# Multi-environment Reinforcement Learning Agent for Robotics Tasks

Panagiotis Argyrakis
*Computer Science*
*Worcester Polytechnic Institute*
Worcester, MA
pargyrakis@wpi.edu

Revant Mahajan
*Computer Science and Robotics Engineering*
*Worcester Polytechnic Institute*
Worcester, MA
rmahajan@wpi.edu

***Abstract*—Reinforcement learning (RL) has shown great promise in enabling robots to perform complex tasks, yet most approaches train models for individual environments, limiting their generalization capabilities. In this work, we address this limitation by developing and evaluating generalized RL models for multi-environment robotic tasks. Using OpenAI's Fetch and Pick Place task as a testing environment, we developed a single actor that is able to perform both these tasks at a comparable success rate with OpenAI's specialized models. The dual-critic approach achieved high accuracy in multi-task learning by leveraging independent critics, while the environment-aware model provided a stable yet slightly less effective alternative using a single critic. The generalized model performed much better and showed lower variance in results when the data being trained was alternated more frequently. Our findings suggest that generalized RL agents can be trained to effectively learn and perform across related environments.**

## I. Introduction

Training robotics-related tasks for reinforcement learning has been an avenue for research in the past decade. A lot of sophisticated algorithms have been developed to train robots ranging from wheeled robots to legged robots. A relatively stable platform for research has been robotic manipulators. We suspect a reason for the popularity of reinforcement learning research for robotic manipulators is due to their inherent presence in industry and academia as one of the most common robots.

The field of Reinforcement learning is concerned with training an agent or a model through trial, error, and exploration. This idea of reinforcement learning profoundly relates to the way we humans learn to interact with our environment. By interacting with the environment, we try to extract useful information. Initially, our actions might not lead to the best results but that is okay. The idea is not to be perfect on the first try. If we are, that is great but the main goal is to learn as much as possible from our interactions. Based on these interactions, we get a pretty good understanding of how to perform various tasks. We can also use this understanding to perform various other tasks of a similar nature or learn them with relative ease. In trying to do so, we don't necessarily forget the learned policy from the previous tasks. We are able to build a generalized model for all the tasks we learn over time.

So far in the class, we have worked with training models for singular specific tasks. With this final project, we wanted to explore the field of training generalized reinforcement learning models. This has various benefits for the robotics domain. Rather than training a robot individually to perform different tasks, a generalized model can be created. This would significantly improve the capabilities of the robots and should bring up more use cases where robots can be used.

## II. Background

We started our literature review by reading the call for research paper published by OpenAI for these robotic tasks[9]. This paper gave us several important insights that helped narrow down which RL algorithms we could use and guided our remaining background research. First, we learned that each model in the robotics fetch environments is multi-goal, meaning that the arm is instructed to perform an action targeted towards some arbitrary point in 3-dimensional space. So, each environment represents an infinite (but limited) space of sub-tasks. Moreover, the environments have sparse rewards of 1 or 0, indicating success or failure. The agent only gets this reward once it reaches the goal or the episode ends, with no feedback in the meantime.

Second, this paper discussed how different algorithms performed, and presented benchmark results, as shown in Figure 1. Deep Deterministic Policy Gradient (DDPG) was the base algorithm for all comparisons, which is a policy gradient-based algorithm optimized for continuous action space tasks[6]. The advantage of policy-based methods is that they are model free. So, we can apply them without full knowledge of the problem or the characteristics of the robots. DDPG simultaneously learns both the Q-function and the policy. It essentially utilizes the Bellman equation and the off-policy data (buffer) to learn the Q-value of the environment. Using this Q-value, it also learns the optimal policy for the environment. The intuitive idea is very much related to Q-learning as we discussed in the class.

