Efficient Hindsight Experience Replay With Transformed Data Augmentation

Jiazheng Sun and Weiguang Li*

School of Mechanical and Automotive Engineering, South China University of Technology Guangzhou, Guangdong, China

* *Corresponding author. E-mail: 1035805780@qq.com*

Received: Feb. 27, 2022; Accepted: Apr. 06, 2023

Motion control of robots is a high-dimensional, nonlinear control problem that is often difficult to handle using traditional dynamical path planning means. Reinforcement learning is currently an effective means to solve robot motion control problems, but reinforcement learning has disadvantages such as high number of trials and errors and sparse rewards, which restrict the application efficiency of reinforcement learning. The Hindsight Experience Replay(HER) algorithm is a reinforcement learning algorithm that solves the reward sparsity problem by constructing virtual target values. However, the HER algorithm still suffers from the problem of long time in the early stage of training, and there is still room for improving its sample utilization efficiency. Augmentation by existing data to improve training efficiency has been widely used in supervised learning, but is less applied in the field of reinforcement learning. In this paper, we propose the Hindsight Experience Replay with Transformed Data Augmentation (TDAHER) algorithm by constructing a transformed data augmentation method for reinforcement learning samples, combined with the HER algorithm. And in order to solve the problem of the accuracy of the augmented samples in the later stage of training, the decaying participation factor method is introduced. After the comparison of four simulated robot control tasks, it is proved that the algorithm can effectively improve the training efficiency of reinforcement learning.

Keywords: Reinforcement learning; Machine learning; Motion control; Data augmentation; component;

http://dx.doi.org/10.6180/jase.202402_27(2).0011

1. Introduction

The motion control problems of robots are often difficult to solve using traditional control algorithms due to high dimensionality and nonlinearity, while reinforcement learning algorithms have higher robustness and environmental adaptability and can deal with motion control problems more flexibly, and In recent years, reinforcement learning algorithms have been studied and applied more successfully in control problems such as robotic arms, multi-legged robots [1], unmanned vehicles [2], and unmanned ships [3].

However, there are still many problems with reinforcement learning, two of which are more influential, one of which is the reward sparsity problem [4], in reinforcement learning tasks, the reward value can only be obtained under special circumstances, such as the grasping problem of a robotic arm, where the reward value can only be obtained when the task is completed. Since reinforcement learning relies on the reward values obtained from experience to update the value function network, if the reward values are sparse, the value function takes a long time to get updated, which seriously affects the training efficiency. One approach is to manually design elaborate reward functions, but this approach is labor-intensive and requires multiple trials and errors, which is difficult to reuse.

Secondly, reinforcement learning requires a lot of trial and error to collect data and train them to get more successful intelligences, and on the one hand, such trial and error takes a long time, and if the task dimension is high and complex, this longer training process will affect the application of reinforcement learning. On the other hand, for some tasks, the cost of trial and error is high, and only a smaller number of real trial and error samples can be improved. Therefore, how to improve the efficiency of sample utilization becomes one of the directions to improve reinforcement learning algorithms.

Data augmentation is an idea commonly used in machine learning to improve sample utilization efficiency. In supervised learning, samples can be randomly flipped, rotated, noisy, etc., so as to expand the data set, improve training efficiency, and effectively reduce the problem of overfitting. In this paper, from this idea of data augmentation, a HER algorithm based on transformed data augmentation is proposed based on the reinforcement learning algorithm based on HER [5]. The algorithm is able to use the already acquired empirical data to expand the empirical data set by certain transformations so as to obtain the enhanced new data, thus greatly improving the training efficiency of the reinforcement learning algorithm.

2. Ralated work

2.1. Reinforcement Learning

The idea underlying reinforcement learning [6] is the existence of an agent that interacts with the outside world, and that agent observes information about the outside environment and makes the appropriate decisions to reach the next state and obtain a certain reward value. This process is known as a Markovian decision process. Markov process is a class of stochastic processes defined in probability theory, characterized by the fact that the state X_{t+1} at the next moment is only related to the state X_t at this moment and not to the state at the previous moment. Based on this definition, the action decision process added to the reinforcement learning control process results in a Markov chain decision process (MDP), represented by a quadruplet (S, A, R, S').

