


RESEARCH ARTICLE

Learning vision-based robotic manipulation tasks sequentially in offline reinforcement learning settings

Sudhir Pratap Yadav¹ , Rajendra Nagar² and Suril V. Shah³

¹iHub Drishti Foundation, Jodhpur, India, iHub Drishti Foundation

²Department of Electrical Engineering, IIT Jodhpur, Jodhpur, India, IIT Jodhpur

³Department of Mechanical Engineering, IIT Jodhpur, Jodhpur, India, IIT Jodhpur

Corresponding author: Sudhir Pratap Yadav; Email: yadav.1@iitj.ac.in

Received: 27 May 2023; **Revised:** 19 January 2024; **Accepted:** 20 February 2024; **First published online:** 2 May 2024

Keywords: sequential task learning; offline deep reinforcement learning; grasping; computer vision; vision-based manipulation

Abstract

With the rise of deep reinforcement learning (RL) methods, many complex robotic manipulation tasks are being solved. However, harnessing the full power of deep learning requires large datasets. Online RL does not suit itself readily into this paradigm due to costly and time-consuming agent-environment interaction. Therefore, many offline RL algorithms have recently been proposed to learn robotic tasks. But mainly, all such methods focus on a single-task or multitask learning, which requires retraining whenever we need to learn a new task. Continuously learning tasks without forgetting previous knowledge combined with the power of offline deep RL would allow us to scale the number of tasks by adding them one after another. This paper investigates the effectiveness of regularisation-based methods like synaptic intelligence for sequentially learning image-based robotic manipulation tasks in an offline-RL setup. We evaluate the performance of this combined framework against common challenges of sequential learning: catastrophic forgetting and forward knowledge transfer. We performed experiments with different task combinations to analyse the effect of task ordering. We also investigated the effect of the number of object configurations and the density of robot trajectories. We found that learning tasks sequentially helps in the retention of knowledge from previous tasks, thereby reducing the time required to learn a new task. Regularisation-based approaches for continuous learning, like the synaptic intelligence method, help mitigate catastrophic forgetting but have shown only limited transfer of knowledge from previous tasks.

1. Introduction

Robotics has experienced a significant transformation with the integration of deep Reinforcement Learning (RL), which has revolutionised robot capabilities in manipulation tasks. Unlike traditional control architecture that depends on fixed rules and explicit programming, RL enables robots to learn adaptively from observations, modifying their behaviour in response to contextual cues. This advancement allows robots to adjust and optimise actions for new tasks, enhancing their utility in diverse scenarios. This allows robots to handle rigid objects in various industrial operations and manage deformable items. While significant progress has been made in enabling robots to handle a variety of rigid and deformable objects, challenges persist in the scalability and efficiency of these learning models.

A significant challenge emerges when a single agent, such as a domestic robot, must learn new tasks as different situations arise. Existing multitask frameworks lack the flexibility to incorporate new tasks without retraining the agent on all existing tasks. This paper uses a sequential learning approach where the robot acquires tasks one after the other. This method enables the robot to adapt to new circumstances without the necessity for comprehensive retraining. We use offline RL as the base framework to learn a single image-based robotic manipulation task and then use a regularisation-based continual learning approach for learning tasks sequentially. This combined framework forms the main contribution of this work.

We mainly focus on two challenges of continual learning: catastrophic forgetting and forward knowledge transfer. Catastrophic forgetting is the tendency of artificial neural networks to forget previous information when new information is provided. In a continual learning scenario, the neural network's accuracy on previous tasks drops significantly as it tries to learn a new task. Forward knowledge transfer tries to capture the improvement in the performance of the current task based on the learning from previous tasks. For example, once a robot learns to pick up an object, it can reuse this knowledge in other tasks that require picking up objects without the need to learn it again. An effective continual learning algorithm should be able to increase its performance on the current task (based on previous tasks) while maintaining its performance on previous tasks.

The developed framework, combining offline reinforcement learning with synaptic intelligence for continual learning, offers a promising approach to overcoming challenges associated with adapting robots to new tasks. Additionally, we analyse the effects of task ordering and the number of object configurations on both forgetting and the knowledge transfer between tasks.

1.1. Related work

In the past several years, the field of robotics has witnessed substantial progress, particularly marked by robots gaining a variety of manipulation skills via deep Reinforcement Learning (RL). The introduction of the Deep Q-Network (DQN) [1] marked a significant advancement in robotic manipulation, paving the way for the development of advanced end-to-end policy training methods [2–5].

1.1.1 Single-task RL

These developments enabled robots to effectively perform a range of rigid object manipulations, including pick-and-place operations [6,7], stacking [8], sorting [9], insertion tasks [10–12], as well as more complex challenges such as opening doors [13], opening cabinets [14], using electric drills [15], and completing assembly tasks [16–18]. Additionally, the application of RL expanded to include the manipulation of deformable objects, like ropes [19] and folding clothes [20], with learning systems typically acquiring these skills from task-specific datasets.

1.1.2 Multitask RL

However, for such an approach to be more effective across a broader range of tasks, it necessitates the collection and use of specific data for each task, along with training distinct networks. A promising solution to this challenge is multitask reinforcement learning (RL). In this approach, an agent undergoes simultaneous training across multiple tasks. Throughout the training process, the algorithm has access to data (sampled trajectories) from all tasks and optimises them jointly. This strategy enables the agent to develop generalised skills applicable across various tasks. Multitask RL has been applied successfully to learn robotic manipulation tasks [21–25]. However, in multitask RL, the set of tasks and the distribution of task-related data remain constant. Consequently, the agent requires retraining from the beginning for any new task, even if there is significant overlap with previously learned tasks. Such a requirement for retraining renders the scaling of this approach to human-equivalent mastery of all manipulation tasks impractical. In contrast, humans leverage their experience from prior tasks to facilitate new task learning, avoiding the need to start from scratch. The sequential (or continual) learning model attempts to address this limitation by providing a framework where an agent learns tasks sequentially. As a result, when encountering a new task, the agent does not necessitate complete retraining.

1.1.3 Online continual RL

Continual learning research began with a focus on classification tasks using datasets like MNIST and CIFAR [26–28]. Recently, the field has expanded to include continual reinforcement learning (RL), with applications in Atari games [29] and GYM environments [30], and extended to robotic manipulation

tasks [31], [32]. [31] introduced a benchmark for continual learning in robotic manipulation, providing baselines for key continual learning methods in online RL settings, particularly using the soft actor-critic (SAC) method [33]. However, this research primarily focuses on online-continual RL with low-dimensional observation spaces, such as joint and task space data, under the assumption of complete access to the simulator. In contrast, our study emphasises offline-continual RL with high-dimensional observation space, specifically images, in the sequential learning of robotic manipulation tasks.