DDPG was tested combined with/without the Hindsight Experience Replay algorithm, and with or without continuous rewards. The results showed two important insights. First, certain tasks reached 100% success rate. All models achieved a 100% success rate in FetchReach, and the combination

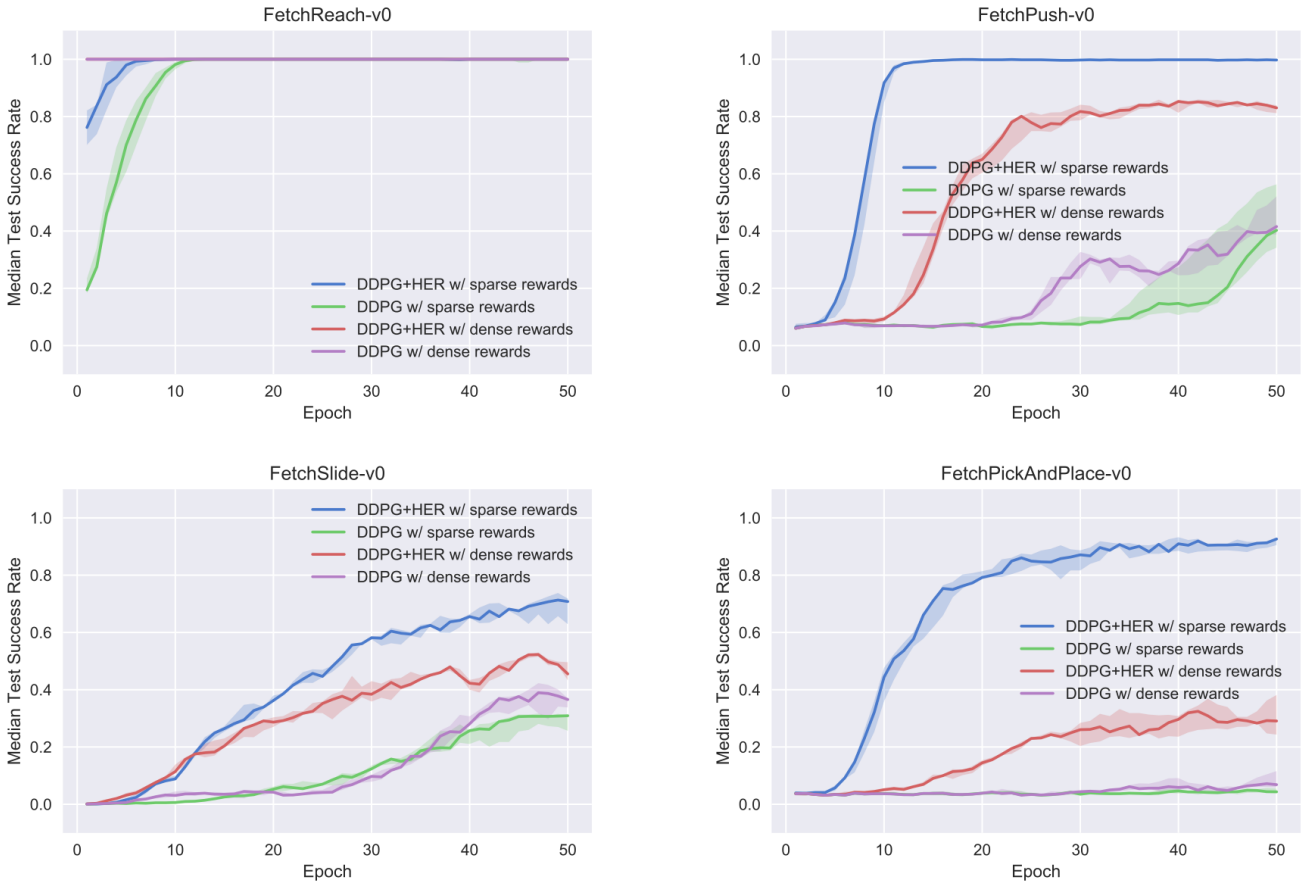


Fig. 1. From Plappert, et. al.: Median test success rate (line) with interquartile range (shaded area) for all four Fetch environments

of DDPG + HER with sparse rewards reached the same in FetchPush, around 90% in FetchPickAndPlace, around 80% in FetchPush and around 70% in FetchSlide. From these results, it is obvious that DDPG + HER with sparse rewards is the best performing model.

