CS230

Collaborative Pick and Place Between Tandem Robotic Arms with Deep Q-learning

Edward Fouad, Kerrie Wu, Zhichao Sun
Department of Computer Science
Stanford University
efouad, kerriewu, zcsun@stanford.edu
https://github.com/kerriewu/cs230-robot-manipulator

Abstract

Deep reinforcement learning offers viable solutions to unsolved challenges in robotic manipulation and multi-agent interaction. We apply deep Q-learning to a robotic manipulation task requiring collaboration between with two synchronously trained agents. A trained agent can achieve 100% on an individual one-step pick and place task, and partial success on a multi-step task involving collaboration with the partner robot. We present the performance of a deterministic policy via the trained Q-network and a non-deterministic policy that takes a random action with some probability. Our results show that stochasticity can improve performance for a dynamic, multi-agent objective.

1 Introduction

Deep reinforcement learning (DRL) is an increasingly viable solution to hard problems in robotic manipulation. Both on-policy advantage actor-critic methods (A2C) and off-policy deep Q network (DQN) and deep deterministic policy gradient (DDPG) approaches have successfully solved benchmark Atari 2600 and MuJoCo simulation games [5, 6, 7].

When learning trajectory planning algorithms for robotic manipulators, researchers often rely on knowledge of the underlying robot dynamics to perform model-based RL. Such an approach offers the advantage of being able to generate synthetic experience and learn efficiently within a simulated environment [8]; however, it is vulnerable to model inaccuracies and does not generalize across different robotic platforms. On the other hand, model-free methods require no a prior specification of the system dynamics (or even the robot's morphology) and have successfully achieved end-to-end learning from raw camera images to motor commands [9].

In this work, we use a model free approach to train a robotic arm to navigate its 3D workspace with a DQN, using no a priori knowledge of its kinematic model. Our goal is to learn a policy that, when used in collaboration with a similarly trained partner arm, successfully completes a tandem pick and place task.

2 Related work

There is a wide range of literature in robotic arms learning point-to-point movements and picking and passing objects. Hao et al.[10] adopted the Deep Deterministic Policy Gradient (DDPG) algorithm to deal with the robot physical problem of the high-dimensional continuous action space and added Hindsight Experience Replay (HER), Expert Data Initialization, and Action Clip Scaling to do tasks such as reach, push, pick and place under the sparse rewards, and in an anthropomorphic robotic arm simulation environment. Luyu et al.[11] proposed a dual-arm deep deterministic policy gradient (DADDPG) algorithm based on deep reinforcement learning of multi-agent cooperation to realize collaborative control of a dual-arm robot refers to collision avoidance and working together to accomplish a task, such as grasping and passing objects.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Simulation Environment

Two agents each control a Franka Emika Panda robotic arm that has been restricted to 4 revolute degrees of freedom: one with a vertical axis at the base and three with horizontal, mutually parallel axes along the length of the arm. The robot also has a gripper that can be either opened or closed. Range of motion limits have been removed to allow continuous revolution at each joint; however, the limbs are constrained to not pass through the ground. The simulation environment employs a continuous state space and discrete action space.

The same model is used to control each agent. Each agent begins with 5 tokens within its reach, and must work with the other agent to place the tokens in a desired location (a "bin"). Figure 1 shows the environment setup and describes the tandem passing task. Each episode is limited to 10,000 timesteps in both training and evaluation. The environment is implemented using a publicly available, preexisting python library for reinforcement learning called PettingZoo [13], and interfaces with StableBaselines3 [3] for training via SuperSuit [14]. Rendering is done using a robotics toolbox for python [16].

3.1 State space

The simulation state variable contains information about the physical environment, including the locations of each token, their desired "bins", and two "checkpoint" regions where the robots can pass their tokens across the median. It also consists of each robot's current joint angles, joint positions, and gripper contents.

3.2 Observation space

Each robot observes the full state space. This gives it perfect knowledge of its own configuration (joint positions, joint angles, gripper contents), the configuration of its partner robot, and the positions of all tokens and bins in the workspace.