1.1.4 Catastrophic forgetting

In sequential deep reinforcement learning, neural networks trained on one task often experience performance degradation when retrained on another, a phenomenon known as catastrophic forgetting. This issue, central to connectionist models, arises from a **stability-plasticity** dilemma: if we strive to make the network flexible enough to accommodate new information, it tends to lose its stability, consequently resulting in a degradation of its performance on previously learned tasks. Conversely, if we lean towards enhancing its stability, the network may struggle to effectively acquire the new task, as presented in ref. [34]. One strategy to address catastrophic forgetting involves the adoption of penalty-based methods. These methods apply constraints on neural network parameters, ensuring that the weights of the neural network stay closer to the solutions derived from previous tasks. Notably, Elastic Weight Consolidation (EWC) [29] and Synaptic Intelligence (SI) [35] have made significant contributions in this area. Kirkpatrick et al. (2017) [29] introduced EWC, offering a regularisation-based solution to catastrophic forgetting; however, its calculation of parameter importance is not localised. This paper adopts the Synaptic Intelligence approach, as proposed by Zenke et al. (2017) [35], due to its localised assessment of synaptic importance (weights in Neural Networks). The local nature of this computation aids in maintaining solution generality, unaffected by specific problem characteristics. Furthermore, SI boasts advantages in computational speed and simplicity of implementation compared to EWC, which necessitates the computation of the Fisher Information Matrix.

Singh et al. (2020) [36] applied Offline-RL to image-based manipulation tasks, focusing on initial condition generalisation without exploring sequential task learning. In contrast, to the best of our knowledge, this study is the first to explore sequential learning in image-based robotic manipulation within offline-RL settings.

2. Learning image-based robotic manipulation tasks sequentially

In this section, we formulate our RL agent and environment interaction setup to learn robotic manipulation tasks. We then discuss the problem of sequential task learning and present an approach to solve this problem.

2.1. RL formulation for learning image-based robotic manipulation tasks

Agent and environment interaction is formally defined by the Markov Decision Process (MDP). A Markov Decision Process is a discrete-time stochastic control process. In RL, we formally define the MDP as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. Here, \mathcal{S} is a finite set of states, \mathcal{A} is a finite set of actions, \mathcal{P} is the state transition probability matrix, \mathcal{R} set of rewards for all state-action pair and γ is the discount factor. A stochastic policy is defined as a probability distribution over actions, given the states, that is, the probability of taking each action for every state. $\pi(a_i|s_i) = \mathbb{P}[a_i \in \mathcal{A}|s_i \in \mathcal{S}]$. We formulate the vision-based robotic manipulation tasks in the MDP framework as below.

- **Environment:** It consists of WidowX 250, a five-axe robot arm equipped with a gripper. We place a table in front of the robot and a camera in the environment in an eye-to-hand configuration. Every task consists of an object placed on the table, which needs to be manipulated to complete the task successfully.

- **State:** The state s_t represents the RGB image of the environment captured at time step t . We use images of size $48 \times 48 \times 3$.
- **Action:** We define the action at the time step t as a 7-dimensional vector $a_t = [\Delta x_t, \Delta o_t, g_t]^\top$. Here, $\Delta x_t \in \mathbb{R}^3$, $\Delta o_t \in \mathbb{R}^3$, $g_t \in \{0, 1\}$ denotes the change in position, change in orientation, and gripper command (open/close), respectively, at time step t .
- **Reward:** The reward $r(s_t, a_t) \in \{0, 1\}$ is a binary variable which is equal to 1 if the task is successful and 0, otherwise.

The reward is kept simple and not shaped according to the tasks so that the same reward framework can be used while scaling for a large number of tasks. Also, giving a reward at each time step, instead of at the end of the episode, makes the sum of rewards during an episode dependent on time steps. Therefore, if the agent completes a task in fewer steps, the total reward for that episode will be more.

2.2. Sequential learning problem and solution

We define the sequential tasks learning problem as follows. The agent is required to learn N number of tasks but with the condition that tasks will be given sequentially to the agent and not simultaneously. Therefore, when the agent is learning to perform a particular task, it can only access the data of the current task. This learning process reassembles how a human learns. Let a sequence of robotic manipulation tasks T_1, T_2, \dots, T_N be given. We assume that each task has the same type of state and action space. Each task has its own data in typical offline reinforcement learning format $\langle s_t, a_t, r_t, s_{t+1} \rangle$. The agent has to learn a single policy π , a mapping from state to action, for all tasks. If we naively train a neural network in a sequential manner, the problem of catastrophic forgetting will occur, which means performance on the previous task will decrease drastically as soon as the neural network starts learning a new task.

We use a regularisation-based approach presented in ref. [35] to mitigate the problem of catastrophic forgetting. Fig. 1 describes the framework we developed to solve this problem.

3. Integrating sequential task learning with offline RL

In this section, we first discuss the SAC-CQL [37] offline RL algorithm used for learning a single robotic manipulation task and its implementation details. We then discuss the Synaptic Intelligence (SI) regularisation method for continual learning and provide details to integrate these methods to learn sequential robotic manipulation tasks.

3.1. SAC-CQL algorithm for offline RL

There are two frameworks, namely online and offline learning, to train an RL agent. In the case of an online RL training framework, an RL agent interacts with the environment to collect experience, update itself (train), interact again, and so on. Simply put, the environment is always available for the RL agent to evaluate and improve itself further. This interaction loop is repeated for many episodes during training until the RL agent gets good enough to perform the task successfully. This dynamic approach allows the agent to adapt to unforeseen circumstances but may be computationally expensive and less sample-efficient. While in offline RL settings, we collect data once and then it is not required to interact with the environment. These data can be collected by executing a hand-designed policy or by a human controlling the robot (human demonstration). Data are a sequence of $\langle s_t, a_t, r_t, s_{t+1} \rangle$ tuples. Offline RL poses unique challenges such as data distribution shifts and the need to balance exploration and exploitation without the ability to collect additional data during the learning process. While it may lack the adaptability of online RL, offline RL is computationally efficient and allows for systematically exploring individual tasks with carefully curated datasets.

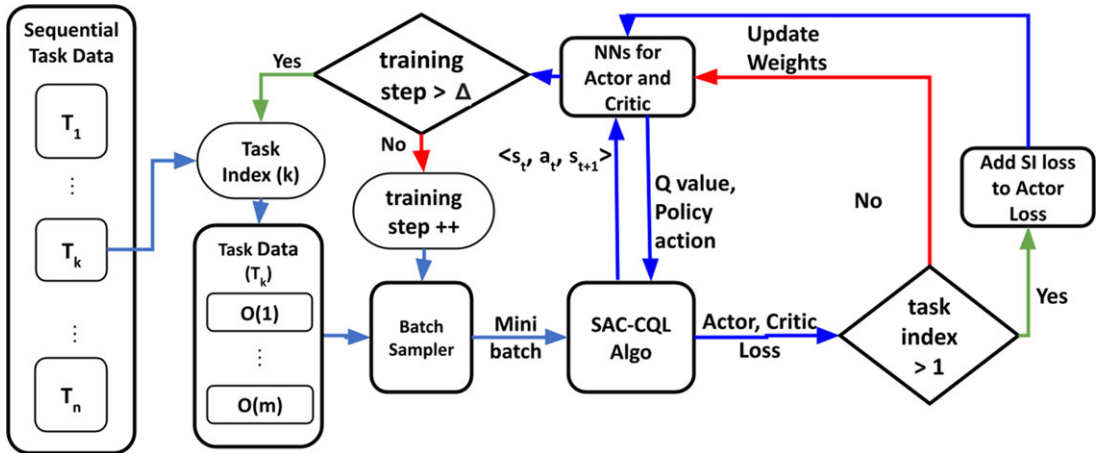


Figure 1. The SAC-CQL-SI method (soft actor-critic - conservative Q-learning - synaptic intelligence) for sequential learning is depicted in the block diagram. This method begins with a dataset comprising tasks 1 to N, from which one task is selected sequentially, as denoted by the task index k . Initially, the algorithm samples a mini-batch from the dataset of the current task. This batch is then processed by the soft actor-critic conservative Q-learning (SAC-CQL) algorithm, which calculates the losses for both the actor (policy network) and the critic (Q-network). In instances where the task index exceeds one, quadratic regularisation [35] is integrated into the actor loss to mitigate forgetting. These calculated losses are subsequently used to update the neural networks that embody the policy (actor-network) and the Q-value function (critic network). The process involves continuous sampling of subsequent batches, with training on the current task persisting until a predetermined number of training steps is reached. After completing training for a task, the agent transitions to the next task by loading its data and incrementing the task index. This cycle is repeated, allowing the agent to progress through and learn each task successively.

