## Reinforcement Learning for Solving Long-Horizon Robot Problems

### Introduction

Reinforcement learning (RL) is a subfield of machine learning in which agents select actions to maximize their reward. Just as a human slowly learns the right way to kick a soccer ball because they get a thrill from scoring a goal, or a dog learns the right way to beg to more frequently obtain a treat, RL agents modify their behavior to gain more reward. RL has demonstrated success in domains ranging from robot manipulation to video games. The most challenging aspect of RL is that, in many cases, reward only follows after a very long sequence of actions—for example, obtaining a bachelor's degree after four years of completing assignments. Learning in such delayed reward situations (known as *long-horizon* problems) is fundamentally hard; there is simply no way around it.

Hierarchical RL algorithms decompose long-horizon tasks into smaller subgoals—like completing a course, or finishing junior year—so that, at a high level of abstraction, the agent need make only a few successive correct choices before receiving its reward. This simplifies the task to: 1) finding the right subproblems, 2) learning solutions (often called *skills*) to each subproblem, and 3) sequencing skills to solve the entire task. Deep Skill Graphs (DSG) is an algorithm that addresses all three steps [1].

DSG has been successful on navigation tasks, in which the agent must simply move through space, but my project involved applying DSG to manipulation tasks, where the robot must use its gripper to modify the world around it. The specific task I was trying to solve required pushing a hockey puck to a target location on a table using a simulated Sawyer robot arm (Figure 1).

Sawyer is significantly harder than the previous maze navigation tasks DSG was applied to because of complex robot arm-puck interactions. To push the puck to the left, the robot arm must get to the right of the puck without disturbing the puck's position; this is exactly the sort of long-horizon planning that is difficult for traditional RL algorithms! Therefore, solving the Sawyer task would be a substantial breakthrough in the RL literature.

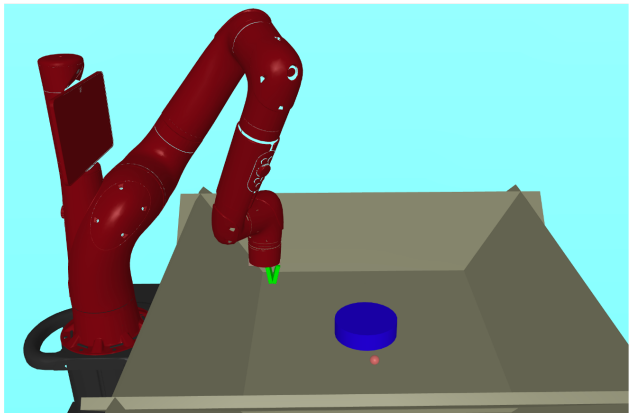

Figure 1: The Sawyer robot (dark red) pushes the puck (blue) to a target location (orange dot).

Applying DSG directly to the Sawyer domain does not work. In fact, it doesn't learn a single skill. I applied two different approaches to attack this problem: improving the parts of DSG that learn skills and better reusing training data across skills.

### Improving Algorithm to Learn Skills

The first task was learning a single skill (a sample skill might be pushing the puck to a corner). However, this requires hundreds of correct actions in a row, just the problem we were trying to avoid! I approached this by modifying the agent's exploration algorithm. Most RL algorithms begin by taking random actions (i.e. exploring); once an agent receives positive reward, that behavior is reinforced. I added several parameters that increased the randomness of initial actions

but slowly decreased the randomness once the skill was somewhat reliable.

After learning a single skill, the next task was chaining together different skills. A challenge I haven't mentioned yet is determining what skills can be executed from where. For example, a student can only execute the skill "complete junior year" when they are a junior.

DSG uses machine learning to identify the valid start states (also known as the *initiation set*) for each skill. The issue is that the initiation set can change as the skill is being learned. To account for the changing skill, I made the process of learning initiation sets more sophisticated. First, I modified DSG to pause training the initiation set until the skill had stabilized. Second, I used data from nearby skills to train the initiation set of the current skill.

Once DSG was able to begin building a skill graph for Sawyer, the next challenge was reducing training time. Building a complete skill graph took about three days of training time. The bottleneck was an inefficient use of training samples: most episodes fail to reach the goal state, so the agent learns nothing from those samples.

A RL technique called goal-conditioning enables more efficient learning with limited data. Suppose the algorithm is targeting state X but reaches state Y. Rather than interpreting this as a failure to reach X, goal-conditioning treats this as a successful attempt to reach Y. Every episode becomes a positive example for some target state. I helped implement a technique called Hindsight Experience Replay that uses goal-conditioning.

