how to break down problems belonging to its time scale, different sequences of transitions are passed to the replay buffer of each network. We will define transitions as belonging to the same sequence if they contain the same higher-level goal. The length of the sequences passed to each replay buffer depends on the length of the sequence the user wants each network to learn. The desired sequence lengths for each network within the stack are denoted by the constants \( T_0, \ldots, T_n \). For instance, if a full episode of a task requires around 100 actions and the user’s hierarchical network has one subgoal layer as in Figure 2 (Left), the user might have the the low-level network learn how to break down subgoals into at most
10 actions ($T_0 = 10$). The high-level network would then learn how to break down end goals into at most 10 subgoals ($T_1 = 10$). The replay buffers for both of these layers would therefore generally see series of 10 consecutive transitions with the same higher-level goal. The timing of when each transition is actually passed to its appropriate replay buffer depends on the time scale of the sequence the network is trying to learn. The low-level network, such as the top network in Figure 2[Left], is passed a transition after every action taken by the agent. The goal used in each transition will change at most every 10 actions when the low-level network is given a new subgoal. As a result, around every 10 actions, the low-level network will have received a full sequence. The higher-level network, which is trying to learn efficient sequences of subgoals that achieve end goals, will only receive a new transition around every 10 actions because each subgoal consists of 10 actions. The goal within each of the high-level network’s transitions will change at most every 10 subgoals. Thus, the replay buffer will typically have received a full sequence after 10 subgoals or 100 individual actions. The key point is that each network will be likely to learn how to break down problems into sequences of size 10 because that is the typical length of sequences they will be given.

Testing whether subgoals can actually be achieved by the current hierarchical policy has proven to be another important strategy toward developing effective subgoal policies. With traditional, flat actor functions that do not include any subgoal layers, the validity of the action that is output by the policy is not a concern. By design, the agent is able to execute any action, such as joint torques, that the policy outputs. However, when subgoal layers are used, the subgoals output by subgoal layers may not be achievable by the agent’s policy and this can be problematic. Hierarchical Actor-Critic purposely limits the sequence length that each layer needs to learn in order improve the efficiency of learning. One downside of this strategy is that if the network is provided a goal that is too far away, the subgoal or action layer may not know how to achieve it. If subgoals are frequently too difficult to achieve, the agent will then have less data from which to learn which sequences of subgoals are better than others. The agent will consequently be less likely to learn how to break down the problem at a high-level, diminishing the efficiency advantages that HAC brings. Indeed, off-policy reinforcement learning algorithms, such as DDPG, will often be biased towards producing unrealistic subgoals. These algorithms try to learn policies that achieve goals as quickly as possible. For subgoal actor-critic layers, this means the agent will try to learn how to achieve higher-level goals with as few subgoals as possible based on what the agent has experienced. If the agent’s experience consists entirely of transitions in which agent has added noise to every action and subgoal, there will be many sequences of subgoals that, by chance, do efficiently achieve higher-level goals. Regardless of whether or not the agent’s policy can actually achieve these ambitious subgoals, the reinforcement learning algorithm will skew the subgoal actor towards outputting them.

To overcome this issue of unrealistic subgoals, the subgoals are occasionally tested to see if they can be obtained with the current policy. More specifically, a certain percentage of the time during training, subgoals are tested by not adding noise to any subgoals or actions. Any subgoals that cannot be achieved by the current policy will be penalized by the transitions passed to the replay buffers during testing. The transitions will indicate that the policy’s proposed sequence of actions or subgoals to achieve a higher-level subgoal yields a low reward. When the actor and critic networks are subsequently updated, the policy should be less likely to propose the same subgoals in the future.