In recent years, SAC [33] has emerged as one of the robust ways for training RL agents in continuous action space (when action is a real vector), which typically is the case in robotics. SAC is an off-policy entropy-based actor-critic method for continuous action MDPs. Entropy-based methods add entropy term to the existing optimisation goal of maximising expected reward. In addition to maximising expected reward, the RL agent also needs to maximise the entropy of the overall policy. This helps make the policy inherently exploratory and not stuck inside a local minima. Haarnoja et al. [33] define the RL objective in maximum entropy RL settings as in (1).

$$J(\pi) = \sum_{t=0}^T \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))]. \tag{1}$$

Here, $\rho_\pi(s, a)$ denotes the joint distribution of the state and actions over all trajectories that the agent could take and $\mathcal{H}(\pi(\cdot | s_t))$ is the entropy of the policy for state s_t , as defined in (2). α is the temperature parameter controlling the entropy in the policy. \mathbb{E} represents the expectation over all state and action pairs sampled from the trajectory distribution ρ_π . Overall, this objective function tries to maximise the expected sum of rewards along with the entropy of the policy.

$$\mathcal{H}(\pi(\cdot | s_t)) = \mathbb{E}[-\log(f_\pi(\cdot | s_t))]. \tag{2}$$

Here, $\pi(\cdot | s_t)$ is a probability distribution over actions and $f_\pi(\cdot | s_t)$ is the probability density function of the policy π , we have selected Gaussian distribution to represent the policy.

SAC provides an actor-critic framework where the actor separately represents the policy, and the critic only helps in improving the actor, thus limiting the role of critic only to training. As our state (s) is

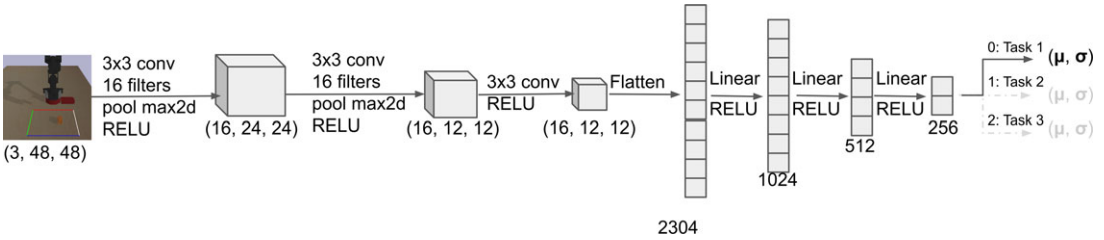


Figure 2. CNN architecture of policy network. It has three convolutional layers with max-pooling following the first two. The convolutional layers are followed by a four-layer MLP (multilayer perceptron). The output layer is multiheaded, with the number of heads corresponding to the number of tasks. Based on the current task index, only one head is enabled during training and testing. Given an input state (an RGB image), the network produces two 7-dimensional vectors, μ and σ , representing the mean and standard deviation of the stochastic policy modelled by a Gaussian distribution.

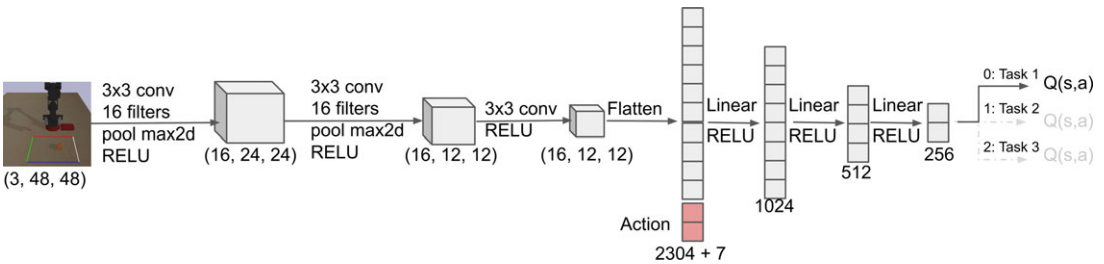


Figure 3. The Q -network's CNN architecture is similar to that of the policy network, with two notable distinctions: firstly, the action vector is incorporated into the first layer of the MLP, and secondly, the network's output is a scalar Q -value. This network takes state (an RGB image) and action (a 7-dimensional vector) and produces scalar Q -value, that is, $Q(\mathbf{s}, \mathbf{a})$.

an image, we use convolutional neural networks (CNNs) to represent both actor and critic. Also, instead of using a single Q -value network for the critic, we use two Q -value networks and take their minimum to estimate better the Q -value, as proposed in ref. [38]. To stabilise the learning, we use two more neural networks to represent target Q -values for each critic network, as described in DQN [1]. Therefore, we use 5 CNNs to implement the SAC algorithm.

Figures 2 and 3 illustrate the architectures of the policy and the Q -value neural networks, respectively. As we parameterise the policy and Q -value using neural networks, ϕ represents the set of weights of the policy network. $\theta_1, \theta_2, \hat{\theta}_1,$ and $\hat{\theta}_2$ represent the set of weights of two Q -value networks and two target Q -value networks for the critic, respectively. Since our policy is stochastic, we use *tanh-Gaussian* policy, as used in ref. [36]. The policy network takes the state as input and outputs the mean (μ) and standard deviation (σ) of the Gaussian distribution for each action. Action is then sampled from this distribution and passed through *tanh* function to bound actions between $(-1, 1)$. Target Q -value is defined as

$$\hat{Q}_{\hat{\theta}_1, \hat{\theta}_2}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) = \mathbf{r}_t + \gamma \mathbb{E}_{(\mathbf{s}_{t+1} \sim \mathcal{D}, \mathbf{a}_{t+1} \sim \pi_{\phi}(\cdot | \mathbf{s}_{t+1}))} [\hat{Q}_{\min} - \alpha \log(\pi_{\phi}(\mathbf{a}_{t+1} | \mathbf{s}_{t+1}))], \tag{3}$$

where \hat{Q}_{\min} represents the minimum Q -value of both target Q -networks and is given by

$$\hat{Q}_{\min}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) = \min[Q_{\hat{\theta}_1}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}), Q_{\hat{\theta}_2}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})] \tag{4}$$

