Dynamic Experience Replay

Jieliang Luo and Hui Li
Autodesk Research, San Francisco, United States
rodger.luo@autodesk.com, hui.xylo.li@autodesk.com

Abstract: We present a novel technique called Dynamic Experience Replay (DER) that allows Reinforcement Learning (RL) algorithms to use experience replay samples not only from human demonstrations but also successful transitions generated by RL agents during training and therefore improve training efficiency. It can be combined with an arbitrary off-policy RL algorithm, such as DDPG [1] or DQN [2], and their distributed versions.

We build upon Ape-X DDPG [3] and demonstrate our approach on robotic tight-fitting joint assembly tasks, based on force/torque and Cartesian pose observations. In particular, we run experiments on two different tasks: peg-in-hole and lap-joint. In each case, we compare different replay buffer structures and how DER affects them. Our ablation studies show that Dynamic Experience Replay is a crucial ingredient that either largely shortens the training time in these challenging environments or solves the tasks that the vanilla Ape-X DDPG cannot solve. We also show that our policies learned purely in simulation can be deployed successfully on the real robot. The video presenting our experiments is available at https://sites.google.com/site/dynamicexperiencereplay

Keywords: Reinforcement Learning, Robotics, Experience Replay

1 Introduction

Industrial robots have been heavily used in manufacturing and other industries, however, as they rely on pre-defined trajectories, they require precise calibration and fail to adapt to uncertainties. Adaptability to imprecision, varying conditions, and less structured environments is key to the future of automation. Reinforcement Learning (RL) has recently led to a range of successes in solving sequential decision-making problems, including learning control policies for robotic tasks. The control policies are learned through agents interacting with their surrounding environments and hold promises for generalizing to new scenarios in reaction to real-time observations [4, 5].

We focus on robotic assembly tasks that involve contact forces, because such tasks are widespread in industrial applications and yet challenging for robots to do. When the assembly pieces are in contact with one another, pose observations (direct from motion capture or indirect from perception learning models) alone are often insufficient. We explicitly consider force/torque observations for policy learning. During training, we randomize the initial condition within a pre-defined range and show flexibility of the learned policy to varying conditions.

Most of the recent success in RL was achieved using model-free methods [6, 2, 1, 7, 8]. They tend to achieve optimal performance, are generally applicable, and are easy to implement, but it is achieved at the cost of being data intensive. Leveraging human demonstrations [9] as well as various experience replay [10, 11, 12] has shown to improve data efficiency.

We present a novel technique called Dynamic Experience Replay (DER) that allows RL algorithms to use experience replay samples not only from human demonstrations but also successful transitions generated by RL agents during training and therefore improve training efficiency. It can be combined with an arbitrary off-policy RL algorithm, such as DDPG or DQN, and their distributed versions. DER can be seen as a technique of over-sampling the under-represented class (successful trajectories in our case) from an imbalanced dataset, which has been studied and addressed in supervised learning [13, 14].

We build upon Ape-X DDPG and demonstrate our approach on robotic tight-fitting joint assembly tasks, in particular, peg-in-hole and lap-joint tasks. In each case, we compare different replay buffer structures and how DER affects them. Our ablation studies show that Dynamic Experience Replay is a crucial ingredient that largely shortens the training time in these challenging environments or solves the tasks that the vanilla Ape-X DDPG cannot solve. We also show that our policies learned purely in simulation can be deployed successfully on an industrial robotic arm performing the physical tasks.

The remainder of this paper is structured as follows. The problem statement and related work are stated in Sec. 2, followed by a detailed explanation of the proposed Dynamic Experience Replay in Sec. 3. Experiment setup, results, and deployment on a real robot are presented in Sec. 4. Sec. 5 concludes the paper and proposes future work.

2 Problem Statement and Related Work

2.1 Problem Statement

The RL problem at hand can be described as learn an optimal policy $\pi_\theta(a_t|s_t)$ for choosing an action a_t given the current observation s_t in order to minimize the expected total loss:

$$\min_{\pi_\theta} \mathbb{E}_{\tau\sim\pi_\theta}(l(\tau)) \tag{1}$$

where θ is the parameterization of policy π, trajectory $\tau = \{s_0, a_0, s_1, a_1, ..., s_T, a_T\}$, $\pi_\theta(\tau) = p(s_0)\prod_1^T p(s_t|s_{t-1}, a_{t-1})\pi_\theta(a_t|s_t)$, and l is the loss function of the trajectory τ.