The Hierarchical Actor-Critic algorithm works as follows. The inner for loops represent the exploration phase of the learning process. The agent adds noise to proposed subgoals and actions and then records the experience in transitions that are passed to the appropriate replay buffer. More specifically, each nested for loop except for the innermost nested for loop represents a subgoal layer. The innermost for loop represents the layer that outputs actual actions. For each nested for loop $i$, the following pattern occurs. The completion of the for loop that is nested within for loop $i$, for loop $i - 1$, produces one non-HER transition for subgoal layer $i$. For loop $i - 1$ ends when $T_{i-1}$ iterations of the loop have occurred or a higher-level goal, $g_1, ..., G$, has been achieved. Thus, the lowest-level actor network will receive a transition after every action. The subgoal layer one level above will receive a transition every $T_i$ actions or earlier if any of the higher-level goals, $g_1, ..., G$, have been achieved. HER transitions for layer $i$ are added to $R_i$ when for loop $i$ terminates. Please see the Appendix for a detailed description of the contents of the transitions.

The termination of the inner for loops indicates that the episode and exploration phase have ended, and the agent will now learn from its experience. Each network will sample from its own replay buffer and update its own Q-function. Each network will subsequently update its actor function. In
Figure 3: Figure shows three key frames from a successful episode of the UR5 Reacher task. Agent learns to reach goal (yellow cube) by breaking down the task into high-level subgoals (green cubes) and by breaking down the high-level subgoals into low-level subgoals (purple cubes).

Figure 4: Figure shows three key frames from a successful episode of the 1-Object Pick-and-Place task. Agent learns to pickup the blue rod and carry it to the yellow rod by breaking down the task into high-level subgoals (green markers) and by breaking down the high-level subgoals into low-level subgoals (purple markers).

Our implementation, we updated lower-level networks more frequently as these replay buffers receive more transitions per episode.

4 Experiments

We evaluated the Hierarchical Actor-Critic approach on three reacher and pick-and-place tasks. 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. Videos showing our results on the three environments can be seen below.

- UR5 Reacher: [https://www.youtube.com/watch?v=4qwCh LiZSc](https://www.youtube.com/watch?v=4qwCh LiZSc)
- 1-Object Pick-and-Place: [https://www.youtube.com/watch?v=K9PzNGF7rY8](https://www.youtube.com/watch?v=K9PzNGF7rY8)
- 2-Object Pick-and-Place: [https://www.youtube.com/watch?v=fcqQwvPmZIU](https://www.youtube.com/watch?v=fcqQwvPmZIU)

4.1 Environments

We considered the following three environments. Each of these simulations was built using the Mujoco physics engine [11].

1. 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. Figure 3 shows a few frames from a successful episode. We found than an efficient policy could solve this task in around 60 individual actions.

2. 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. Figure 4 shows a few frames from a successful episode. Our more efficient agents can solve this task in around 110 steps.

3. 2-Object Pick-and-Place
   This task is a harder version of the 1-Object pick place. Now there rectangular and cylindrical blue and yellow rods. The goal is to pick up each blue rod and place it on the yellow rod of
the same shape. The agent can only pick up one rod at a time. We found that a reasonably efficient policy can solve this task in 220 steps on average.

**States, Actions, Rewards**
The state space in all three environments include joints positions and velocities. The pick-and-place environments also provide the agents with binary flags to indicate when the blue rod has been attached and has been placed correctly on the yellow rod.

The actions in the reacher and pick-and-place tasks are joint torques. In the 1-object pick-and-place task, the agent has an additional action enabling the agent to pick up or drop off a blue rod. If the value for this action is above a certain positive threshold and a blue rod is nearby, the agent will pick up the blue rod. If the value for this action is below a certain negative threshold and the blue rod is attached, the agent will drop the blue rod. This extra dimension is removed in the 2-object pick-and-place environment in order to make the task easier. In the 2-object pick-and-place task, the agent picks up a blue rod by moving within a certain distance of the rod. The rod is then automatically released if the blue rod is moved within a certain distance of the correct yellow rod.

Each network within the stack has its own sparse, binary reward function. 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 binary reward for taking that action given the current state and subgoal because the action was unsuccessful.