The target Q -value in (3) is then used to calculate Q -loss for each critic network as

$$J_Q(\theta_i) = \frac{1}{2} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} [(\hat{Q}_{\hat{\theta}_i, \hat{\theta}_2}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t))^2], \tag{5}$$

where $i \in \{1, 2\}$, \mathbf{a}_i^π is the action sampled from policy π_ϕ for state \mathbf{s}_i and \mathcal{D} represents the current task data. Further, policy-loss for actor-network is defined as

$$J_\pi(\phi) = \mathbb{E}_{(\mathbf{s}_i \sim \mathcal{D}, \mathbf{a}_i \sim \pi_\phi(\cdot|\mathbf{s}_i))} [\alpha \log(\pi_\phi(\mathbf{a}_i|\mathbf{s}_i)) - \min[Q_{\theta_1}(\mathbf{s}_i, \mathbf{a}_i^\pi), Q_{\theta_2}(\mathbf{s}_i, \mathbf{a}_i^\pi)]] \tag{6}$$

For offline-RL, we use the non-Lagrange version of the conservative Q-learning (CQL) approach proposed in ref. [37] as it only requires adding a regularisation loss to already well-established continuous RL methods like SAC. Upon adding this CQL-loss to (5), total Q-loss becomes

$$J_Q^{\text{total}}(\theta_i) = J_Q(\theta_i) + \alpha_{\text{cql}} \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} [\log \sum_{\mathbf{a}_i} \exp(Q_{\theta_i}(\mathbf{s}_i, \mathbf{a}_i)) - \mathbb{E}_{\mathbf{a}_i \sim \mathcal{D}} [Q_{\theta_i}(\mathbf{s}_i, \mathbf{a}_i)]], \tag{7}$$

where $i \in \{1, 2\}$, α_{cql} controls the amount of CQL-loss to be added to Q-loss to penalise actions that are too far away from the existing trajectories, thus keeping the policy conservative in the sense of exploration. These losses are then used to update actor and critic networks using the Adam [39] optimisation algorithm.

3.2. Applying synaptic intelligence in offline RL

Synaptic intelligence is a regularisation-based algorithm proposed in ref. [35] for sequential task learning. It regularises the loss function of a task with a quadratic loss function as defined in (8) to reduce catastrophic forgetting.

$$L_\mu = \sum_k \Omega_k^\mu (\tilde{\phi}_k - \phi_k)^2 \tag{8}$$

Here, L_μ is the SI loss for the current task being learned with index μ , ϕ_k is k -th weight of the policy network, and $\tilde{\phi}_k$ is the reference weight corresponding to policy network parameters at the end of the previous task. Ω_k^μ is per-parameter regularisation strength. For more details on calculating Ω_k^μ , refer to [35]. SI algorithm penalises neural network weights based on their contributions to the change in the overall loss function. Weights that contributed more to the previous tasks are penalised more and thus do not deviate much from their original values, while the other weights help in learning new tasks. SI defines the importance of weights as the sum of the gradients over the training trajectory, as this approximates the contribution to the reduction in the overall loss function. We use a similar approach to apply SI to Offline-RL as presented in ref. [31]. Although the authors did not use SI or offline RL, the approach is similar to applying any regularisation-based continual learning method for the actor-critic RL framework. We regularise the actor to reduce forgetting on previous tasks while learning new tasks using offline reinforcement learning. We add quadratic loss as defined in ref. [35] to the policy-loss term in the SAC-CQL algorithm. So now overall policy-loss becomes as described in (9)

$$J_\pi^{\text{total}}(\phi) = J_\pi(\phi) + cL_\mu \tag{9}$$

Here, c is regularisation strength. Another aspect of continual learning is providing the current task index to the neural network. There are many approaches to tackle this problem, from 1-hot encoding to recognising the task from context. We chose the most straightforward option of a multi-head neural network. Each head of the neural network represents a separate task. Therefore, we select the head for a given task.

4. Experiments, results, and discussion

In this section, we first discuss the RL environment setup and provide details of data collection for offline RL. Further, we evaluate the performance of SI with varying numbers of object configurations and densities for different task ordering.

Table I. Details of the collected 6 datasets for each task. The number of trajectories decreases as the area of the object space decreases to maintain a consistent object-space density.

Object Space Density (objects/cm ²)	Object-space Area (cm ²)	Number of Trajectories	Number of Data Points
20	1000	20k	400k
20	360	12k	240k
20	40	4k	80k
10	1000	10k	200k
10	360	6k	120k
10	40	2k	40k

4.1. Experimental setup

Our experimental setup is based on a simulated environment, Roboverse, used in ref. [36]. It is a GYM [30]-like environment based upon open-source physics simulator py-bullet [40]. We collected data for three tasks using this simulated environment.

4.1.1. Object space

We define *object space* as a subset of the robot's workspace where the task's target object is to be placed. In our case, it is a rectangular area on the table before the robot. When initialising the task, the target object is randomly placed in the object space. Fig. 4 object space of all three tasks is visible as a rectangle on the table.

4.1.2. Tasks definitions

We selected three tasks for all our experiments with some similarities. In each task, an object is placed in front of the robot on the table. At the start of every new episode, the object's initial position is randomly changed within the object-space area. Completion of a task requires some interaction with the object. Task definitions are given below.

1. *Press Button*: Button is placed in the object space. The objective of the task is to press the button. This is the easiest task, as this task can be seen as a go-to-goal task where the goal point is the point on the button in pressed configuration.
2. *Pick Shed*: This task aims to pick the object successfully. Thus, the robot also needs to learn to close the gripper at a specific position, apart from reaching the object.
3. *Open Drawer*: The objective of this task is to open the drawer.

4.1.3. Data collection

To examine the impact of the object-space area and object placement density, we collected a dataset for each combination of the object-space area (40 cm², 360 cm², and 1000 cm²) and the object placement density (10 and 20 placements per cm²) resulting in a total of six datasets for each task. Table I shows the quantitative details of 6 datasets for each task. It shows the number of trajectories and images collected for all combinations of object density and size of area. The length of a single episode (single trajectory) is 20; therefore, 20 data points per trajectory are collected. Format of each data point is $\langle s_t, a_t, r_t, s_{t+1} \rangle$, here s_t is $48 \times 48 \times 3$ RGB image. Fig. 4 displays trajectories (in green colour) and reward distribution across object space in the dataset for all three tasks, with an object-space area of 360 cm² and a density of 20 object configurations per cm². It can be seen that when the object is placed closer to the