We focused the rest of our background research on understanding DDPG and HER so we can tweak them to achieve our project objectives. Deep Deterministic Policy Gradient (DDPG) is a type of policy gradient method. Furthermore, DDPG can be combined with Hindsight Experience Replay (HER) to significantly increase the performance and reduce the training time. Hindsight Experience Replay (HER) takes inspiration from the way we humans learn. The majority of the RL algorithms extract information based on a sequence of actions and whether those actions help us achieve the desired goal or not. If these specific actions don't help us reach our goal, they are used as a blind counterexample so very little learning occurs. HER utilizes the failed episodes by learning how to get to the end state, assuming it actually was the goal. For example, if a set of actions did not lead to the desired position (G) but to some other position (P), HER would try to extract information from this attempt as if the robot all along wanted to go to position P. This way, the model can learn from both successful and unsuccessful attempts.[1]

## III. Environment

OpenAI released eight simulated robotics environments a couple of years ago. All of these environments are multi-goal environments simulated in the MuJoCo physics engine. Four of these environments simulated robotic arms and are called Fetch environments. These simulated arms are 7 DOF and have a two-fingered parallel gripper[9]. For our experiments, we considered the following environments

1) Fetch Pushing: A box is placed on a table in front of the robot. The robot's goal is to push the box to the desired goal position indicated with a red sphere. In this situation, the robot's fingers are locked to prevent grasping to force the robot to reach the goal position by pushing the box.
2) Fetch Pick and Place: A box is placed on a table in front of the robot. The robot's goal is to pick the box from the initial position and to move it to the desired goal position, indicated with a red sphere. In this situation, the robot's fingers are not locked. This allows the robot to grasp the object to perform the task.

The goal for these fetch tasks is three dimensional and describes the end-effector position at goal. These environments

are sparse and binary reward environments. The agent gets a reward of 0 if it is able to reach the correct goal position and -1 otherwise. The observations consist of the absolute position of the gripper with respect to the environment, the position of an object with respect to the gripper if present in the environment and the goal position. The action space is 4-dimensional, where the first three dimensions indicate the cartesian position of the end effector at the next time step and the last dimension specifies the distance between the gripper fingers.[9]

**Algorithm 1** Dual-environment DDPG algorithm

Randomly initialize two critic network $Q_1(s, a|\theta^{Q_1})$ and $Q_2(s, a|\theta^{Q_2})$ and an actor $\mu(s|\theta^{\mu})$ with weights $\theta^{Q_1}$, $\theta^{Q_2}$ and $\theta^{\mu}$.
Initialize target network $Q'_1$, $Q'_2$ and $\mu'$ with weights $\theta^{Q'_1} \leftarrow \theta^{Q_1}$, $\theta^{Q'_2} \leftarrow \theta^{Q_2}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replay buffer $R_1$ for environment 1 and $R_1$ for environment 2
**for** episode = 1, M **do**
  Select the environment to get the observation from, the corresponding replay buffer and the corresponding critic using the environment selection strategy
  $Q(s, a|\theta^{Q}) \leftarrow Q_1(s, a|\theta^{Q_1})$ and $R \leftarrow R_1$
  or
  $Q(s, a|\theta^{Q}) \leftarrow Q_2(s, a|\theta^{Q_2})$ and $R \leftarrow R_2$
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  **for** t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random mini batch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^{Q}))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

  **end for**
**end for**

## IV. Methodology

In this project, our primary goal was to modify the standard DDPG + HER algorithm to be able to train with an arbitrary number of different multi-goal environments(n). We restricted n to two - FetchPickandPlace and FetchPush to ensure our scope was achievable. Rather than creating our own implementation from scratch, we decided to modify an existing implementation. We used the implementation from Tianhong Dai[3] because it is full-featured and highly readable. We decided against using the official OpenAI baselines because the high complexity of the code would make our changes more difficult to read and understand.

### A. Environment Selection Strategy

We trained four different versions of each model design to represent different ways to interlace the two environments during training. The first, which we called "PickPlaceThenPush", split the training process into two parts by training exclusively on Fetch Pick and Place (environment 1) for the first half of training and then exclusively on Fetch Push (environment 2) for the second half. The second approach was the same as the first but in reverse order. We aptly named it "PushThenPickPlace". The third approach switched which environment was being used for training at every epoch. So, the first epoch would use Fetch Pick and Place (environment 1), the second epoch would use Fetch Push (environment 2), and so on. We named this approach "EpochInterlaced". Our final approach switched which environment was being used for training at every cycle and was named "CycleInterlaced". All of our models performed 50 cycles at every epoch. All result visualizations were smoothed to make trends easier to discern.