Equation 1 can be solved if a dynamics model $p(x_t|x_{t-1}, a_{t-1})$ is provided, however, the dynamics model in contact-rich tasks is difficult to obtain. Alternatively, the equation can be solved by model-free RL algorithms to avoid using dynamics. DDPG is a model-free off-policy RL algorithm for continuous action spaces. In DDPG, an actor policy $\pi : S \to A$ is created to explore the space and store the collected transition (s_j, a_j, s_{j+1}, r_j) in a replay buffer R. Meanwhile, a critic policy $Q : S \times A \to \mathbb{R}$ is created to approximate the actor's action-value function Q^π.

We would like to learn an optimal policy, which takes Cartesian pose and force/torque observations as input and outputs Cartesian velocity.

2.2 RL for High Precision Assembly

RL has been studied actively in the area of high precision assembly as it can reduce human involvement and increase the robustness to uncertainties. Inout el al. [15] used a Q-learning based method with LSTM [16] for Q-function approximation to solve low-tolerance peg-in-hole tasks. Luo el at. [17] extended a model-based approach MDGPS [18] with haptic feedback for learning the insertion of a peg into a deformable hole. Fan el at. [19] combined DDPG [1] and GPS [6] to take advantage of both model-free and model-based RL [4] to solve high-precision Lego insertion tasks. Luo el at. [20] combines iLQG [21] with force/torque information by incorporating an operational space controller to solve a group of high-precision assembly tasks. In our work, we use torque/force and pose in Cartesian space as observations and 6 DOF Cartesian velocities as actions. This method bypasses the robot dynamics, which are usually inaccurate in simulation.

2.3 Leveraging Experience Replay in RL

Experience replay [10] has been used to improve training efficiency in many RL algorithms, particularly for model-free RL, as it's less sample efficient than model-based RL. The technique has become popular after it was incorporated in the DQN [2] agent playing Atari games. Prioritized experience replay [11] is a further improvement to prioritize transitions so agents can learn from the most "relevant" experiences. Hindsight experience replay [12] stored every transition in the replay buffer not only with the original goal but also with a subset of other goals to acquire a more generalized policy. DDPG from Demonstrations [9] modified DDPG to permanently store a set of human demonstrations in the replay buffer to solve a group of insertion tasks. We extend DDPG

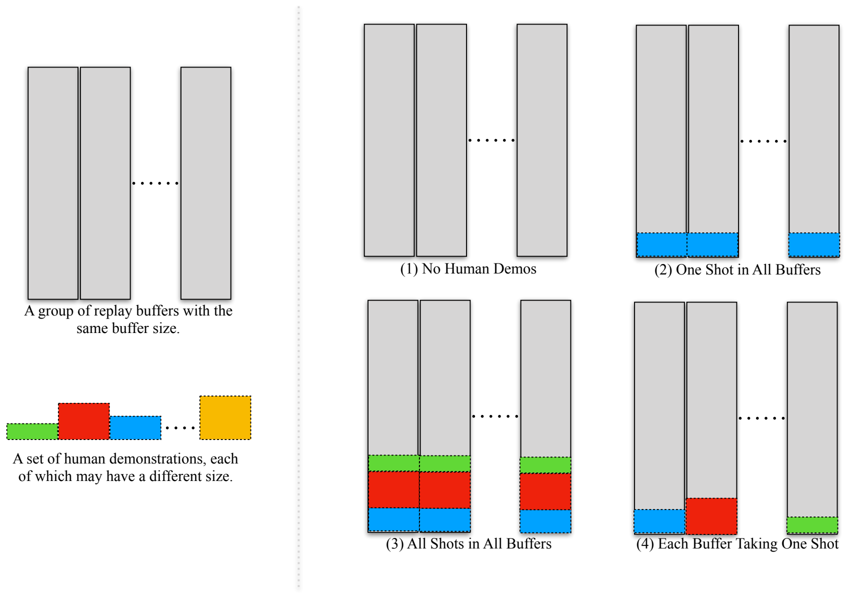
A group of replay buffers with the same buffer size.
A set of human demonstrations, each of which may have a different size.
(1) No Human Demos
(2) One Shot in All Buffers
(3) All Shots in All Buffers
(4) Each Buffer Taking One Shot