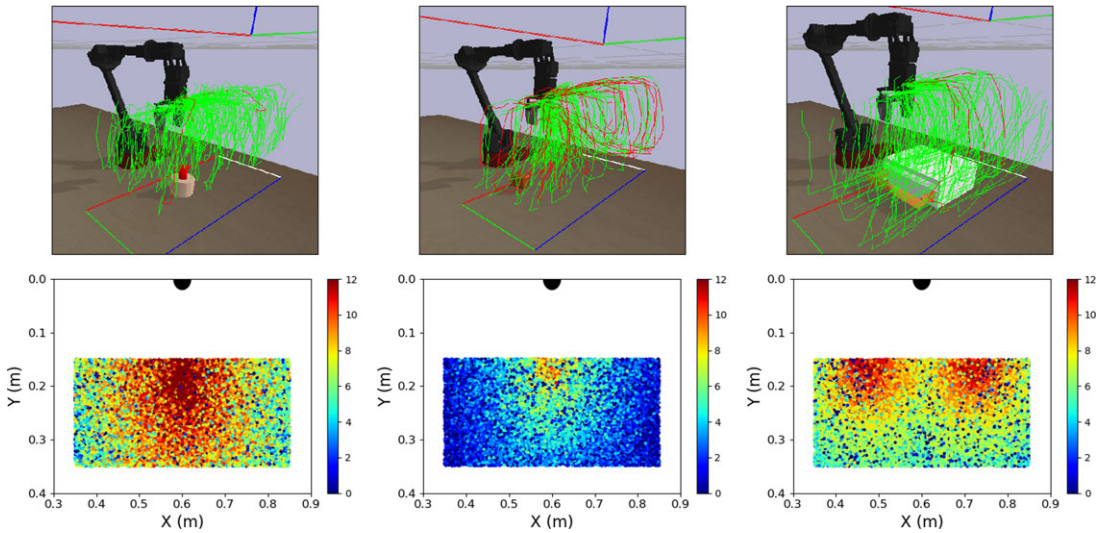


Figure 4. Top row displays sampled trajectories and bottom row displays the scatter plot of the reward distribution for tasks button-press, pick-shed and open drawer with object-space area 1000cm^2 and density 20 object configurations per cm^2 . The robot's base is at $(0.6\text{m}, 0.0\text{m})$, which is shown as a black semicircle.

robot, the reward is high as the task is completed in a few steps, while it becomes low as the object moves away.

Hand-designed policies. We collect data by employing hand-designed policies. The core of the hand-designed policies revolves around action selection, where the policy decides the next action based on the robot's current state and the task's requirements. These actions include moving the end-effector, closing/opening the gripper, lifting the end-effector upward, or halting all movement. The policy has full access to simulation and thus can use various parameters required for task completion, which are otherwise unavailable to the RL agent. We use the following hand-designed policies

1. **Press-button Policy:** The policy first calculates the distance between the gripper and the button. It moves the gripper towards the top of the button until a certain threshold is crossed. Once the gripper is on the top of the button, it is moved down to press it. The state of the button is continuously monitored at every step. The task is considered successful if the button is pressed and a reward of 1 is awarded per step until the end of the episode.
2. **Pick-shed Policy:** The policy calculates the distance between gripper and object. If the distance is greater than a threshold, the gripper is moved in the direction of the object. If the gripper and object are close enough, the policy gives action to close the gripper. Then, the object is lifted up in the z direction until a certain height threshold.
3. **Open Drawer Policy:** Similar to the above policy, this policy also calculates the distance between the drawer handle and gripper. The gripper is moved towards the door handle until a certain threshold. Once the gripper is above the handle, the gripper is closed. Finally, the gripper is moved in the direction of opening the drawer. This direction depends on the drawer's orientation, which is provided in the policy from the simulation.

Our data collection procedure, as defined in algorithm 1, follows the standard agent-environment loop. We initiate an outer loop to iterate over the total number of trajectories (or episodes) N_T , which is determined by considering the area and density of the object space, ensuring a consistent density across different areas. Each episode consists of 20 time steps. During each time step, the hand-designed

Algorithm 1. Data Collection Procedure.

```

data ← [];
n ← 0;
while n ≤ NT do
  t ← 0;
  while t ≤ 20 do
    st ← env.observation();
    at ← policy.action(env) + N(0, 1);
    st+1, rt ← env.step(at, st);
    data ← data + < st, at, rt, st+1 >;
    t ← t + 1;
  end
  n ← n + 1;
end

```

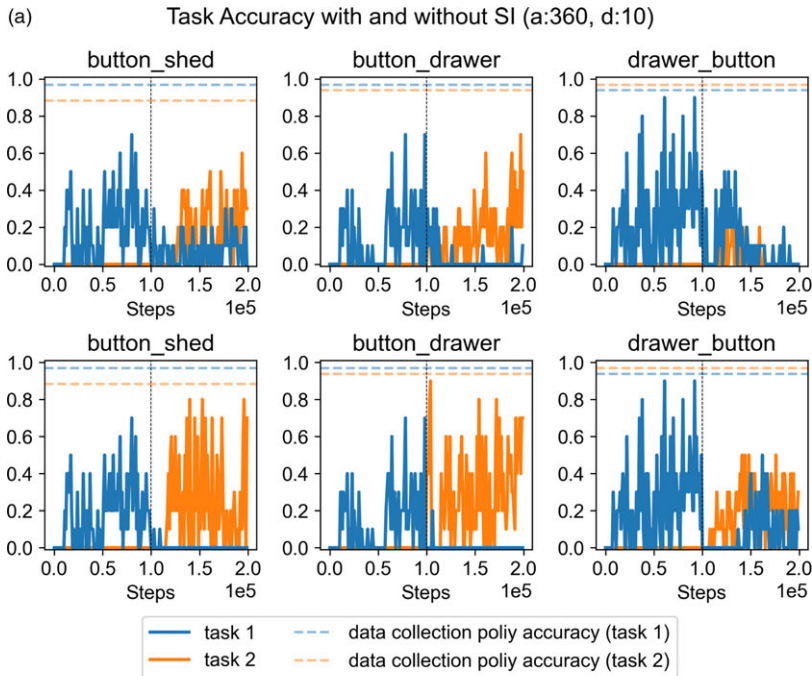
policy generates an action (a_t) based on the current environment state. Then, a Gaussian noise (N) is added to the action. The purpose of this noise is twofold: it reduces the policy's accuracy from 100% to approximately 80% so that we get both successful and failure cases, and it introduces variations in the trajectories. The noisy action is then provided to the simulator, which produces the next state (s_{t+1}) and the corresponding reward (r_t). We store this information as a typical tuple $\langle s_t, a_t, r_t, s_{t+1} \rangle$, which is commonly utilised in reinforcement learning.