### B. Baseline

Our baselines used an unmodified implementation of the DDPG + HER algorithm. To avoid sampling bias, we used two separate HER and replay buffers to store rollouts for the two environments.

### C. Design of multi-environment agent

DDPG uses the critic networks to identify how good a certain policy is for a particular environment and the actor-networks give the desirable action to execute based on the current observation. Since we are training a generalized model, we had to stick with using just one-actor in the approaches we decided to pursue. We maintained two different HER and replay buffers for both the environments.

*1) Dual-critic:* Since the critic network essentially estimates how good a policy is for a given observation, we decided on the approach of using two different critic networks, each corresponding to the different environments. The intuition is to train the critic networks for their respective environments and use the feedback from these networks to train the actors such that the actor can generalize the policy for both of the environments. The pseudocode for this approach is present algorithm 1.

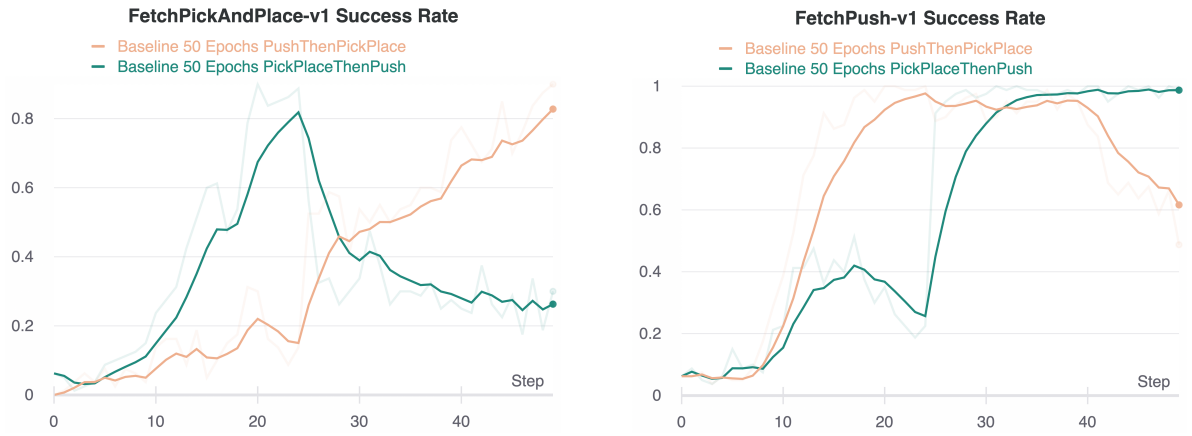


Fig. 2. Success Rate of the two Environments with PickPlaceThenPush and PushThenPickPlace Training Mode

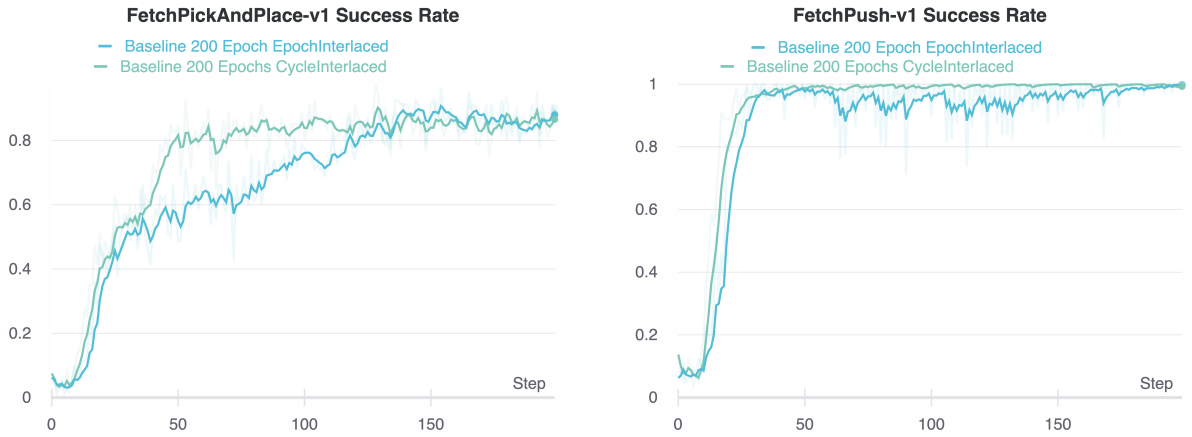


Fig. 3. Success Rate of the two Environments with EpochInterlaced and CycleInterlaced Training Mode over Epochs