Figure 1: The four types of replay buffer structure for our experiments: (1) No human demonstrations in any buffer; (2) Same one-shot human demonstration in all buffers; (3) All human demonstrations in all buffers; (4) Each buffer with a different one-shot demonstration.

from Demonstrations to a distributed framework and propose a set of experience replay structures in the context of distributed RL. Details are discussed in Sec. 3.2.

2.4 Distributed RL

Distributed RL can greatly increase training efficiency of model-free RL. Ape-X [3] disconnects exploration from learning by having multiple actors interact with their own environments and select actions from a shared neural network. D4PG [22], with the Ape-X framework, uses a distributional critic update to achieve a more stable learning signal. There are also a growing number of examples applying the distributed architecture to popular RL algorithms, such as Distributed PPO [23] and Distributed BA3C [24]. Since our action space is continuous, we build our algorithm based on RLlib's [25] implementation of Ape-X DDPG.

3 Method

As model-free RL algorithms require excessive data, one or multiple shots of human demonstrations are sometimes introduced in the replay buffer R for complex manipulation tasks [9, 26]. However, human demonstrations are not always helpful if the observation space during human demonstration does not match that during training. For example, for the high-precision lap-joint assembly task, we do not have haptic feedback during demonstration in simulation and only visual inspection is used, while during training, force/torque observations are required. Therefore, we propose a novel technique that augments human demonstrations with successful transitions generated by RL agents to improve training efficiency.

3.1 Setup

Observations: The observation space is 13-dimensional. The policy is given as input the position (x, y, z) and orientation (q_x, q_y, q_z, q_w) of the timber piece attached to the robot end-effector, and the torque/force reading $(f_x, f_y, f_z, t_x, t_y, t_z)$ from the sensor, which is mounted on the end of the robot arm. We do not use visual input to simplify the problem.

Actions: The action space is 6-dimensional. The policy outputs the desired linear velocity (v_x, v_y, v_z) and angular velocity (w_x, w_y, w_z) of the timber piece attached to the robot end-effector.

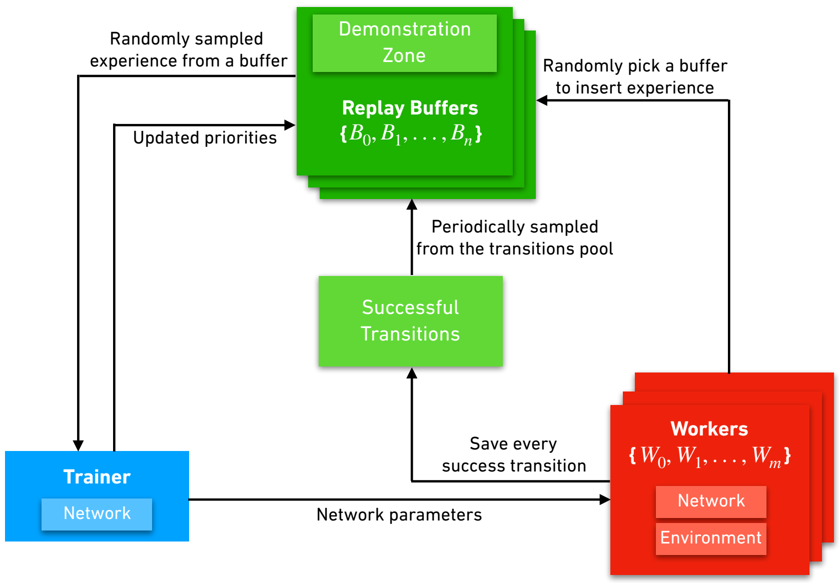
Randomly sampled experience from a buffer
Demonstration Zone
Replay Buffers
$\{B_0, B_1, \ldots, B_n\}$
Randomly pick a buffer to insert experience
Updated priorities
Periodically sampled from the transitions pool
Successful Transitions
Save every success transition
Workers
$\{W_0, W_1, \ldots, W_m\}$
Network
Environment
Trainer
Network
Network parameters

Figure 2: The Dynamic Experience Replay framework: multiple workers, each with its own instance of environment, and multiple replay buffers, each with capacity $\mathbb{C}$ for demonstrations. Human demonstration(s) are stored in the demonstration zones before training starts. During training, all successful transitions that are generated by workers are saved in a pool, which is sampled periodically by each replay buffer and stored in the demonstration zone.