4.2. Empirical results and analysis

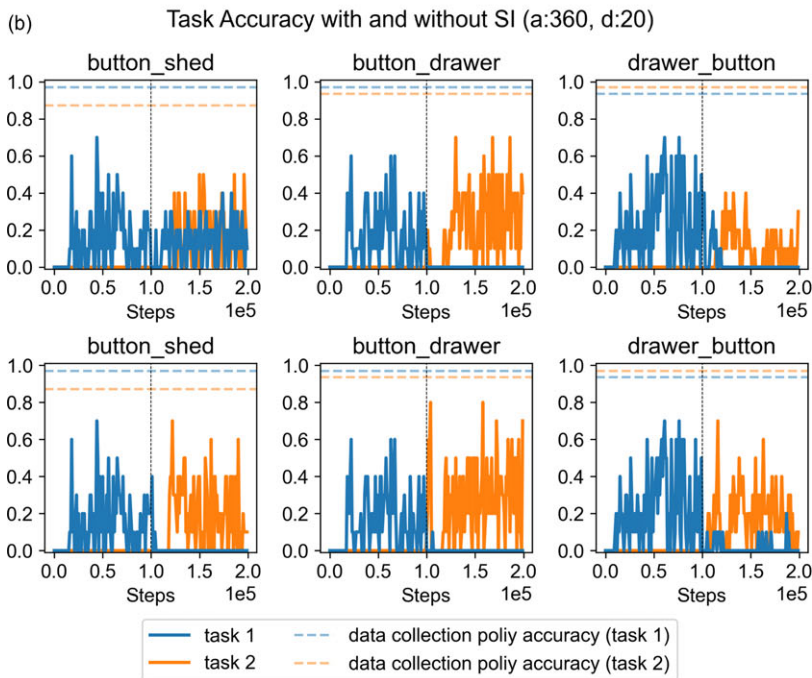
For one sequential learning experiment, we select a sequence of two tasks from the three tasks set, as mentioned in the previous section. This selection yields six possible combinations: button-shed, button-drawer, shed-button, shed-drawer, drawer-shed, and drawer-button. We perform two sets of experiments for each doublet sequence, one with SI regularisation and another without SI regularisation. Each set contains six experiments by varying the area and density of object space. Therefore, in total, we performed 72 experiments of sequential learning. Apart from these 72 experiments, we also trained the agent for single tasks using SAC-CQL for reference baseline performance to evaluate forward transfer. We do behaviour cloning for the initial 5k steps to learn faster as we have a limited compute budget. We use metrics mentioned in ref. [31] for evaluating the performance of a continual learning agent. Each task is trained for $\Delta = 100K$ steps. The total number of tasks in a sequence is $N = 2$. Total steps $T = 2 \cdot \Delta$. The i -th task is trained from $t \in [(i - 1) \cdot \Delta, i \cdot \Delta]$.

4.2.1 Task accuracy

We evaluate the agent after every 1000 training steps by sampling ten trajectories from the environment for each task. The agent's accuracy $p_i(t)$, for a task i , is defined as the number of successful trajectories out of those ten trials. Fig. 5 shows the accuracy of three experiments corresponding to button-shed, button-drawer, and drawer-button task combinations for the area size of $40cm^2$ with a density of 10 and 20 object configurations per cm^2 . The top row represents sequential learning with SI, while the bottom represents sequential learning without SI. SMMMMMMMMMI is working better, as evidenced by overlapping Task-1 and Task-2 accuracy. We observed that SI was most helpful in button-shed task doublet due to the overlapping nature of these tasks, as both require reaching the object. This shows the benefit of using SI for overlapping tasks.



Task Accuracy (area=360cm², density=10 objects/cm²)



Task Accuracy (area=360cm², density=20 objects/cm²)

Figure 5. Task accuracy for tasks button-shed, button-drawer and drawer-button. The top row is with SI, and the bottom row is without SI. In each plot, the X-axis represents the number of gradient update steps, and the Y-axis represents accuracy.

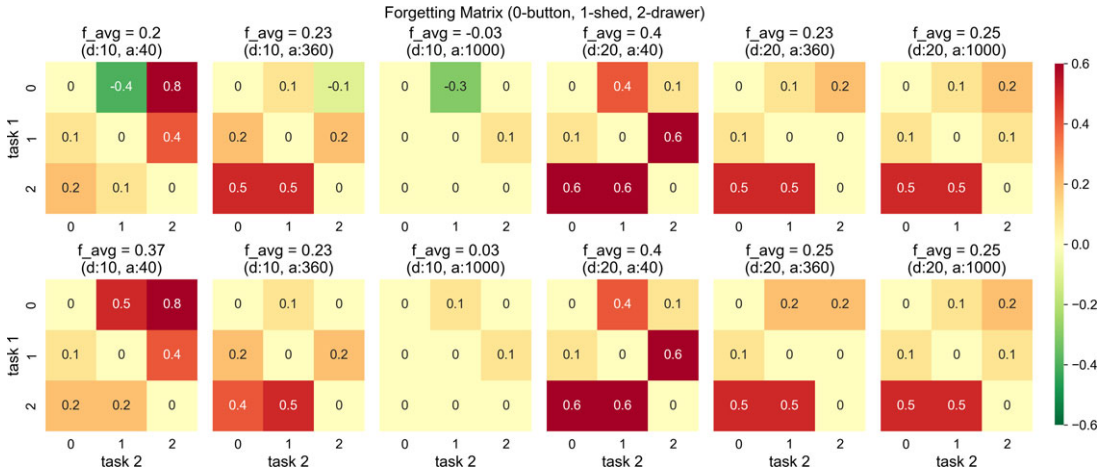


Figure 6. Forgetting matrix (0-button, 1-shed, 2-drawer). The top row is with SI regularisation, and the bottom row is without regularisation. Measuring units for area and density are cm² and objects/cm² respectively for every task.

4.2.2 Forgetting

It measures the decrease in accuracy of the task as we train more tasks and is defined as $F_i := p_i(i, \Delta) - p_i(T)$. Here, $p_i(t) \in [0, 1]$ is the success rate of task i at time t . Fig. 6 shows the forgetting of Task-1 after training Task-2. We can see that SI performed better or equal in all cases. In fact, in some cases, like button-shed forgetting is negative, which means that the performance of Task-1 improved after training on Task-2. This indicates knowledge transfer from Task-1 to Task-2. This phenomenon is not seen in the case of sequential learning without SI. This indicates that SI helps in reducing catastrophic forgetting. No significant trends are observed in the variation of object-space area, but forgetting increases with increased object-space density. This might be due to the limited computing budget (100K) per task, as tasks with more area size and density would require more training to show good results.

4.2.3 Forward transfer

It measures knowledge transfer by comparing the performance of a given task when trained individually versus learning the task after the network is already trained on previous tasks and is defined as

$$FT_i := \frac{AUC_i - AUC_i^b}{1 - AUC_i^b}, \tag{10}$$

where $AUC_i = \frac{1}{\Delta} \int_{(i-1)\Delta}^{i\Delta} p_i(t) dt$ represents area under the accuracy curve of task i and $AUC_i^b = \frac{1}{\Delta} \int_0^\Delta p_i^b(t) dt$ represents area under curve of the reference baseline task. $p_i^b(t)$ represents reference baseline performance. We use single-task training performance as the reference for Task-2 while evaluating forward transfer. Fig. 7 shows forward transfer for Task-2 after it is trained on Task-1. We observed that in most cases, training without SI gives a better transfer ratio than training with SI. Since reducing catastrophic forgetting is the primary objective of the sequential learning framework, we set a high value of SI regularisation strength. This restricts the movement of weights from the solution of the previous task, which helps to reduce catastrophic forgetting but also hinders the ability to learn new task thus reducing forward transfer. This can also be noticed in Fig. 5, where the accuracy of Task-2 is lower for SI than its non-SI counterpart. This highlights the problem of stability-plasticity; any method that tries to make learning more stable to reduce forgetting inadvertently also restricts the flexibility of the connectionist model to learn a new task.