*2) Environment-aware:* In this approach we used one critic for both environments. We modified the observation to make the model aware of which environment was being processed by injecting an extra parameter into the observation. This parameter, referred to as env_id in the code, was 0.0 for environment 1 and 1.0 for environment 2. With the added parameter, we expected the agent to be capable of differentiating the policy for each environment.

### D. Experiments

We performed the following experiments:

1) For our baseline models (unmodified DDPG + HER):
   a) PickPlaceThenPush with 50 epochs
   b) PushThenPickPlace with 50 epochs
   c) Epoch Interlaced with 200 epochs
   d) Cycle Interlaced with 200 epochs
2) For our dual-critic approach:
   a) PickPlaceThenPush with 50 epochs
   b) PushThenPickPlace with 50 epochs
   c) Epoch Interlaced with 50 epochs
   d) Cycle Interlaced with 200 epochs
3) For our environment aware approach:
   a) Cycle Interlaced with 200 epochs

We used the same model hyperparameters as the baseline OpenAI implementation. We trained each model for 50 epochs on an 8-core, Intel-based Google Elastic Compute virtual machine instance.

## V. Results and Analysis

### A. Baselines

For both PickPlaceThenPush and PushThenPickPlace, we expected the model to perform well for the tasks (environment 2) it trained on the latter part of the training. Our hypothesis was shown to be correct, as evident from the results discussed below.

*1) PickPlaceThenPush for 50 Epochs:* In Figure 2 at epoch 25, we see a sudden dip in the performance on the FetchPickAndPlace task which was being trained so far. As the training process approaches 50 epochs, the FetchPickAndPlace performance keeps going down whereas the performance for FetchPush starts increasing.

*2) PushThenPickPlace for 50 Epochs:* The PushThenPickPlace behaved similarly to the PickPlaceThenPush training mode but in reverse order, as expected. In Figure 2 at epoch 25,

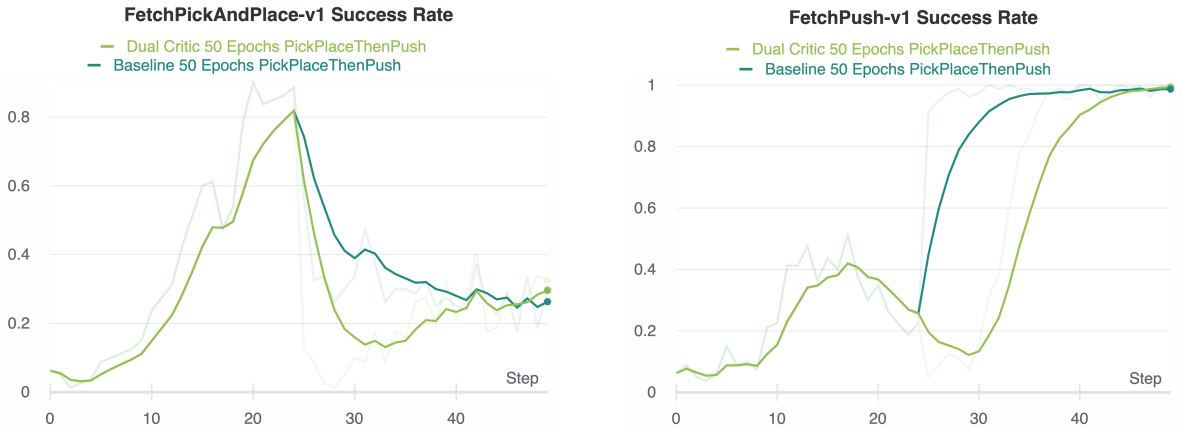


Fig. 4. Success Rate of Dual-Critic PickPlaceThenPush Versus the Baseline over Epochs