Human demonstrations: For each task, depending on the replay buffer structure (Sec. 3.2), zero, one or six human demonstrations are recorded in simulation, using a game controller to drive the robot end-effector until the joint is successfully assembled. Each demonstration includes all transitions from one successful episode. Each transition is of the form $e_t = (s_t, a_t, s_{t+1}, r_t)$.

Rewards: We use a simple linear reward function based on the distance between the goal pose and the current pose of the timber piece attached to the robot arm for both tasks. Additionally we use a large positive reward (+1000 for the peg-in-hole and +100 for the lap-joint) if the object is within a small distance of the goal pose:

$$r = \begin{cases} -|g-x|, & |g-x| > \epsilon \\ -|g-x| + R, & |g-x| \leq \epsilon \end{cases}$$

where x is the current pose of the object, g is the goal pose, ϵ is a distance threshold, and R is the large positive reward. We use negative distance as our reward function to discourage the behavior of loitering around the goal because the negative distance also contains time penalty.

3.2 Replay Structures in Distributed RL

Off-policy RL algorithms perform experience replay by sampling minibatches from a pool of stored samples, which allows the use of arbitrary data like human demonstrations. Based on prioritized experience replay and Ape-X DDPG, we suggest four different replay buffer structures that can take advantage of demonstrations in distributed RL, as shown in Fig. 1.

Each buffer structure consists of a fixed number of replay buffers that load zero, one, or multiple human demonstrations before training starts. Without Dynamic Experience Replay, each buffer permanently keeps all the demonstrations with top priorities during training. The following section discusses how the buffer structures work with Dynamic Experience Replay.

3.3 Dynamic Experience Replay

The idea behind Dynamic Experience Replay (DER) is to augment human demonstrations using successful trajectories generated by RL agents during training, especially in cases where human demonstrations are not very helpful. We define *demonstrations* as either human demonstrations or the successful trajectories generated by RL agents. If DER is activated, regardless of the buffer structures mentioned above, each buffer allocates capacity $\mathbb{C}$ specifically for demonstrations. We refer to this as the *demonstration zone*. During training, all the successful episodes generated by RL agents are stored in a pool. Periodically, each replay buffer randomly samples one successful episode from the pool and stores it in the demonstration zone. When the demonstration zone is full, the oldest transitions are discarded. DER's framework in a distributed architecture is shown in Fig. 2.

As in Ape-X, the DER algorithm consists of two concurrent parts, which are workers and a trainer. For each worker, after collecting the transitions of one episode, it randomly chooses a replay buffer and sends over the transitions. The trainer, in parallel, randomly selects a replay buffer and samples a batch of transitions for network update. The trainer also updates the priority of transitions in the selected buffer at the end of the training cycle. See Alg. 1 for a formal description of the algorithm.