For each step of the Markov decision process, first, the intelligence obtains some information from the current environment, which constitutes the state information $S_t \to \mathrm{S}$ at the current moment, and then generates an action a based on the current policy π, by sampling or deterministic form, and executes the action. After executing the action, the agent shifts to the next moment's state S_{t+1} through the state transfer equation. At the same time the agent obtains the reward $R_{t+1} \in R$ for the next moment from the environment according to the reward function $\mathrm{S} \times \mathrm{A} \to R$. This way multiple MDP processes are connected in series until they finally reach the termination state, which is called a trajectory.

The training goal of reinforcement learning is to find such an optimal policy π^* that generates actions based on states that maximize the expectation of the cumulative reward value obtained for the whole trajectory afterwards. Usually, when calculating the expectation of the reward, the closer to the current moment the value of the reward should be higher, so a discount factor γ is added, and the cumulative expectation is called the value function, the expectation calculated according to the state is the state value function $V(S)$, and the expectation of the reward calculated according to the state action pair is the state action value function $Q(S, A)$. And the optimal policy π^* can make each decision process choose the action that obtains the maximum Q value, which can be expressed by the Bellman transfer equation as follows.

$$Q^*(s,a) = E\left[R(s,a) + \gamma \max Q^*\left(s',a'\right)\right] \tag{1}$$

2.2. DQN, DDPG, and HER

A deterministic strategy establishes a direct connection between the state space and the action space $\pi : \mathrm{S} \to \mathrm{R}$, while a random strategy generates a probability distribution about the action information based on the state information, and then samples an action value from the distribution action value $a_t \to \mathrm{R}$. Both have advantages and disadvantages. The randomness strategy can train faster, take into account the exploration function, and avoid getting into a local optimum situation. However, the disadvantage is that if the action space dimension is high or the action values are continuous, the stochastic strategy can only be implemented by discretizing the action values or some special methods, while the continuous deterministic strategy can directly output the action values, so the deterministic strategy is often used for control tasks that require continuous action values.

DQN [7] is a model-free reinforcement learning method applicable to discrete action spaces. In DQN, a neural network is used to estimate the action state value function $Q(s, a)$. The key point is the use of experience replay and target network to complete the training of the Q network. The experience replay employs a replay buffer to store each Markov decision process tuple (s_t, a_t, r_t, s_{t+1}). During the training process, one batch of data is taken from the replay buffer each time, and the Q value is obtained using the method of temporal difference, and then the parameters of the Q network are updated using this Q value with the method of gradient descent. In contrast, the target network is used to make the updating process more stable, and a Q network that was stable in the previous step, called the target Q network, is used for estimating the Q value, instead of the Q network that is currently being updated, and the parameters of the target Q network are updated after the training of a batch is completed.

The DQN algorithm is only applicable when the action space is discrete, and thus the application is limited. DDPG [8] is a reinforcement learning algorithm based on DQN, which combines a policy gradient approach such as PG, to form a decision maker-commentator architecture. There exists a decision maker network, i.e., a strategy network $\pi : \mathrm{S} \rightarrow \mathrm{A}$, which can receive state information and output action information directly; and an evaluation network $Q : \mathrm{S} \times \mathrm{A} \rightarrow R$, which receives state and action information and outputs the evaluated Q value. During the training process, the collected experience replay data is used to train the Q network to improve the Q network accuracy, and then the evaluation of the Q network is used to train the policy network so that the policy network is close to the optimal policy network. Thus, the problem of controlling the continuous action value task is solved. both DQN and DDPG are off-policy [9] reinforcement learning algorithms, i.e., their update training data does not depend on the data generated by a particular policy.