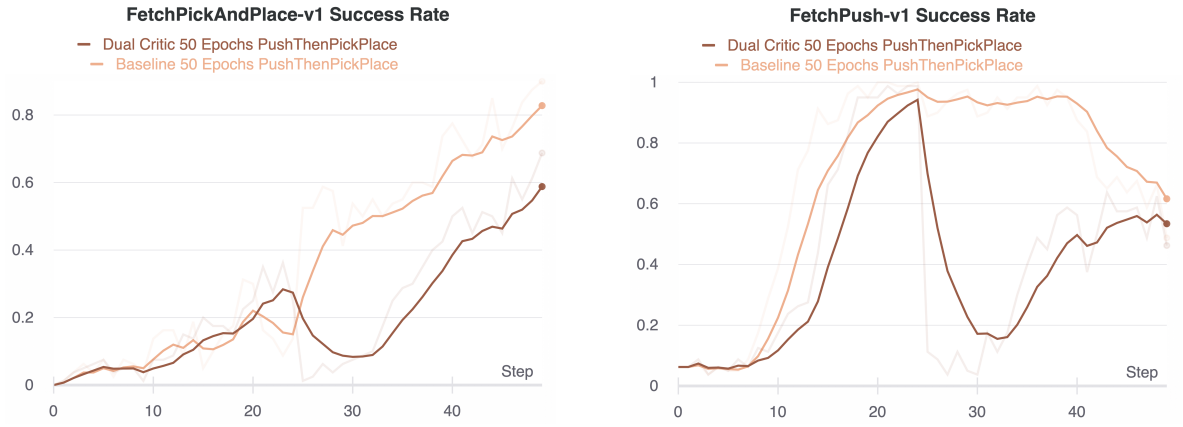


Fig. 5. Success Rate of Dual-Critic PushThenPickPlace Versus the Baseline over Epochs

we see a sudden dip in the performance on the FetchPush task which was being trained so far. As training nears epoch 50, this performance keeps decreasing whereas the performance for FetchPickAndPlace increases.

*3) EpochInterlaced for 200 Epochs:* By switching the environment being trained at each epoch, the model had higher variances in accuracy between epochs and took longer than PickPlaceThenPush and PushThenPickPlace to get similar results. We observe the success rate over 200 epochs in Figure 3. However, in contrast to those approaches, the performance increases over time for both environments. Ad the end of 50 epochs, the success rate of FetchPickAndPlace and FetchPush was 0.5625 and 0.9874 respectively. At the end of 200 epochs, the score was 0.8875 and 1.0 respectively.

*4) CycleInterlaced for 200 Epochs:* Each epoch consists of 50 cycles. By switching the environment at every cycle, the model had higher variances in accuracy as compared to PickPlaceThenPush and PushThenPickPlace but lower compared to the EpochInterlaced training mode. It also took longer than PickPlaceThenPush and PushThenPickPlace to get similar results but, as we can see in Figure 3, was significantly quicker compared to the EpochInterlaced training mode.

### *B. Dual-critic agent*

*1) FetchPickAndPlace then FetchPush for 50 Epochs:* We observed that, similarly to the baselines, the PickPlaceThenPush approach resulted in the agent learning only one of the two environments at either half of the training and unlearning the other. As we can see in Figure 4, the agent achieved a success rate of 0.8875 at FetchPickAndPlace-v1 at epoch 24 which was reduced to 0.325 at the end of the training. Similarly, the agent had a score of 0.225 at epoch 24 for FetchPush-v1 which increased to 1.0 by the end of the training. The dual-critic model performed worse than the baseline. We believe this happens because, in the dual-critic model, the new critic has to be trained from scratch starting at epoch 25. Since the two environments are similar the single-critic approach can reuse some of its existing training and converge quicker. However, both models achieve the same results.