Algorithm 1: Dynamic Experience Replay

```
Given:
    • a distributed off-policy RL algorithm $\mathbb{A}$,          ▷ e.g. APE-X DDPG, APE-X DQN
    • an experience replay structure $\mathbb{S}$ [1].
    • one-shot or a group of human demonstrations $\mathbb{D}$ (optional)
Initialize $\mathbb{A}$                                  ▷ Initialize neural networks
Initialize replay buffers $\mathbb{B}$                     ▷ Initialize a group of replay buffers
Load $\mathbb{D}$ to $\mathbb{B}$ based on $\mathbb{S}$   ▷ Load human demos to the replay buffers based on the replay structure
Initialize $\mathbb{T}$                     ▷ Initialize a pool to save success transitions from agents

For each worker:
for episode = 1, M do
    $\theta_0 \leftarrow$ Trainer.parameters()      ▷ Update the latest network parameters for the trainer
    $s_0 \leftarrow$ Environment.reset()                  ▷ Get initial state from its own environment
    for t = 1, T do
        $a_{t-1} \leftarrow \pi_{\theta_{t-1}}(s_{t-1})$                ▷ Choose an action from the current policy
        $(r_{t-1}, s_t) \leftarrow$ Environment.step($a_{t-1}$)        ▷ Apply the action to the environment
        Transitions.Add($[s_t, a_{t-1}, r_{t-1}, s_{t-1}]$)          ▷ Add data to a temp buffer
    $\mathbb{B}_n$.Add(Transitions)          ▷ Send the transitions to a randomly selected replay buffer
    if $episode_t$ succeeds then
                                                          ▷ Save success transitions
        $\mathbb{T}$.Add(Transitions)
    Periodically($\theta_t \leftarrow$ Trainer.Parameters())       ▷ Update to the latest network parameters

For the trainer:
for episode = 1, M do
    $\theta_0 \leftarrow$ InitializeNetwork() for t = 1, T do
        $\tau \leftarrow \mathbb{B}_n$.Sample()      ▷ Sample a batch of transitions from a randomly selected buffer
        $l_t \leftarrow$ ComputeLoss($\tau; \theta_t$)      ▷ Calculate loss using an off-policy algorithm, like DDPG
        $\theta_{t+1} \leftarrow$ UpdateParameters($l_t; \theta_t$)
        $p \leftarrow$ ComputePriorities()          ▷ Calculate priorities of the transitions in buffers[2]
        $\mathbb{B}_n$.SetPriority($p$)                     ▷ Update priorities to the selected buffer
    Periodically($\mathbb{B}_i$.Update($\mathbb{T}_j$))  ▷ Replace previous demos with a success transition from the pool
```

1. The four different experience replay structures are discussed in Sec 3.2
2. We use absolute TD error for the calculation.

A hyper-parameter to experiment with DER is which replay buffer structure to use. In the next section, we compare the four types of replay buffer structure discussed in Sec. 3.2 and how DER affects them.

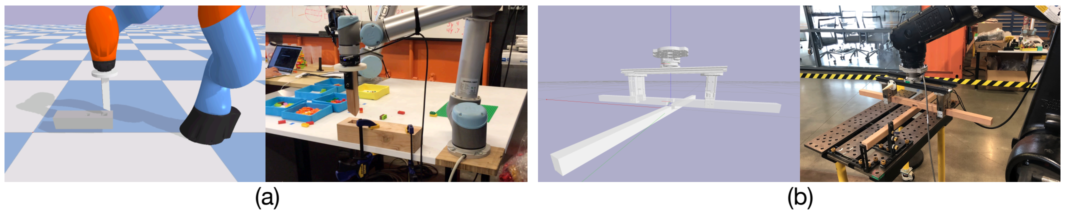

Figure 3: Two joint assembly tasks for algorithm evaluation: (a) chamfered peg-in-hole, (b) lap-joint. For both joints, the CAD model used in simulation is used to fabricate the real-world pieces.

4 Experiments

This section is organized as follows. In Sec. 4.1 we introduce distributed RL environments we use for the experiments as well as our training setup and procedure. In Sec. 4.2 we compare the performance of different replay buffer structures with and without DER. In Sec. 4.3 we describe the deployment on the physical robot.

The video presenting our experiments is available at `https://sites.google.com/site/dynamicexperiencereplay`.

4.1 Environments

We modeled our assembly tasks in the PyBullet [27] simulation engine. Specifically, we customized two tasks, chamfered peg-in-hole and lap-joint, which correspond to the real-world setup, as shown in Fig. 3.

For the peg-in-hole task, we used a KUKA LRB iiwa robotic arm in simulation and attached a torque/force sensor between the end of the arm and a peg, as shown in Fig. 3(a). In order to be robot agnostic, we limit both the observations and actions in the Cartesian space. This way the trained model can be deployed on any arbitrary robotic arm. To demonstrate the point, we created a robot-less mode for training in simulation for the lap-joint task, as shown in Fig. 3(b). The robot-less setup helps us bypass needing a robot model in simulation, as most of them are inaccurate.

For each task, we initialized 6 replay buffers and collected 6 human demonstrations. Each demonstration consists of a sequence of transitions of different lengths. Depending on which replay buffer structure is activated, the human demonstration data are used differently, as described in Fig. 1. Training is performed using the Ape-X DDPG algorithm and we adapted it from RLlib's implementation.