Building on the above off-policy reinforcement learning, the HER algorithm [5] borrows the idea of human trial-and-error in order to solve the previously mentioned reward sparsity problem. HER algorithm considers a goal-oriented reinforcement learning task, i.e., reinforcement learning to obtain a reward has a direct correlation with reaching a certain goal, and HER borrows the idea from UVFA [10] by setting G as the space of all possible goals, and each $g \in \mathrm{G}$ is related to the reward function $r_g : \mathrm{S} \times \mathrm{A} \rightarrow E$. From this reward function one can derive the value function $Q^{\pi}(s_t, a_t, g) = E[R_t \mid s_t, a_t, g]$ containing the target. The HER algorithm not only puts the transfer data $(s_t \| g, a_t, r_t, s_{t+1} \| g)$ containing the real target into the experience replay buffer, but also generates a pseudo-target g' and a pseudo-reward r' based on the state information, and puts the modified transfer data $(s_t \| g', a_t, r', s_{t+1} \| g')$ into the experience replay buffer. This solves the problem of not getting rewards in the early stage of training and improves the training efficiency.

2.3. Data Augmentation

2.3.1. Data augmentation for supervised learning:

The data augmentation [10] method refers to the expansion of the existing data set based on the existing data by analyzing the completeness of the data and using certain rules. Data augmentation has gained wide application in supervised learning machine learning tasks. For example, in image-based machine learning tasks, the images of the original dataset can be cropped, flipped and rotated, scaled, shifted, added with Gaussian noise, added with color dithering, and other different methods [11] to obtain new labeled data and expand the existing dataset. Among the natural language processing tasks [12], the enhanced dataset can be generated by adding noise to the original data; mastering the distribution of the original data and sampling new data from the distribution; training a language model based on the original data and then generating synonymic near-synonyms of the original data by the model.

2.3.2. Data augmentation for reinforcement learning

Reinforcement learning has a higher demand for data utilization efficiency than supervised learning, and there exists a lot of research on data enhancement for reinforcement learning, which is generally mostly based on image input reinforcement learning tasks, with the following representative algorithms.CURL [13] adopts a contrast learning approach, where the input image is first cropped in some way to produce a new series of state data. CURL contains two feature extraction networks, where the original data is first passed through the feature extraction network to generate features, which are directly used to train the agent, while the enhanced data is also passed through the feature extraction network to generate features, and the enhanced features are used to compare with the original features, and if the loss is within a certain range, the data is considered valid and can be put into the replay buffer. Thus, the dataset is expanded.

The RAD [14] and DrQ [15] algorithms are more straightforward, i.e., more data augmentation methods are applied to the input image, which is then fed into the intelligence for training. There is also a reinforcement learning method based on the world model [16], the idea of this algorithm is to use the obtained Markov decision transfer information to train a model neural network that simulates the environment while training the intelligent body. This network can predict the next state in a sentence of the input state action. If the world model is more accurate, a trajectory can be simulated by model sampling as data for training, thus improving the efficiency of data utilization. However, among the above data augmentation methods, those with models require additional training and longer time to obtain an accurate model, while the rest of the algorithms are based on image based state data inputs, all of which have the disadvantage of being inapplicable in robot motion control problems.

3. Hindsight experience replay with transformed data augmentation

3.1. Reinforcement learning environment for motion control tasks

In the control task of reinforcement learning, usually the agent does not have complete access to all the information about the environment, but only to a part of it, called "state information". At the same time, the environmental information needs to be processed and transferred to the agent in a certain form. Different forms of state can affect the training efficiency, action results, etc. The state data can be expressed in various forms, such as image data, position pose data, point cloud data, etc. It is also possible to characterize the abstraction of the state by some function, so as to form abstracted feature data and improve the training efficiency of reinforcement learning.

Excellent state design should be built on the basis of task design, extracting the factors in the environmental data that have critical and comprehensive influence on the task data, so as to provide more accurate state information to the agent. Therefore, task analysis is needed first, and the task environment covered in this paper is the motion control task of the robot. The motion control of a robot is characterized by high dimensionality and sparse reward functions, as well as high overlap between state information and task goals.

Therefore, in the reinforcement learning task studied in this paper, the positional information is used as the main state information expression, and at the same time, the positional information is also used as the expression of the target information to realize the connection between the state information and the target information. Specifically, the state space information S is a collection of a series of coordinate information x of the robot's own elements and environmental targets, etc., and some state data y that are independent of the coordinate system in the global coordinate system. This includes, but is not limited to, the position coordinates of each joint, velocity vectors, etc.

$$S = [x_1, x_2, x_3 \dots x_n, y_1, y_2, \dots y_n] \tag{2}$$

The target information G needs to be linked to the state information in the form of a function: G=f(S)

Taking the Fetch task in Openai's Gym [17, 18] environment as an example, the robot it controls is a 7-degree-of-freedom robotic arm with a grasping mechanism at the end, and the task goal is to make the arm's grasping mechanism reach the coordinates of the target object. Its state information, then, contains the following parts, showed in Table 1.