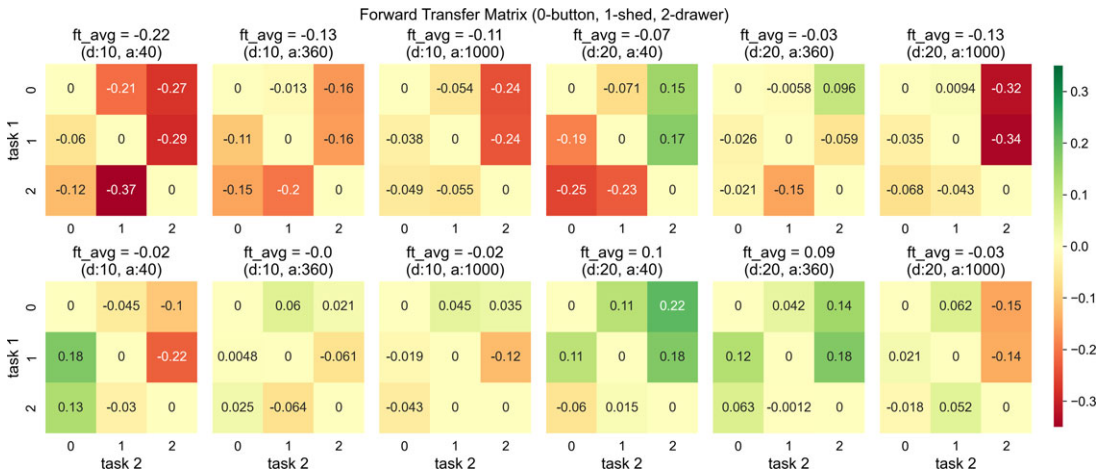


Figure 7. Forward transfer matrix (0-button, 1-shed, 2-drawer). The top row is with SI regularisation, and the bottom row is without regularisation. Measuring units for area and density are cm^2 and $\text{objects}/\text{cm}^2$ respectively for every task.

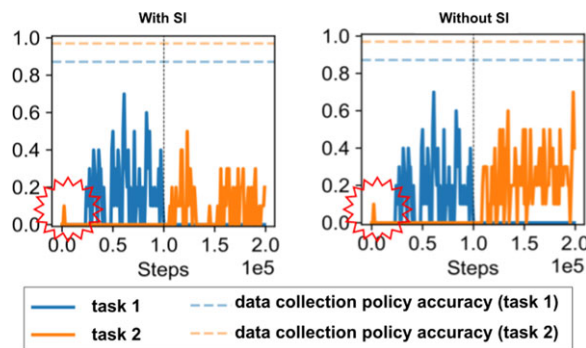


Figure 8. Accuracy for sequentially learning pick-shed and press-button tasks (area = 360cm^2 , density = $20\text{objects}/\text{cm}^2$). The left column is with SI, and the right column is without SI. In each plot, the X-axis represents the number of gradient update steps, and the Y-axis represents accuracy.

4.2.4 Training time

We used NVIDIA DGX A100 GPU for training. Training time for one experiment with three sequential tasks on 1 GPU is 18h (6h per task). We used 8 GPUs in parallel for training and testing all 72 experiments, which took approximately 7 days. Apart from these metrics, we observed that the agent requires 14k, 10k, and 16k steps on average to achieve its first success on Task-2 when trained directly, sequentially without SI, and sequentially with SI, respectively. This shows the advantage of the sequential training framework as the agent learns the task faster when trained sequentially (without SI) than when directly training the task, but the agent slows down a little when we add SI to reduce forgetting.

Fig. 8 shows another interesting observation we made in the case of sequential learning of pick-shed and press-button (area = 360cm^2 , density = $20\text{objects}/\text{cm}^2$) tasks. While training for Task-1 (pick shed), the agent showed some success on Task-2 (press button) even before getting any success on Task-1 itself. This might be due to the nature of the tasks, as the trajectory of the press-button task is common for another task. Therefore, the agent tends to acquire knowledge for similar tasks. This may also result from behaviour cloning for the initial 5k steps, where the agent tries to mimic the data collection policy for a few initial training steps. Also, we observed that increasing the object-space area (keeping the density

the same) helps in knowledge transfer, which the increase can be seen in average forward transfer with area size.

5. Conclusion and future work

We investigated catastrophic forgetting and forward knowledge transfer for sequentially learning image-based robotic manipulation tasks by combining a continual learning approach with an offline RL framework. We use SAC-CQL as an offline deep RL framework with synaptic intelligence (SI) to mitigate catastrophic forgetting. Multiheaded CNN was used to provide knowledge of the current task index to the neural network. We performed a series of experiments with different task combinations and with a varying number of object configurations and densities. We found that SI is useful for reducing forgetting. However, it showed a limited transfer of knowledge from previous tasks. We also found that the ordering of tasks significantly affects the performance of sequential task learning. Experiments also suggest the importance of prior knowledge for continual learning. Agent trained only with state-action pairs of many diverse tasks (even without reward) may provide better prior knowledge.

In addition to the findings presented in this work, there are several promising directions for extending the research on sequential learning of image-based robotic manipulation tasks. Firstly, the scope of our framework can be expanded to encompass a more comprehensive array of robotic manipulation tasks, thereby demonstrating its versatility and applicability across diverse manipulation domains. This extension would involve testing the agent's ability to continuously acquire new skills, even when the task set evolves. Secondly, the order in which tasks are presented to the agent can significantly influence the learning process. Therefore, investigating curriculum learning strategies that dynamically arrange the sequence of tasks to optimise knowledge transfer represents a valuable avenue for future exploration. Finally, we used a simulator for data collection. However, simulation may not account for varying lighting conditions, distribution of natural images, unmodelled dynamics, and uncertainty of the real world. These factors introduce a discrepancy between the simulated and real-world scenarios, known as the reality gap. As such, the future will also focus on datasets collected from real hardware for a more comprehensive understanding of continual learning in robotic manipulation in real-world scenarios.

Author contributions. All three authors, Sudhir Pratap Yadav, Rajendra Nagar, and Suril V. Shah, were involved in conceptualising the problem, discussing solutions, and writing the paper. At the same time, Sudhir Pratap Yadav carried out data collection and experiments.

Financial support. This work was done in collaboration with IIT Jodhpur and the iHub Drishti Foundation, IIT Jodhpur.

Competing interests. The author(s) declare none.

Data availability statement. The data and code supporting this study's findings are openly available at <https://github.com/sudhirpratapyadav/sac-cql-si>; further inquiries can be directed to the corresponding author/s.

References

- [1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing atari with deep reinforcement learning," (2013). arXiv preprint arXiv: [1312.5602](https://arxiv.org/abs/1312.5602), 2013.
- [2] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan and V. Vanhoucke, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation, (2018). arXiv preprint arXiv: [1806.10293](https://arxiv.org/abs/1806.10293), 2018.
- [3] C. Devin, A. Gupta, T. Darrell, P. Abbeel and S. Levine, "Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer," *In: International Conference on Robotics and Automation (ICRA)*, (2017) pp. 2169–2176.
- [4] S. Gu, E. Holly, T. Lillicrap and S. Levine, "Deep reinforcement learning for robotic manipulation," (2016). arXiv preprint arXiv: [1610.00633](https://arxiv.org/abs/1610.00633) 1, 2016.
- [5] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel and S. Levine, "Composable Deep Reinforcement Learning for Robotic Manipulation," *In: International Conference on Robotics and Automation (ICRA)*, (2018) pp. 6244–6251.

