Hindsight Experience Replay
Practice Environment

Siddharth Ancha, Nicholay Topin
MLD, Carnegie Mellon University
(10-703 Recitation Slides)
Environment (states)

Goal
(random initial location within boundary)
(does not move during episode)

Box
(fixed initial position)
(can be pushed by pusher)

Pusher
(fixed initial position)
(directly controlled by agent)

Each state is of form:

\((X_{\text{pusher}}, Y_{\text{pusher}}, X_{\text{box}}, Y_{\text{box}}, X_{\text{goal}}, Y_{\text{goal}})\)
Environment (transitions)

- Each action is of form:
  \[(X_{\text{movement}}, Y_{\text{movement}})\]
- Moves pusher proportional to values
- Box moves if pusher collides with it
Environment (rewards)

• Uniform reward for non-terminal step (living penalty of -1)

• Terminates if out of bounds (prorated negative reward)

• Terminates if box touches goal (0 reward)

• Also terminates after “max steps” (same -1 living penalty)
HER Motivation

• 2D Pusher environment has sparse reward
• Random actions rarely push box into goal
• As a result, most tuples have -1 reward (few “informative” tuples)

• Even though agent is not getting to goal, it is getting *somewhere*
• Could learn how to reach desired state of world from arbitrary reached states
• Main idea: Create new trajectory with new goal which is reached in trajectory
Algorithm 1 Hindsight Experience Replay (HER)

Given:
- an off-policy RL algorithm $\mathbb{A}$,
- a strategy $\mathbb{S}$ for sampling goals for replay,
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$.

Initialize $\mathbb{A}$
Initialize replay buffer $R$

for episode = 1, $M$ do
    Sample a goal $g$ and an initial state $s_0$.
    for $t = 0, T - 1$ do
        Sample an action $a_t$ using the behavioral policy from $\mathbb{A}$:
        \[ a_t \leftarrow \pi_b(s_t || g) \]
        Execute the action $a_t$ and observe a new state $s_{t+1}$
    end for

\[ \triangleright \text{Standard DRL} \]

\[ \triangleright \text{e.g. DQN, DDPG, NAF, SDQN} \]
\[ \triangleright \text{e.g. } \mathbb{S}(s_0, \ldots, s_T) = m(s_T) \]
\[ \triangleright \text{e.g. } r(s, a, g) = -[f_g(s) = 0] \]
\[ \triangleright \text{e.g. initialize neural networks} \]
\[ \triangleright || \text{ denotes concatenation} \]
HER Pseudocode (2)

for $t = 0, T - 1$ do
  $r_t := r(s_t, a_t, g)$
  Store the transition $(s_t \parallel g, a_t, r_t, s_{t+1} \parallel g)$ in $R$  
  Sample a set of additional goals for replay $G' := \mathcal{S}(\text{current episode})$
  for $g' \in G'$ do
    $r' := r(s_t, a_t, g')$
    Store the transition $(s_t \parallel g', a_t, r', s_{t+1} \parallel g')$ in $R$
  end for
end for

for $t = 1, N$ do
  Sample a minibatch $B$ from the replay buffer $R$
  Perform one step of optimization using $A$ and minibatch $B$
end for
# returns list of new states and list of new rewards for use with HER

def apply_hindsight(self, states, actions, goal_state):
    goal = goal_state[2:4]  # get new goal location (last location of box)
    states.append(goal_state)
    num_tuples = len(actions)
    her_states, her_rewards = [], []
    states[0][-2:] = goal.copy()
    her_states.append(states[0])
    # for each state, adjust goal and calculate reward obtained
    for i in range(1, num_tuples + 1):
        state = states[i]
        state[-2:] = goal.copy()
        reward = self._HER_calc_reward(state)
        her_states.append(state)
        her_rewards.append(reward)
    return her_states, her_rewards
action, q = agent.pi(obs, apply_noise=True, compute_Q=True)
assert action.shape == env.action_space.shape

new_obs, r, done, info = env.step(max_action * action)
t += 1
episode_reward += r
episode_step += 1
agent.store_transition(obs, action, r, new_obs, done)

# storing info for hindsight
if kwargs["her"]:
    states.append(obs.copy())
    actions.append(action.copy())

obs = new_obs

if done:
    [...]

Implementation (HER change)

[...] if done:
    if kwargs["her"]:
        # create hindsight experience replay
        her_states, her_rewards = 
            env.env.apply_hindsight(states, actions, new_obs.copy())
        # store her transitions: her_states: n+1, her_rewards: n
        for her_i in range(len(her_states)-1):
            agent.store_transition(her_states[her_i], actions[her_i], 
                                    her_rewards[her_i], her_states[her_i+1], 
                                    her_rewards[her_i] == 0)
[perform memory replay]
Parameters

- We used OpenAI Baselines DDPG
- Batch size = 128
- Gamma = 0.98
- Learning rate (actor) = 1e-4
- Learning rate (critic) = 1e-3
- Noise = epsilon normal action noise (0.01, 0.2)
- Architecture (actor and critic) = 3 hidden layers each, 64 nodes each
- Num actors = 8
- Max rollout steps = 320
Comparison Plots

![Comparison Plot](image)

**Axes:**
- **Y-axis:** Portion of 20 Evaluation Episodes where Task Completed
- **X-axis:** Epochs (800 steps each)