Table 1. State Information of Fetch

	State content
1	Cartesian coordinates of the gripping mechanism
2	Cartesian coordinates of the target object
3	relative coordinates of the target object
4	the state of the gripping mechanism (independent of the coordinates)
5	the angular state of the target object
6	linear velocity of the target object
7	rotational coordinates of the target object
8	linear velocity of the gripping mechanism
9	opening and closing speed of the gripping mechanism

3.2. Transformation consistency discussion

Here we consider the way humans think when performing control tasks. When humans finish performing an action, they will naturally associate that if the state space consistency is high, then the translation, flip and rotation of this control trajectory, should also be valid. For example, if we are driving in an open territory, when the car turns left to reach a certain place; then with the same starting point, we would assume that turning the car to the right would lead to a symmetric one target point.

In the training task of reinforcement learning, each training needs to utilize real trajectory data as the dataset, and the training is less efficient. By such an idea, the existing trajectory data can be transformed accordingly to form a new trajectory, thus forming a richer data set with limited data, which can significantly extend the data utilization efficiency of reinforcement learning, improve the training speed of reinforcement learning algorithm and enhance the robustness of the intelligence.

In a typical reinforcement learning control task, the control chain of reinforcement learning is composed of a Markovian control chain. In the training process of DDPG, the trajectories of this Markov chain are used for the training of two networks, one is the Q network, which receives data in state space with actions and outputs Q values corresponding to the state and R values in the trajectory; the other is the policy network, which receives a state space data and outputs an action. Where the trajectory plays a key role is the data when training the Q-value network. When we transform the state space to some extent, a new state S′ is obtained, and when the intelligence is located in the new transformed state, a new action is generated based on the new state.

Here, it is hoped that the Q network can be trained using the control chain of the transformed trajectory. Assuming that S is the original state and the transition is T, we have $S_t = ST$.In the Q-network training algorithm of DDPG, the update formula is that:

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s,d)-\mathcal{D}}{\mathrm{E}} \times \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) Q_{\phi_{uz}} \left(s', \mu_{\theta_{ux}} \left(s' \right) \right) \right) \right)^2 \right] \tag{3}$$

In this update formula, there are four main state spaces and action quantities involved, one of which is the original state s, the second is the action a performed, the third is the reward value r obtained, and the fourth is the state s′ to which the action is converted after execution. Then when the corresponding transformation of the state and space is performed, its update formula becomes:

$$L(\phi, D) = \underset{(s_t,a,r_t,d)-\mathcal{D}}{\mathrm{E}} \times \left[\left(Q_\phi \left(s_t, a \right) - \left(r + \gamma(1-d) Q_{\phi_{\phi_{sy}}} \left(s_t', \mu_{\theta_{\theta_s}} \left(s_t \right) \right) \right) \right)^2 \right] \tag{4}$$

It can be analyzed that the transformation inconsistency in the updated formula after conversion mainly comes from the following aspects.

- inconsistency of the preceding and following actions: if the original strategy is still used after the transformation is performed, then the action generated by the sampling will also change because the state has been transformed, and then the inconsistency of the action will be generated.
- Inconsistency in the state after performing the action: In the original Markov chain, each new state is generated by a real action, and when the original state is converted, whether the new state generated by the action can correspond to the new state in the converted Markov chain, thus generating inconsistency in the state after performing the action.
- Inconsistency of obtained rewards: In the original Markov chain, the current obtained reward value is the reward value obtained in the real environment, and after the conversion, if the original reward value is still used as the reward value in the update function, the authenticity of the reward value cannot be guaranteed, which results in the inconsistency of the reward value.
- inconsistency of temporal different Q values: in the update formula, the temporal difference method is used to estimate the subsequent Q values, so that the current reward value and the temporal difference Q value to find the current Q value. The Q-value utilizes the untransformed strategy, and its resulting estimate is also biased, thus generating the inconsistency of temporal different.