Initial states: For the peg-in-hole tasks, the initial angle along the z-axis of the peg is randomized from 0 to 360 degree and other parts are fixed. For the lap-joint tasks, we randomized the initial angle along the z-axis and x-y position of the timber on the ground within a small range. The details of the initial randomnesses are documented in Fig. 4 and Fig. 5.

4.2 Results

In order to evaluate how DER affects the performance we evaluate Ape-X DDPG with and without DER on both two tasks. For each task, we conducted eight types of experiments, which are of four different replay-buffer structures with and without DER. Each experiment was performed on an Amazon AWS c5n.9xlarge instance.

Fig. 4 shows that DER significantly improves the performance of the peg-in-hole task in most of the buffer structures, including No Human Demos, One Shot in All Buffers, and All Shots in All Buffers. For the latter two buffer structures, the average successful rates of DER are greatly higher than vanilla Ape-X DDPG. For the No-Human-Demos buffer structure, although both algorithms have similar average successful rates by the end of the training, DER is nearly two times as fast at achieving the success rate as vanilla Ape-X DDPG. For the Each-Buffer-Taking-One-Shot buffer structure, with DER and without have similar performance.

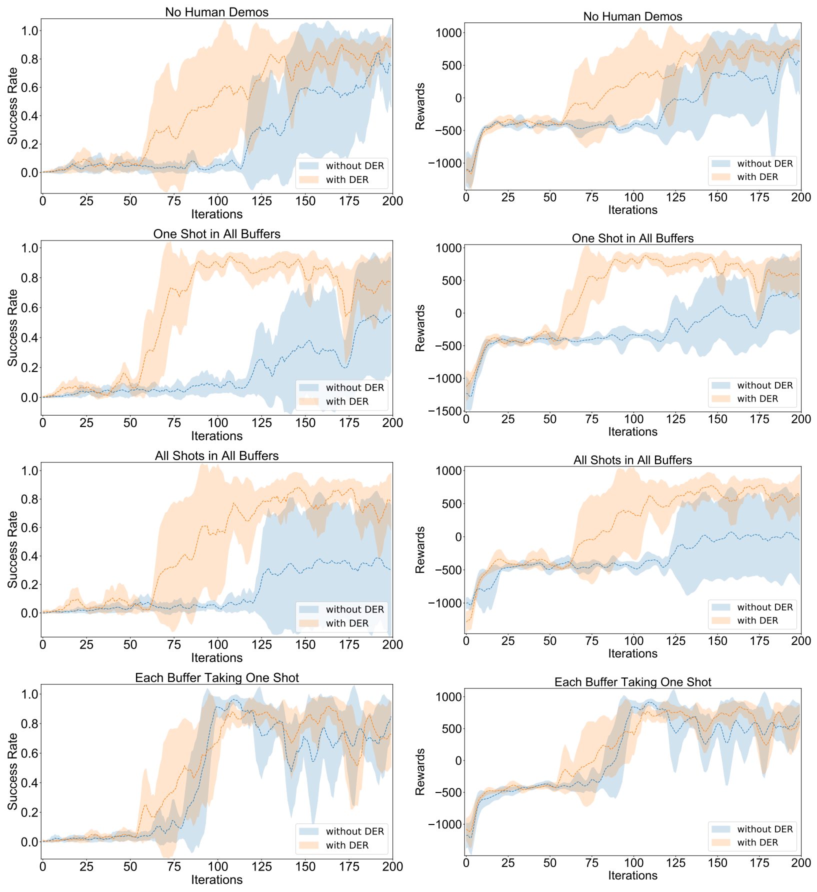

Figure 4: Success rate comparison (left) and reward comparison (right) for the peg-in-hole experiments, in which the initial angle of the peg along the z-axis is randomized within [0, 360°]. Each plot compares the performances of a replay buffer structure with and without DER. Each iteration consists of 50 to 80 episodes and is approximately 200,000 timesteps. The dotted lines show the mean of each iteration across 3 trainings with different random seeds and the shaded areas show the 95% confidence bound. Each training experiment is terminated at 200 iterations.