**Reusing Data Across Skills**

Another, more ambitious, technique to learn with limited data is reusing the same data to train different skills. Many of the skills for Sawyer overlapped (moving the robot arm from one region of the table to another might intersect the trajectories of several other skills). Therefore, the training data collected from one skill could be used to train other skills, saving training time. This is known as *off-policy learning*.

As an analogy, suppose you want to become a professional football player, but the fields are occupied (limited training time). The next best thing is watching professional football games (off-policy learning).

Unsurprisingly, I found that off-policy learning alone was not sufficient to train reliable skills; it would be hard to become a professional football player by only watching others play. However, pretraining a skill with off-policy data and then training it normally (watching others play and then practicing) also led to poor results. Therefore, I focused on the following questions, in the hopes that the findings would help me reduce DSG's training time:

1. How close do the original and off-policy goal have to be? Does watching rugby help more with football than watching swimming?
2. Are there certain domains for which off-policy learning is more or less successful? Is it easier to learn football than volleyball by watching?

To better understand the preliminary Sawyer results, I branched out to three other robot domains: Point, Swimmer, and Ant, in order from easiest to hardest. For each robot, a model agent was trained to target a goal. Then, off-policy agents were initialized with the model agent's policy and then trained to target goals in the vicinity of the original target. The off-policy agents were compared with a baseline that trained directly on the off-policy targets.

For the simpler domains such as Point and Swimmer, the off-policy runs performed better for all test goals. For the more complex Ant domain, however, the off-policy

was better than the baseline when the off-policy goal was close to the pretrained goal but significantly worse than the baseline when the off-policy goal was far away from the pretrained goal. This was consistent with the preliminary findings in the Sawyer domain and suggests that in more complex domains, the agent can fail to recover from bad initializations. Therefore, a skill should only train on off-policy data if the off-policy target is close to the skill's target (i.e. watching golf to learn to play football can make you a worse football player!).

Applying these findings to DSG is still an open problem because there is no obvious distance metric to measure how close together two goals are. In the Sawyer domain, even though the puck locations for two goals might be close in the Euclidean sense, the robot arm might have to take a drastically different trajectory (such as around the puck before pushing) to reach that goal.

**Results**

Even without the off-policy training, there were significant updates to DSG. To evaluate the success of these updates, I trained DSG for 3,000 episodes of 1,200 steps to populate the skill graph. Then, I randomly sampled five goal states and tracked DSG's success rate of reaching each goal over 250 episodes. These results were compared to a baseline of goal-conditioned RL without DSG.

DSG performs significantly better than the baseline, state-of-the-art goal-conditioned algorithm, reaffirming how challenging Sawyer is (Figure 2). Further, the upward trend suggests that DSG's success rate would increase with more training time (experiments were limited to 250 episodes due to computational constraints).

The results are especially promising because DSG only receives a sparse reward (a binary reward indicating success or failure). I was unable to find any comparisons in the literature which are successful on the Sawyer robot domain with a sparse reward. In fact, DSG performs as well as state-of-the-art algorithms that receive a much easier dense reward, which gives the agent a reward based on how far it is from the goal.

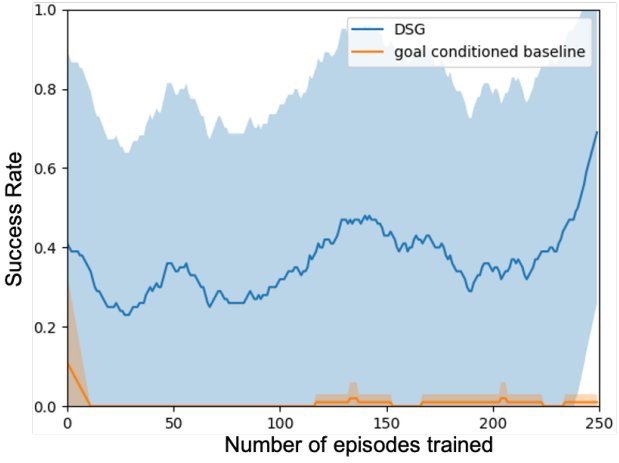


Figure 2: DSG learning curve (shaded region is standard deviation over different goals)

There is still room for improvement: if the skill graph covered the entire state space, we would expect a success rate near one. The current bottleneck is likely that a few skills have low success rates. Work is currently being done on prioritizing the training time of different skills to encourage more executions of weak skills.

**Future Work**

DSG gives agents the power of abstraction. The success of DSG in the Sawyer domain suggests that it will be a promising approach for a myriad of RL problems. Future work will explore using DSG to solve the Atari game Montezuma's Revenge, a holy grail long-horizon problem in artificial intelligence.

**References**

[1] A. Bagaria et al. "Skill Discovery for Exploration and Planning using Deep Skill Graphs". In: 2020.