It can be seen that a variety of different situations arise after the conversion of the original state, and these can have an impact on the effectiveness of the training. Therefore, this paper proposes two methods to improve the converted states. One is the action consistency conversion, and the other is the decay participation factor. The specific methods for both are stated in detail in the next section.

3.3. Transformed data augmentation

3.3.1. State and goal transformation

The state information of the robot control task has been introduced in the previous section, and this section describes how to perform certain transformations on the state to obtain new state data and achieve data enhancement. There are three types of state transformations: random translation, random flip, and random rotation, and as described in the previous section, each state information is composed of a set of coordinates. Therefore, to perform a state transformation, we need to process the coordinates in the state information that will be transformed, and finally form the transformed state information.

First, the coordinates and rotation information in the state information need to be processed and transformed into the form of a positional matrix suitable for the transformation, and for the positional coordinates, the coordinates themselves are the position description of the change point. $P = \left[p_x, p_y, p_z \right]^T$ As for the pose description, the state information provides generally the Euler angular coordinates of the rotations, and the rotations along the X-axis Y-axis Z-axis are represented by γ, β, α respectively, then the rotation matrix of the three rotations is:

$$\begin{aligned} R_x(\alpha) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \\ R_y(\beta) &= \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \\ R_z(\gamma) &= \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{5}$$

The final pose coordinates are:

$$R = R_z(\gamma) R_y(\beta) R_x(\alpha) \tag{6}$$

The final pose description is stitched from the position and pose information into the pose matrix as.

$$B = [R, P] \tag{7}$$

The pose is then subjected to certain transformations.

1. random translational transformation: for the translational transformation, it is only necessary to add an identical random transformation value to each of the original position coordinates, with the pose unchanged, to complete the random translational transformation:

$$P_T = P + P_r \tag{8}$$

2. random rotational transformation: for the rotational transformation, it is necessary to multiply the bit-pose expression by the corresponding rotation operator, which is the same as the expression of the pose, and is the rotation matrix corresponding to the rotation of the corresponding angles around the X-axis, Y-axis and Z-axis, respectively, rz, rx, ry are the angles of each random axis.

$$B_T = \mathrm{Rot}(Z, rz) \times \mathrm{Rot}(B, rb) \times \mathrm{Rot}(y, ry) \times B \tag{9}$$

Since the rotation and translation algorithms can be transformed uniformly, the random translation rotation transformation will be processed uniformly in the algorithm of this paper, i.e., the state information will be transformed by random translation:

$$B_T = \mathrm{Rot}(Z, rz) \times \mathrm{Rot}(B, rb) \times \mathrm{Rot}(y, ry) \times P_r \times B \tag{10}$$

3. Random mirror transformation: Considering that the transformation along a random axis in the mirror transformation is difficult and not conducive to the processing of Markov chain for data enhancement, only the coordinates of the state information are mirrored along a certain coordinate axis in the global coordinate system. For example, mirroring along the X-axis means mirroring along the face formed by two coordinate axes YZ. Then, for the position information P, both of its YZ coordinate values remain unchanged, while the X-axis coordinate values become the original negative numbers.

$$P_T = \left[-P_x, P_y, P_z\right]^T \tag{11}$$

As for the attitude information, the Euler angles of the mirror axes remain unchanged, while the Euler angles of the other two axes become the original negative numbers, and then the relevant attitude matrix can be calculated according to the previous equation as follows.

$$R_T = R_{-\gamma} R_{-\alpha} R_{-\beta} \tag{12}$$

The above transformation can be uniformly expressed as some kind of random transformation T. For the state information, the absolute coordinate information and the rotation quantity among the states are extracted first, and the transformation is applied after getting the positional matrices to get the transformed positional matrices, and then these positional matrices are restored back to the coordinate information and the rotation quantity to get the transformed state information $S_T : S_T = T(S)$

3.3.2. Action Consistency Conversion

As mentioned above, if only the state information is flipped, it will cause inconsistency in the transformation, which will affect the effectiveness of data enhancement. Therefore, in order to maximize the efficiency of data utilization and reduce the inconsistency, each part of the Markov decision process needs to be transformed accordingly. A Markov decision process consists of $\{\mathrm{S}, \mathrm{a}, \mathrm{r}, \mathrm{S}'\}$, which requires the following transformations.