**Goals and Subgoals**
The goals and subgoals in these environments can be interpreted as desired states the agent would like to achieve. In our experiments, the goals and subgoals are never actually the full state, but rather a lower-dimensional version. In the UR5 reacher task, both the end goals and subgoals are the desired \((x,y,z)\) position for the end effector. The end goals in the 1 and 2-object pick place tasks are the desired \((x,y)\) final positions of the blue rods, which are at the yellow rods. The subgoals in the pick-and-place environments include (i) the desired positions of the blue rods and gripper and (ii) the desired attachment status of the blue rods. The desired attachment status is essentially a continuous version of a binary flag. Discrete binary flags cannot be part of the subgoal because subgoals are outputs of actor functions, and actor functions must be differentiable in order to use off-policy RL algorithms such as DDPG.

In our experiments, we tested agents with 0, 1, and 2 subgoal layers, which are visualized in the videos as follows. No subgoal markers will appear in the environment when the agent has 0 subgoal layers. When the agent uses 1 subgoal layer, additional purple markers representing the subgoal will appear in the environment. In the UR5 reacher environment, the additional marker will be a purple cube in the \((x,y,z)\) subgoal position. In the pick-place environments, there will be multiple purple markers when the agent has 1 subgoal layer. The purple sphere represents the desired subgoal position of the gripper. The short purple rectangular and cylindrical rods mark the desired subgoal positions for the blue rods of the same shape. When the agent uses 2 subgoal layers, the same markers in green are added representing the higher-level subgoal. All purple and green subgoal markers are strictly added for visualization purposes. They have no effect on the actual physics of the environment.

### 4.2 Results

The accompanying videos and Figure 5 show the results of our experiments. In Figure 5, each chart plots the percentage of goals achieved by agents using 0, 1, and 2 subgoal layers in a given testing period. Testing periods are separated by about 300 episodes. Agents learn each task from scratch and thus begin with no pre-training.

The key result from our work is that the use of Hierarchical Actor-Critic resulted in a substantial improvement in performance. In all three environments, agents that used HAC learned a robust policy significantly faster. Indeed, the videos show that in the UR5 Reacher and 2-Object Pick-and-Place environments, in particular, only using DDPG and HER resulted in very little progress towards learning a robust policy.

The videos and the frames in Figures 3 and 4 also demonstrate that the algorithm is working as intended – the agents are learning how to break problems down into easier subproblems. In the UR5
Testing Period
% Goals Achieved
UR5 Reacher

![Graph](image1)

Figure 5: HAC performance on three tasks. The figures show percent of goals achieved over the testing period for DDPG+HER (brown), 1-subgoal HAC (red) and 2-subgoal HAC (blue).

Testing Period
% Goals Achieved
1-Object Pick Place

![Graph](image2)

Figure 6: HAC performance with different subgoal testing percentages: 0% (blue), 20% (red), 60% (brown)

Reacher environment, if the goal location marked by the yellow cube is far away from an agent using 2 subgoal layers, the agent would often put the high-level subgoal marker, the green cube, about halfway to the end goal, like in the first frame of Figure 5. The low-level subgoals, marked by the purple cubes, then appear to guide agent toward the high-level subgoal, which can be seen in the second and third frames of Figure 5. This indicates the agent has learned how to effectively break down 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 rod, will often appear in the vicinity of the blue rod when the blue rod has not yet been picked up, as in the first frame of Figure 4. Once the blue rod has been attached, the high-level subgoal moves to the yellow rod, like in the final frame of Figure 4. The low-level subgoal, represented by the purple sphere and rod, again guide the agent to the green high-level subgoal. A similar pattern happens in the 2-Object Pick-and-Place environment. Initially, the low-level and high-level subgoals will lead the agent to one of the blue rods and then to the correct yellow rod. Immediately after one blue rod has been correctly placed, both subgoals then work to guide the agent to the other blue rod.

Our experiments also confirmed the importance of testing subgoals. Figure 6 shows the performance of agents in the 1-Object Pick-and-Place environment using 2 subgoal layers and subgoal testing percentages of 0%, 20%, and 60%. The figure shows that learning is drastically slowed when subgoals are not tested at all. This result confirms that agents will struggle to learn how to break down problems if unrealistic subgoals are not penalized through testing.