- [6] M. Gualtieri, A. ten Pas and R. Platt, "Category level pick and place using deep reinforcement learning," *Computing Research Repository* (2017). arXiv preprint arXiv: [1707.05615](https://arxiv.org/abs/1707.05615).
- [7] L. Berscheid, P. Meißner and T. Kröger, "Self-supervised learning for precise pick-and-place without object model," *Robot Automa Lett* **5**(3), 4828–4835 (2020).
- [8] A. Lee, C. Devin, Y. Zhou, T. Lampe, K. Bousmalis, J. Springenberg, A. Byravan, A. Abdolmaleki, N. Gileadi and D. Khosid, "Beyond Pick-and-Place: Tackling Robotic Stacking of Diverse Shapes." *In: Conference on Robot Learning*, (2021).
- [9] J. Bao, G. Zhang, Y. Peng, Z. Shao and A. Song, "Learn multi-step object sorting tasks through deep reinforcement learning," *Robotica* **40**(11), 3878–3894 (2022).
- [10] X. Wu, D. Zhang, F. Qin and D. Xu, "Deep reinforcement learning of robotic precision insertion skill accelerated by demonstrations," *In: International Conference on Automation Science and Engineering (CASE)*, 1651–1656, (2019).
- [11] A. Yasutomi, H. Mori and T. Ogata, "A Peg-in-Hole Task Strategy for Holes in Concrete," *In: International Conference on Robotics and Automation (ICRA)*, (2021) pp. 2205–2211.
- [12] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. Ojea, E. Solowjow and S. Levine, "Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards," *In: International Conference on Intelligent Robots and Systems (IROS)*, (2020) pp. 5548–5555.
- [13] B. Nemeč, L. Žlajpah and A. Ude, "Door Opening by Joining Reinforcement Learning and Intelligent Control," *In: International Conference on Advanced Robotics (ICAR)*, (2017) pp. 222–228.
- [14] Y. Chen, C. Zeng, Z. Wang, P. Lu and C. Yang, "Zero-shot sim-to-real transfer of reinforcement learning framework for robotics manipulation with demonstration and force feedback," *Robotica* **41**(3), 1015–1024 (2023).
- [15] X. Sun, H. Naito, A. Namiki, Y. Liu, T. Matsuzawa and A. Takanishi, "Assist system for remote manipulation of electric drills by the robot WAREC-1R using deep reinforcement learning," *Robotica* **40**(2), 365–376 (2022).
- [16] A. Apolinarska, M. Pacher, H. Li, N. Cote, R. Pastrana, F. Gramazio and M. Kohler, "Robotic assembly of timber joints using reinforcement learning," *Automat Constr* **125**, 103569 (2021).
- [17] M. Neves and P. Neto, "Deep reinforcement learning applied to an assembly sequence planning problem with user preferences," *Int J Adv Manuf Tech* **122**(11–12), 4235–4245 (2022).
- [18] P. Kulkarni, J. Kober, R. Babuška and C. D. Santina, "Learning assembly tasks in a few minutes by combining impedance control and residual recurrent reinforcement learning," *Adv Intell Syst* **4**(1), 2100095 (2022).
- [19] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik and S. Levine, "Combining Self-Supervised Learning and Imitation for Vision-based Rope Manipulation," *In: International Conference on Robotics and Automation (ICRA)*, (2017) pp. 2146–2153.
- [20] R. Lee, D. Ward, A. Cosgun, V. Dasagi, P. Corke and J. Leitner, "Learning arbitrary-goal fabric folding with one hour of real robot experience (2020). arXiv preprint arXiv: [2010.03209](https://arxiv.org/abs/2010.03209).
- [21] A. Gupta, J. Yu, T. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin and S. Levine, "Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors Without Human Intervention," *In: International Conference on Robotics and Automation (ICRA)*, (2021) pp. 6664–6671.
- [22] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine and K. Hausman, "Scaling Up Multi-Task Robotic Reinforcement Learning," *In: Conference on Robot Learning (CoRL)*, (2021).
- [23] S. Sodhani, A. Zhang and J. Pineau, "Multi-Task Reinforcement Learning with Context-based Representations," *In: International Conference on Machine Learning*, (2021) pp. 9767–9779.
- [24] Y. Teh, V. Bapst, W. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess and R. Pascanu, "Distral: Robust Multitask Reinforcement Learning," *In: Advances in Neural Information Processing Systems*, (2017).
- [25] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman and C. Finn, "Gradient surgery for multi-task learning," *Adv Neur Infor Pro Syst* **33**, 5824–5836 (2020).
- [26] I. Goodfellow, M. Mirza, D. Xiao, A. Courville and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," (2013). arXiv preprint arXiv: [1312.6211](https://arxiv.org/abs/1312.6211), 2013.
- [27] A. Mallya and S. Lazebnik, "Packnet: Adding Multiple Tasks to a Single Network by Iterative Pruning," *In: Computer Vision and Pattern Recognition*, (2018) pp. 7765–7773.
- [28] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans Patt Anal Mach Intell* **40**(12), 2935–2947 (2017).
- [29] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceed Nat Acad Sci* **114**(13), 3521–3526 (2017).
- [30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "OpenAI gym," *Comp Res Reposit* (2016). arXiv preprint arXiv: [1606.01540](https://arxiv.org/abs/1606.01540).
- [31] M. Wołczyk, M. Zajac, R. Pascanu, Ł. Kuciński and P. Miłoś, "Continual world: A robotic benchmark for continual reinforcement learning," *Adv Neur Infor Pro Syst* **34**, 28496–28510 (2021).
- [32] M. Caccia, J. Mueller, T. Kim, L. Charlin and R. Fakoore, "Task-agnostic continual reinforcement learning: In praise of a simple baseline," (2022). arXiv preprint arXiv: [2205.14495](https://arxiv.org/abs/2205.14495), 2022.
- [33] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *In: International Conference on Machine Learning*, (2018) pp. 1861–1870.
- [34] R. French, "Catastrophic forgetting in connectionist networks," *Trends Cogn Sci* **3**(4), 128–135 (1999).
- [35] F. Zenke, B. Poole and S. Ganguli, "Continual Learning through Synaptic Intelligence," *In: International Conference on Machine Learning*, (2017) pp. 3987–3995.
- [36] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar and S. Levine, "Cog: Connecting new skills to past experience with offline reinforcement learning," (2020). arXiv preprint arXiv: [2010.14500](https://arxiv.org/abs/2010.14500).

- [37] A. Kumar, A. Zhou, G. Tucker and S. Levine, “Conservative q-learning for offline reinforcement learning,” *Adv Neur Info Pro Syst* **33**, 1179–1191 (2020).
- [38] H. Van Hasselt, A. Guez and D. Silver, “Deep reinforcement learning with double Q-learning,” *Proceed AAAI Conf Arti Intell* **30**(1), (2016). doi: [10.1609/aaai.v30i1.10295](https://doi.org/10.1609/aaai.v30i1.10295)
- [39] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” (2014). arXiv preprint arXiv: [1412.6980](https://arxiv.org/abs/1412.6980).
- [40] C. Erwin and B. Yunfei. *PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning* (PyBullet, 2016).