Additionally, each robot observes their own and their partner's "phase". The phase is an integer flag corresponding to task the robot is actively working towards: fetch partner token (0), deposit partner token (1), fetch own token (2), deposit own token (3). This helps the agent remember which objective it is working towards and complement the activity of its partner.

3.3 Action space

At each timestep, the robot can increment or decrement any one of its joints by 5 degrees, attempt to pick up a token, or attempt to drop a token. Attempting to pick up a token when already holding a token or not near a token has no effect on the environment. Attempting to drop a token when not holding a token or not at a desired end location (checkpoint or bin) has no effect.

3.4 Reward

The reward includes a large discrete contribution for correctly picking up or dropping a token, as well as a small continuous contribution for moving towards the correct target (based on the robot's current phase). The phase target locations are the partner token stack (p_1), partner checkpoint (p_2), own checkpoint (p_3), and own bin (p_4). After the robot successfully picks up or drops a token, its phase is incremented. See Algorithm 1.

Algorithm 1 Agent One-Step Reward Function

Require: Action a; end effector location x; phase j; phase target locations p_1, p_2, p_3, p_4

Select phase target location p_j
Initialize reward $r \leftarrow -\|x - p_j\|_2$
if j = 0 or 2 (fetch token) and a = <pick up> **then**
    $r \leftarrow r$ + 1e6
    $j \leftarrow (j + 1)$ mod 4
else if j = 1 or 3 (deposit token) and a = <drop> **then**
    $r \leftarrow r$ + 1e6
    $j \leftarrow (j + 1)$ mod 4
end if
return r

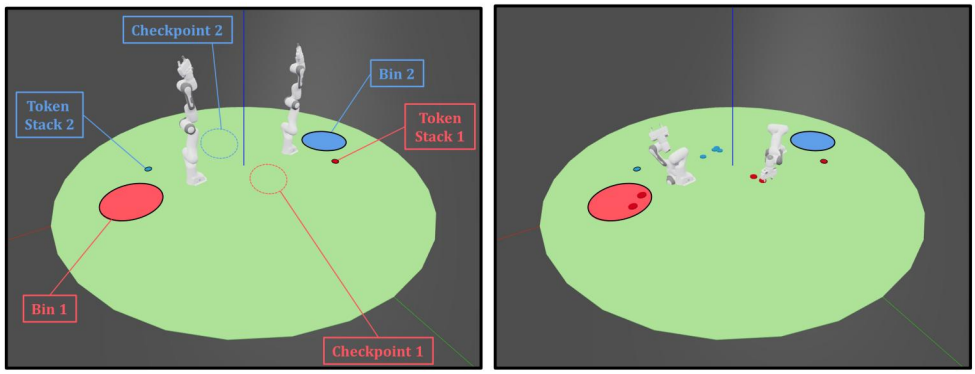

Figure 1: Tandem robot environment. Two Franka Emika Panda robotic arms (https://www.franka.de/) must work together to move two sets of labeled tokens from their initial stack to their respective storage bins. Left: Initial episode configuration. Right: Example of a mid-episode configuration.

4 Methods

4.1 Deep Q-Network

To begin the learning task with an established baseline, we chose to apply a pre-implemented DQN baseline from Stable Baselines3 [3], a set of reliable implementations of reinforcement learning algorithms in PyTorch, to our problem. To paraphrase the DQN strategy in [2, 3, 4], the Q-learning algorithm is implemented but using a deep learned approximation of the value function $Q(s, a)$. The agent learns an optimal policy by choosing sequences of actions which maximize the final reward over the duration of the training period.