Our results did not yield any definitive conclusions on whether additional subgoal layers improve the speed of learning. Agents using 2 subgoal layers appeared to outperform agent using 1 subgoal layers in the UR5 task. However, the results were reversed in the 1-Object Pick-and-Place task. The advantage of additional subgoal layers is that each subgoal layer is responsible for learning shorter sequences. On the other hand, additional subgoal layers do result in more sequences that need to be learned. This may be problematic as higher-level subgoal layers may need to learn their sequences before the lower-level layers can fully learn theirs.
5 Related Work

Hierarchical RL is a topic of ongoing research [10], [3], [12], [4], [2], [8], [5]. Much of the literature builds off the options framework [10], which 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 the 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. Using one low-level network instead of several to learn low-level policies should provide some efficiency advantages because learning how to achieve one subgoal will often help in learning how to achieve different subgoals.

Kulkarni et al.[6] proposed a similar approach to HAC, named hierarchical-DQN (h-DQN), which 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. 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, binary reward functions. In the paper’s ‘Montezuma’s 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. On the other hand, the agents in the examples presented in this paper had to learn both the subgoal states and their order solely from experience while only using sparse, binary reward functions. The 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, which enables the agent to learn more from failure. 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.

The use of Hindsight Experience Replay [1] on its own can be viewed as a type of hierarchical learning. On the way to achieving a distant goal, the agent may learn to achieve intermediary goals in hindsight. Learning these easier goals may lead the agent to learning the distant goal. However, HER does not provide a way to tie these easier subgoals together so the agent’s actor network is still forced to learn a long sequence of actions.

6 Conclusion

We introduced a new technique called Hierarchical Actor-Critic that enables agents to learn from experience how to break down problems into easier subproblems. Our results indicate that only using one neural network to learn a long sequence of actions in an environment with sparse binary rewards can be problematic. A better approach may be to use a set of neural networks that work together to learn a hierarchical policy, thereby reducing the sequence length that each neural network needs to learn.

7 Acknowledgements

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

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. CoRR, abs/1707.01495, 2017.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. CoRR, abs/1609.05140, 2016.
was not achieved. In this case, the transition whether the original proposed subgoal was achieved or not. If the proposed subgoal $g^i$ has not been achieved, the hindsight goal achieved during that transition, $g^i_{t+1}$, will replace $g^i_t$ in the transition.

The reason for this change is that the Q-value of a subgoal depends on the Q-value of the subgoal prescribed in the next state as defined in the 1-step Q-update equation. If the next state is not near the original subgoal state $g^i_t$, then keeping $g^i_t$ in the transition would make the estimate $Q(s(t_i), r(t_i), s(t_{i+1}))$ inaccurate. Additional modifications are made to transitions if the agent is in testing mode and $g^i_t$ was not achieved. In this case, the transition $Q(s(t_i), r(t_i), N/A, g^i_{t+1})$ is added to layer $i$’s replay buffer. The next state value is N/A because the Q-value for the missed subgoal, $Q(s(t_i), g^i_t, g^i_{t+1})$, should not depend on the Q-value of the next state that was actually reached because this state is different than the subgoal state the agent intended to be at.

The HER transitions are created as follows. For each non-HER transition, a separate transition is made and temporarily stored with the following contents: $(s(t_i), g^i_{t+1}, TBD, s(t_{i+1}), TBD)$. The original goal, $g^i_t$, has been replaced with the hindsight goal, $g^i_{t+1}$. The reward and input have also been removed. These will be updated when a new input goal has been added. When for loop $i$ has...
finished, either because it has finished $T_i$ iterations or the agent has achieved a higher-level goal, $g_{i+1}, \ldots, G$, hindsight experience replay begins.

During HER, a certain number of the $T_i$ transitions within the batch will be chosen to serve as hindsight goals. All transitions leading up to and including the selected transition will have the goal element, currently TBD, changed to the hindsight goal achieved during the selected transition. The reward element in each transition will also be replaced with the appropriate value given the new goal. The completed transitions will then be placed in layer $i$’s replay buffer.