The lap-joint task is more challenging because the timber pieces have straight corners (no chamfer) and tight tolerance (1mm). Hence, as seen in Fig. 5, the average success rate of each iteration across different training runs is slightly lower than the peg-in-hole task. Fig. 5 shows that DER has better performances than vanilla Ape-X DDPG with two of the buffer structures, No Human Demos and All Shots in All Buffers, while the performances of with DER with the other two buffer structures are similar to without DER. It is unclear why DER does not improve the performance with the Each-Buffer-Taking-One-Shot structure in either task. Further studies need to be conducted.

4.3 Deployment on a physical robot

After training purely in simulation, we deployed the learned policy of the lap-joint task on a KUKA KR60 industrial robot arm, as shown in Fig. 3(b). Our hardware setup includes an ATI Delta 6-axis force/torque sensor, two Schunk parallel-jaw grippers, and two pre-fabricated timber pieces with a half notch on each. As discussed in Sec. 4.1, the observations are force/torque values obtained from the force/torque sensor and pose information of the top timber piece obtained from the robot controller. The actions, linear and angular velocity of the top timber piece, are sent to the robot controller from the policy. The No-Human-Demos buffer structure was used for training the policy, which was successfully deployed on the real robot 3 out of 3 times. We have included the deployment in the video.

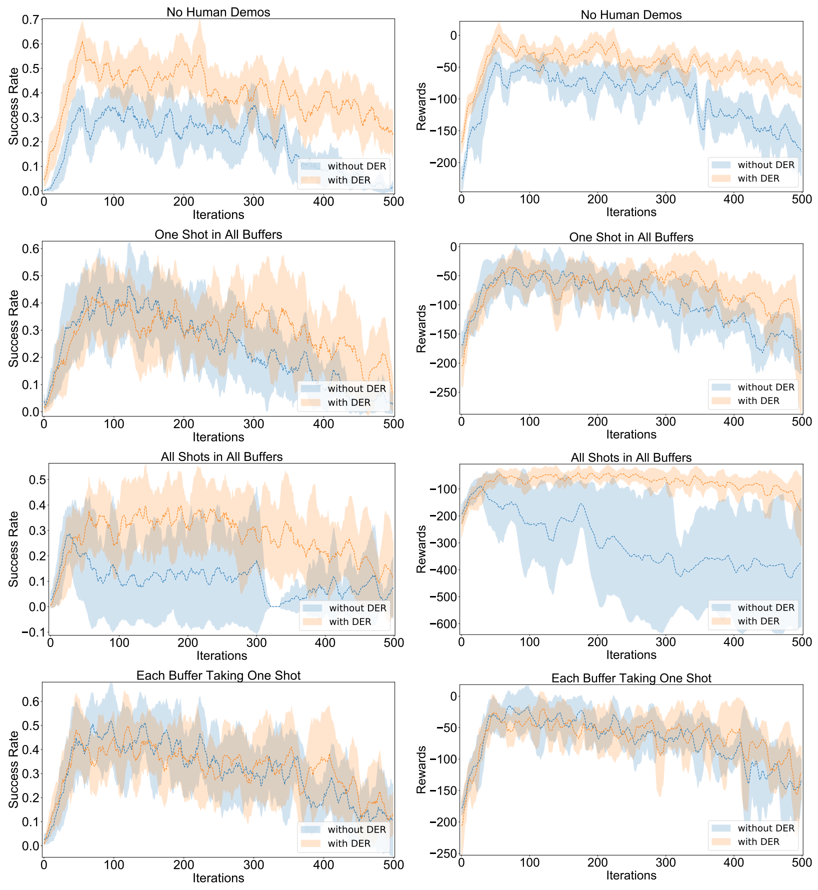

Figure 5: Success rate comparison (left) and reward comparison (right) for the lap-joint experiments, in which the initial angle of the ground timber piece along the z-axis is randomized within [$-2°$, 0] and the initial position [-2mm, 2mm] in both x and y. Each graph compares the performances of a replay buffer structure with and without DER. Each iteration consists of 50 to 80 episodes and is approximately 200,000 timesteps. The dotted lines show the mean of each iteration across 3 trainings with different random seeds and the shaded areas show the 95% confidence bound. Each training experiment is terminated at 500 iterations.