Three notable features of the Stable Baselines [3] implementation include:

1. *Replay buffer*: Training batches are sampled from a revolving experience pool, rather than the last experiences collected
2. *Target network*: A copy of the Q-network is retained and used to evaluate the Q-network while the base network is being updated
3. *Gradient clipping*: Gradients above a certain magnitude are capped during the update

The Q-learning equation is as follows:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q * (s_{t+1}, a_{t+1}) \tag{1}$$

Where $r(s_t, a_t)$ is the reward function associated with making action a in state s at time t. $Q*$ represents the optimal reward obtainable given some state st. It can be more formally defined as:

$$Q * (s_t, a_t) = \underset{a \in \mathcal{A}}{max} \{Q(s_t, a_t) | s \in \mathcal{S}\} \tag{2}$$

where $\mathcal{A}$ and $\mathcal{S}$ represent the Action and State space respectively.

Data for training the function in the form of (state, action, reward, next state) experiences are collected real-time and stored in an experience database, which are periodically sampled from the database and "replayed" for learning the Q-function in minibatches [2, 3]. The DQN baseline uses epsilon-greedy exploration [2, 3].

4.2 Training and hyperparameter selection

The network architecture includes an input layer with the same size as the observation space, 2×64 neuron hidden layers, and a size 1×10 output layer (size of the action space). The 2×64 hidden layers network architecture is the default network architecture used in Stable Baselines for DQN [12].

The main hyperparameters we tuned included training duration, the exploration duration during training, starting exploration epsilon, ending exploration epsilon, learning rate, discount factor, and reward function. We found that a learning rate of 0.001 and a discount factor of 0.999 worked best for both tasks.

Training for too long resulted in significantly decreased performance. "Too long" depended on the task, with the more complex task of tandem training requiring more time. For the A to B placement task, we found that 500,000 timesteps was ideal, while for the tandem training task, we found that 1,000,000 timesteps was ideal.

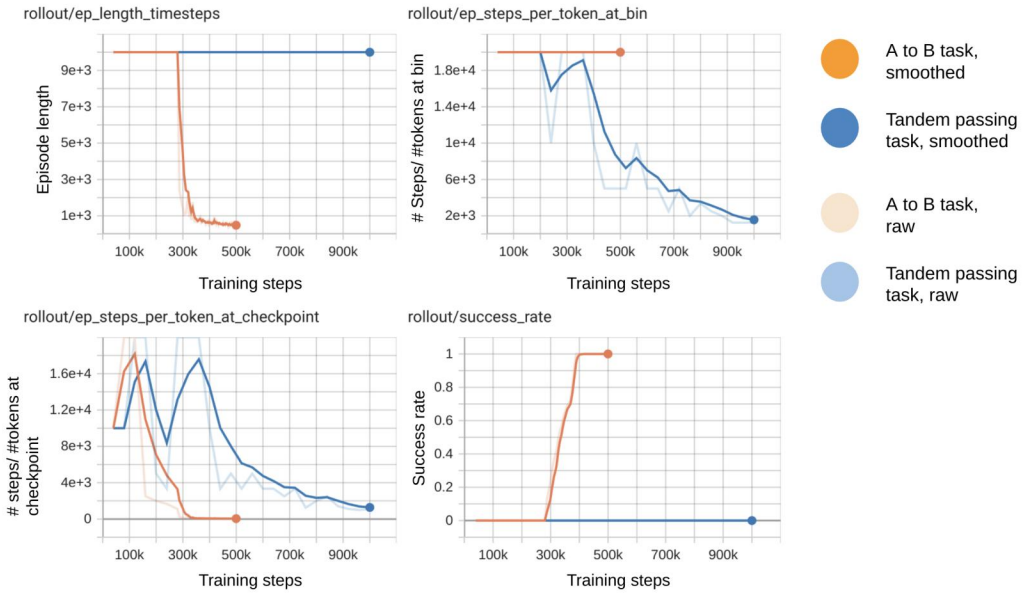

Figure 2: Plots of the models' performance during training. For the A to B passing task, training takes a shorter number of timesteps due to the lower task complexity. A success occurs when all tokens are move to the desired location (the checkpoint for A to B placement, or the bin for tandem passing). The model trained for the tandem passing task achieves placement of some tokens into the bin, but also becomes worse at placing tokens at the checkpoint location compared to the model trained only for A to B passing. To avoid division by 0, the number of training steps per token at a given location are capped at 2e4.