1. the state information after the action S'_T : for the enhanced data samples, it is necessary to maintain the consistency of their state information before and after the action, so the corresponding data in the transformed S′ need to be transformed in the same way as S to obtain the enhanced , i.e., $S'_T = T(S)$..

2. Transformation of action information: Since the state information before and after the action has been transformed accordingly, if the action is not transformed, the inconsistency of the action will occur. Therefore, we need to transform the action space according to its data form. The action space of the FetchReach task, for example, is a four-dimensional control information. The first three dimensions are the expected position coordinates of the gripping mechanism, while the last data controls whether to open or close the gripping mechanism. The last data is independent of the transformation and therefore does not need to be changed, while the desired position requires the same transformation $a_T = T(a)$ to be applied according to the transformation of the state information. For some tasks, the action information is not related to the position information, but only to the relative direction, e.g., the turn

signal given to the vehicle in the driving task does not need to be changed in the translation and rotation task, while the corresponding mirroring is performed in the mirroring task.

3. Transformation of reward and target: In the standard reinforcement learning training task, the current reward value obtained after the action and the Q value after the time difference are used to calculate the gradient to complete the training. After the transformation, there is no guarantee that the transformed state can still obtain the exact same reward value, but only assume that it is still equal, and then use the time-permitted difference method to complete the estimation of the reward value:

$$Q_T = r_t + \gamma Q\left(s_t', \pi\right) \tag{13}$$

4. In HER algorithm, then also need to transform the corresponding target value, because the target value is associated with the state, so the transformed state information, and then the corresponding target transformation, you can get the corresponding target information, and then from the target information, to obtain the corresponding pseudo reward value, can constitute the HER algorithm, after the transformation of the data.

$$g_T := S_T' \tag{14}$$

$$r_T' := r\left(S_T', a_T', g_T\right) \tag{15}$$

3.3.3. *Decay participation factor setting*

Through the transformation processing of the relevant Markov decision process above, the inconsistency of the before and after states and actions is solved here, but the inconsistency of the reward information still exists, so the enhanced data cannot be used as accurate training data, and can only be a tool to improve the training efficiency in the early stage. Alternatively, the enhanced data obtained by transformation for training is equivalent to a means of exploration, and in the later stages of training, it will instead become a side effect that reduces the efficiency of the algorithm due to its inaccuracy. Therefore, in this paper, borrowing from the $\epsilon-$ greedy method of reinforcement learning exploration, we set a participation factor ϵ that decreases gradually with the number of training steps, transform the data with higher probability at the beginning of training and use the augmented data for training, and transform with lower probability at the later stage of training, while using only the real training data for training.

Algorithm 1:Decay participation factor

```
$r_{refresh} = (R_{target} R_{threshold\,0}) / n_{steps}$
$\epsilon_{refresh} = 1 / n_{steps}$
if $\epsilon > 0$ and $R_{last} > R_{threshold}$ :
    $R_{threshold} += r_{refresh}$
    $\epsilon -= \epsilon_{refresh}$
```

In order to realize the participation factor can be decayed dynamically, a common method is to perform exponential decay, but this method is more rigid and not flexible enough. The reward-based dynamic decay method is used here, and the steps are as follows: set a target reward value R_{target} , an update reward value $R_{\text{threshold}}$, set a number of steps n_{steps} , divide the reward value and the factor into equal parts according to the number of steps, and whenever the average reward value reaches the update reward value, increase $R_{\text{threshold}}$ is increased by one and ϵ is decreased by one, and when the reward value reaches the target reward value, ϵ also reaches 1 . The pseudo code is as Algorithm 1 .

3.3.4. *Hindsight Experience Replay with Transformed Data Augmentation*

The flow of the algorithm is shown in the pseudo-code of Algorithm 2.