5 Discussion and Future Work

This paper proposed a novel technique called Dynamic Experience Replay (DER), which improves training efficiency of an off-policy RL algorithm. The technique uses successful episodes generated by RL agents as demonstrations in replay buffers to augment human demonstrations. DER can be seen as a technique of over-sampling the under-represented class from imbalanced data in supervised learning. Our technique can be considered as an add-on feature to an arbitrary off-policy RL algorithm and we experimentally demonstrated that with Ape-X DDPG.

We showed that DER in both the peg-in-hole and the lap-joint tasks improved training efficiency in comparison to the vanilla Ape-X DDPG algorithm. For occasions where the vanilla RL algorithm failed to solve the task within the given timeframe, DER could either achieve the training goal or largely improve the success rate. We also showed that the learned policy for the lap-joint task can be successfully deployed on the real robot.

In the future, we would like to evaluate DER on a group of model-free off-policy RL algorithms, such as PPO, and on other assembly tasks. We would also further study DER in terms of hyperparameters, such as the sampling rate and the number of replay buffers.

Acknowledgments

We thank Erin Bradner and Mike Haley for budgetary support of the project; Nicolas Cote for setting up communication and control of the KUKA robot; Matteo Pacher, Aleksandra Anna Apolinarska and Rafael Pastrana for discussions.

A Hyperparameter Details

We used Adam [28] as the optimizer for both the actor and the critic networks with a learning rate of 10^{-3}. Instead of using two learning rates, we used two different loss coefficients, 0.1 for the actor and 1.0 for the critic. The target network update frequency is 50,000 and the buffer size is 2,000,000. We allocated 1% of the buffer size for storing demonstrations, which is 20,000. We used prioritized experience reply and the prioritized replay alpha is 0.5. The sample batch size is 50, the train batch size is 512, and the batch mode is truncate episodes. The minimum per iteration time is 20 seconds. We set soft target updates τ to 1 as we used target network update frequency for network update. We used mean standard filter as the observation filter.

For all the peg-in-hole tasks, both the actor and the critic networks have 2 hidden layers with 64 and 64 units. We assigned 5 workers and each of them occupied one logical CPU core. For all the lap-joint tasks, both the actor and the critic networks have 2 hidden layers with 256 and 256 units. We assigned 30 workers and each of them occupied one logical CPU core as well.

For the rest of the hyperparameters, we inherited directly from RLlib's default setup of Ape-X DDPG.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *6th International Conference on Learning Representations*, 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature 518, pages 529533*, 2015.

[3] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[4] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[5] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research 32.11 (2013), pp. 12381274*, 2013.

[6] S. Levine and V. Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

[7] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[9] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[10] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[12] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[13] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[14] H. He, Y. Bai, E. A. Garcia, and S. Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328. IEEE, 2008.

[15] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana. Deep reinforcement learning for high precision assembly tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 819–825. IEEE, 2017.

[16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, and A. M. Agogino. Deep reinforcement learning for robotic assembly of mixed deformable and rigid objects. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2062–2069. IEEE, 2018.

[18] W. H. Montgomery and S. Levine. Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*, pages 4008–4016, 2016.

[19] Y. Fan, J. Luo, and M. Tomizuka. A learning framework for high precision industrial assembly. *arXiv preprint arXiv:1809.08548v3*, 2018.

[20] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, A. Tamar, and P. Abbeel. Reinforcement learning on variable impedance controller for high-precision robotic assembly. *arXiv preprint arXiv:1903.01066*, 2019.

[21] E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.

[22] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, A. Muldal, N. Heess, and T. Lillicrap. Distributed distributional deterministic policy gradients. In *6th International Conference on Learning Representations*, 2018.

[23] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[24] I. Adamski, R. Adamski, T. Grel, A. Jedrych, K. Kaczmarek, and H. Michalewski. Distributed deep reinforcement learning: Learn how to play atari games in 21 minutes. In *International Conference on High Performance Computing*, pages 370–388. Springer, 2018.

[25] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica. Rllib: Abstractions for distributed reinforcement learning. *arXiv preprint arXiv:1712.09381*, 2017.

[26] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[27] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*, 2016.

[28] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.