*2) FetchPush then FetchPickAndPlace for 50 Epochs:* The results of this experiment follow the same pattern from the previous. FetchPush, which was trained first, achieved a success rate of 0.9875 at epoch 24 while at the same epoch FetchPickAndPlace had a success rate of 0.25 as we can see in Figure 5. At epoch 50, FetchPush dropped to a success rate of 0.4625 and FetchPickAndPlace rose to 0.6875. We observe

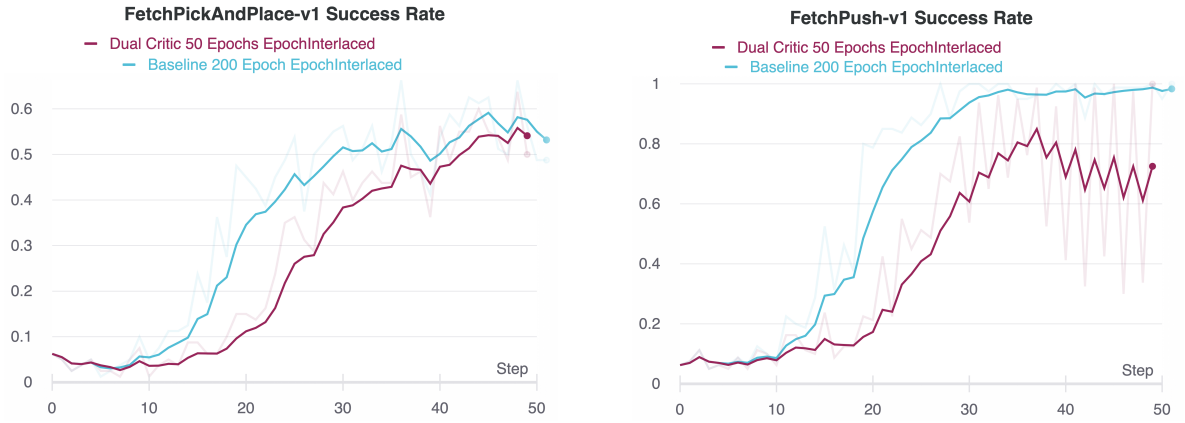


Fig. 6. Success Rate of Dual-Critic EpochInterlaced Versus the Baseline over Epochs


Fig. 7. Success Rate of Dual-Critic CycleInterlaced Versus the Baseline over Epochs

that the baseline reached a significantly higher FetchPickAndPlace success rate of 0.9 which is what the original OpenAI baseline model achieved. Moreover, we notice the baseline retained a much higher FetchPush score for the vast majority of the training process while also improving at FetchPickAndPlace faster. So, the environments share various characteristics. Intuitively, it is easy to understand that the FetchPickAndPlace tasks that have a target location on the platform surface can be solved by simply performing FetchPush.

*3) Epoch Interlaced Training with 50 Epochs:* This experiment trained for both environments almost simultaneously by altering which environment was being sampled at every epoch. In Figure 6 we observe that the success rate for both environments trends upwards. For the dual-critic model, it reaches 0.5 for FetchPickAndPlace and 0.7252 for FetchPush at the end of epoch 50. However, the model shows a very high variance for FetchPush since the unsmoothed data fluctuates from 0.3 to 1.0 during the last 10 epochs. This result is explained by the fact that the independent critics lead to high values for the gradient update. Last, we observe that the baseline performed much better with dramatically less variance in FetchPush, with a final success rate of 1.0.

*4) Cycle Interlaced with 200 epochs:* This experiment switches the environment at every cycle during training. Each epoch consists of 50 cycles, so this method of training interlaces the environments much more tightly than the epoch-based method. We also extended the training time to 200 epochs to make sure the models fully converge.

We observe in Figure 7 that using this method, both the baseline and the dual-critic agent reach the benchmark success rate achieved by the OpenAI models [9] for both environments at the same time. This is a very surprising result that shows that the two environments are so closely related that an agent can learn to perform both without being explicitly given which environment is fed as an input. It is worth noting that the CycleInterlaced method produces significantly less variance than all previous methods. The baseline outperforms the dual-critic method for the first 70 epochs but then is slightly surpassed by it. At epoch 200, the baseline reaches FetchPickAndPlace and FetchPush success rates of 0.9 and 1.0 respectively while the dual-critic method reaches 0.9375 and 1.0 respectively.