Training with a high initial exploration epsilon (0.95) and an 0.5 final exploration epsilon produced best results, as opposed to lower amounts of exploration. Keeping some randomness in action selection was ideal which we discuss more in the results section.

A well defined reward function was critical to training success. Because completing the entire task or even moving a single item to the correct location is too sparse of a reward, we experimented with different rewards as a function of distance from the arm to multiple tokens, start locations, and end locations, during different steps of the task. The best performing reward function is described in Algorithm 1.

5 Results

Trained Model	Random action probability	# Avg. tokens to checkpoint	# Avg. tokens to bin
Single step	$p = 0.00$	8.0	N/A
Single step	$p = 0.50$	10.0	N/A
Multi-step	$p = 0.00$	0.0	0.0
Multi-step	$p = 0.25$	4.0	2.7
Multi-step	$p = 0.38$	5.0	3.7
Multi-step	$p = 0.50$	6.0	4.7
Multi-step	$p = 0.63$	7.1	5.8
Multi-step	$p = 0.75$	7.8	6.4
Multi-step	$p = 0.87$	7.5	5.9
Multi-step	$p = 0.94$	4.4	2.7
Multi-step	$p = 1.00$	0.6	0.0

Table 1: Average moved and scored tokens in a single episode, with agents following stochastic policies with different random action probabilities. 0 tokens are scored when each agent deterministically follows its learned Q-policy ($p = 0$), and 0 tokens are scored when each agent takes purely random actions ($p = 1$). Maximum performance is achieved with $p = 0.75$. Metrics are averaged over 150-700 episodes.

5.1 Moving Objects to a Target Location

As a stepping stone to the final tandem passing problem, we first tried training the model to only move tokens from the start location to a checkpoint location in the center of the simulation environment where both arms can reach the tokens. An episode was deemed successful if each arm moved all 5 tokens to the checkpoint location before the max episode length was reached. We were able to achieve a 100% success rate over the course of 31 episodes by using the trained model's best predicted action 50% of the time and a random action 50% of the time, compared to a random baseline that achieved 0% success rate. Interestingly, we found that including 50% randomness performed better than using the model's best predicted action 100% of the time, which did not succeed at the task because it only moved 4 out of 5 tokens to the end location.

When watching the arm perform this task, the trained model with randomness achieves the task quite efficiently, with minimal eccentric movement. There is some jittering that may be explained by the inclusion of randomness in action selection.

5.2 Tandem Passing

After accomplishing a good success rate for moving items to the checkpoint, we looked into training the model to accomplish the final tandem passing task. We found that, although it is possible to train the model to move some tokens to the bin, we are not able to achieve a reliable success (defined as moving all tokens to the bin, between the two robots). We also found that when the model improves at placing more tokens into the bin, it is less effective at moving tokens to the checkpoint location. Similar to the A to B passing case, we found that using the trained model's best predicted action 50% of the time and a random action 50% of the time resulted in better performance than a random or totally deterministic trained model.

When watching the arms perform this task, the trained model with randomness is not very efficient and frequently appears to get "stuck."

6 Conclusion and Future Work

In this project, we used deep Q-learning to try to accomplish a two-agent tandem passing problem where two robotic arms must work together to move tokens to desired locations. We were able to train a model to reliably solve a subtask with 100% success rate, moving tokens to a central desired location, when including 50% randomness in action selection. We also were able to train a model that partially accomplishes the tandem passing task, when including 50-75% randomness in action selection.

We find it very interesting that including randomness in action selection improves the performance of the model compared to using the model exclusively. If we had more time and resources, we would explore why this is the case. One of our hypotheses is that we have overtrained the (relatively small) model, and a larger / deeper network may be better able to handle the complexity of the task. It would be interesting to try different, larger model architectures.