### C. Environment-aware agent


Fig. 8. Success Rates of CycleInterlaced Baseline, Dual-Critic and Environment-Aware Models over Epochs

*1) CycleInterlaced for 200 Epochs:* As shown in Figure 8, the environment-aware agent initially performed better than the dual-critic model but after epoch 30 it fell behind both the baseline and the dual-critic model. Past epoch 30 it consistently performed worse than the other two but generally converged to the same success rate. The final result was 0.8625 and 1.0 for FetchPickAndPlace and FetchPush respectively. For FetchPickAndPlace, performance fell short of the baseline by 3.375% and the dual-critic model by 7.5%. However, this agent showed more predictable and stable behavior during the video demonstration. Specifically, the baseline actor would oscillate the robot fingers when not grasping the cube. The environment-aware agent did not show this behavior.

Overall, we could attribute the lower performance to the exponentially increased state space due to the extra input parameter. The curse of dimensionality may have prevented this agent from maximizing its performance in PickAndPlace.

## VI. Conclusion

Through this project, we wanted to understand and evaluate the possibility of training generalized tasks for robotic environments. We had not expected the baselines to perform as well as the Dual Critic and Environment Aware approaches. We think one reason for this result is the similarity between the two tasks we considered: FetchPush and FetchPickAndPlace. Intuitively, we can easily understand how all the FetchPickAndPlace tasks with the target location on the platter can be done as FetchPush. We believe this is the main reason why the baselines scored similar success rates as the OpenAI benchmarks [9], Dual Critic approach, and Environment Aware Approach. From the results, it can be clearly seen that generalized agents can be created for FetchPush and FetchPickAndPlace tasks. It would be an interesting avenue of research to extrapolate this study and examine how generalized agents would perform on other Fetch robotics environments.

## VII. Future Work

Within the scope of this project, we aimed to develop a generalized model for only a pair of tasks. Consequently, our modified algorithm is tailored towards training these specific two tasks. The next step for this project should be to create a generalized training algorithm capable of handling 'n' number of environments. From our discussions, we realized that FetchPush is essentially a subset of FetchPickAndPlace. All the pick and place tasks with a goal location on the platter can also be achieved by pushing the objects. We are interested in exploring how well these generalized models can be trained for tasks that overlap less. An example would be training on FetchPickAndPlace and FetchSlide. We can also use a similar approach to evaluate the similarity between two tasks from the perspective of how models can generalize them. This can be done by making unique pairs from the 4 tasks and then comparing the results of the trained generalized models. It would also be interesting to train the generalized models on Hand environments, which were shown to be more complex than the Fetch environment in OpenAI's call for research [9].

## VIII. References

### References

[1] Marcin Andrychowicz et al. *Hindsight Experience Replay*. URL: https://goo.gl/SMrQnI..

[2] Panagiotis Argyrakis and Revant Mahajan. *panargirakis/hindsight-experience-replay: This is the pytorch implementation of Hindsight Experience Replay (HER) for mulitple environments*. URL: https://github.com/panargirakis/hindsight-experience-replay.

[3] Tianhong Dai. *TianhongDai/hindsight-experience-replay: This is the pytorch implementation of Hindsight Experience Replay (HER) - Experiment on all fetch robotic environments.* URL: https://github.com/TianhongDai/hindsight-experience-replay.

[4] Matteo Hessel Deepmind et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. URL: www.aaai.org.

[5] Alishba Imran. *Training a Robotic Arm to do Human-Like Tasks using RL — by Alishba Imran — Data Driven Investor — Medium*. URL: https://medium.com/datadriveninvestor/training-a-robotic-arm-to-do-human-like-tasks-using-rl-8d3106c87aaf.

[6] Timothy P Lillicrap et al. *CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING*. URL: https://goo.gl/J4PIAz.

[7] Rivlin Or. *Reinforcement Learning with Hindsight Experience Replay — by Or Rivlin — Towards Data Science*. URL: https://towardsdatascience.com/reinforcement-learning-with-hindsight-experience-replay-1fee5704f2f8.

[8] Jan Peters and Stefan Schaal. *Policy Gradient Methods for Robotics*.

[9] Matthias Plappert et al. *Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research*. URL: http://fetchrobotics.com/.

[10] Chris Yoon. *Deep Deterministic Policy Gradients Explained — by Chris Yoon — Towards Data Science*. URL: https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b.