We manually tuned hyperparameters during this project. If we had more time, we would explore using automated hyperparameter search tools such as Optuna [15] to better cover the hyperparameter seach space.

Finally, the problem setup space for the project is currently deterministic, and it would be interesting to see if we can develop a solution that generalizes to randomized token start locations and destination (bin) locations.

7 Contributions

Overall, we shared the load of this project fairly equally. We came up with the concept together and defined the project plan as a team. In the first part of our project where we tried to accomplish point-to-point movement in a 2D space, Edward implemented gym environment for a 2-degree of freedom arm, Kerrie wrote scripts to train a DQN model using Stable Baselines3 [3], Zhichao built a DPPO network for training, and we each tuned the models. We found the DQN model worked best, so we used that to continue working on a 4-DOF robot arm in 3D space. In the second part of our project where we try to move objects to a target location, we worked together to brainstorm ideas on how to approach the problem, Edward rendered environment in Robotics Toolbox and explored the idea of including randomness in action selection, Kerrie created the multi-agent environment for the task and logged/plotted training metrics, and we all tried different methods to tune the model. All team members worked together for the final paper and the video presentation.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. In:arXiv preprint arXiv:1606.01540, 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236.

[3] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann. Stable Baselines3: Reliable Reinforcement Learning Implementations. In:Journal of Machine Learning Research 22(268), 1-8 (2015). http://jmlr.org/papers/v22/20-1364.html.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Weistra, M. Reidmiller. Playing Atari with Deep Reinforcement Learning. NIPS Deep Learning Workshop 2013. In:arXiv preprint arXiv:1312.5602.

[5] Z. Hong, T. Shann, S. Su, Y. Chang, T. Fu, and C. Lee. Diversity-driven exploration strategy for Deep Reinforcement Learning. Advances in Neural Information Processing Systems 31 (NeurIPS 2018). https://proceedings.neurips.cc/paper/2018/file/a2802cade04644083dcde1c8c483ed9a-Paper.pdf

[6] H. von Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1) (2016). Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/10295

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. NIPS Deep Learning Workshop (2013). https://doi.org/10.48550/arXiv.1312.5602/

[8] J. Kober, J.A. Bagnell, J. Peters. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research. 2013;32(11):1238-1274. doi:10.1177/0278364913495721

[9] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-End Training of Deep Visuomotor Policies. Journal of Machine Learning Research 17 (2016) 1-40. http://arxiv.org/abs/1504.0070

[10] H. Cheng, F. Duan H. Zheng. Deep Reinforcement Learning for an Anthropomorphic Robotic Arm Under Sparse Reward Tasks. International Conference on Intelligent Robotics and Applications. ICIRA 2021: Intelligent Robotics and Applications: 79–89. https://doi.org/10.1007/978-3-030-89098-8_8.

[11] L. Liu, Q. Liu, Y. Song, B. Pang, X. Yuan and Q. Xu. A Collaborative Control Method of Dual-Arm Robots Based on Deep Reinforcement Learning. Appl. Sci. 2021, 11, 1816. https://doi.org/ 10.3390/app11041816.

[12] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research 22 (2021) 1-8. https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html.

[13] J. K Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sulivan, L. Santos, R. Perez, C. Horsch, C. Dieffendahl, N. Williams, Y. Lokesh, R. Sullivan, and P. Ravi. PettingZoo: Gym for Multi-Agent Reinforcement Learning. In:arXiv preprint arXiv:2009.14471, 2020.

[14] J. K Terry, B. Black, and A. Hari. SuperSuit: Simple Microwrappers for Reinforcement Learning Environments. In:arXiv preprint arXiv:2008.08932, 2020.

[15] T. Akiba, S. Sano, T. Yanase, T. Ohta,and M. Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In KDD, 2019.

[16] P. Corke and J. Haviland. Not your grandmother's toolbox–the Robotics Toolbox reinvented for Python. In:IEEE International Conference on Robotics and Automation (ICRA) (2021) 11357-11363.