4. Expriment

4.1. Expriment environment

As mentioned earlier, the HER-based algorithm described in this paper is suitable for a reward-sparse and goal-oriented robot control environment. The robot environment in the openai gym is used here as the standard experimental environment, which uses MuJoCo [19] as the engine for the physical simulation. We also use the DDPG+HER provided by stablebaseline [20] as a base criterion for algorithm evaluation. We conducted experiments in four robot tasks (fetchReach-v1, fetchPush-v1,fetchSlide-v1, FetchPickAndPlace-v1) and compared them with the standard version of the HER algorithm.

- 1FetchReach-v1: As shown in Fig. 1, there is a target point in the state space, and the robot's end-effector needs to be moved to the target point by controlling the robot arm.

- 2FetchPush-v1: As shown in Fig. 2, a block exists in the space and a target point is set randomly on the table

Algorithm 2: Hindsight Experience Replay with Transformed Data Augmentation (TDAHER)

```
initialize:
Off-policy reinforcement learning algorithm A, DQN DDPG etc.
Reward function r: $S \times A \times G -> R$
Initialize A
Initialize replay buffer R
for episode = 1,M do:
    Random generate a goal g and a initial state $s_0$
    for t=0,T-1 do:
        Use A strategy to draw an action $a_t$:
        $\$ a_t \leftarrow \pi_t(s_t \| g)$
        Execute $a_t$ and observe a new state $s_{t+1}$
    end for.
    For t=0,T-1 do:
        $r_t := r(s_t, a_t, g)$
        Store transition data $(s_t \| g, a_t, r_t, s_{t+1} \| g)$ to replay buffer R
        Use current states generate goals:   .
        for $g' \in G$ do
            $r_t := r(s_t, a_t, g)$
            Store transition data $(s_t \| g, a_t, r_t, s_{t+1} \| g)$
             to replay buffer R
            Random draw number between 0~1, if less than ε,
            then:
                Apply M times random transformations to
                $(s_t \| g', a_t, r', s_{t+1} \| g')$ to obtain m transformed
                transition data:
                $(s_{tT} \| g'_T, a_{tT}, r'_T, s_{(t+1)T} \| g'_T) = T((s_t \| g', a_t, r', s_{t+1} \| g'))$..
                Store them to replay buffer R
            End for
        End for
        for t=1,N do
            Random draw a batch of data B from R
            Use A to train agent
        End for
        Refresh $\epsilon$
    End for
End for
```

at the same time. Control the robot's end-effector to push the block to the target location to get the reward.

- FetchSlide-v1: As shown in Fig. 3,there is a puck on the table and a target point at the far end of the table, you need to control the robot to push the puck and get a reward if the puck stops near the target position after sliding.
- FetchPickAndPlace-v0: As shown in Fig. 4, there is a block object on the table, you need to control the robot's end-effector to pick the object and place it on the target object.

The simulated robot in this task is a 7-degree-of-freedom robotic arm. Its state space has been described before, and its goal space is related to the state space by a fixed tolerance between the position of the object of interest (e.g., the position of the actuator or the position of the target object) and the desired position. If it is less than this tolerance, the goal is considered to be achieved and the corresponding

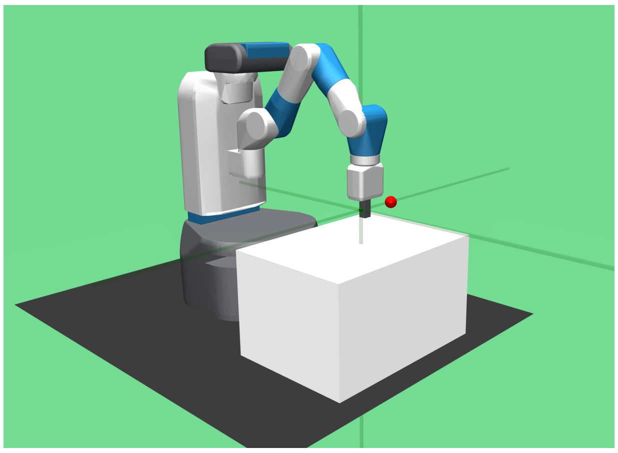
Fig. 1. FetchReach

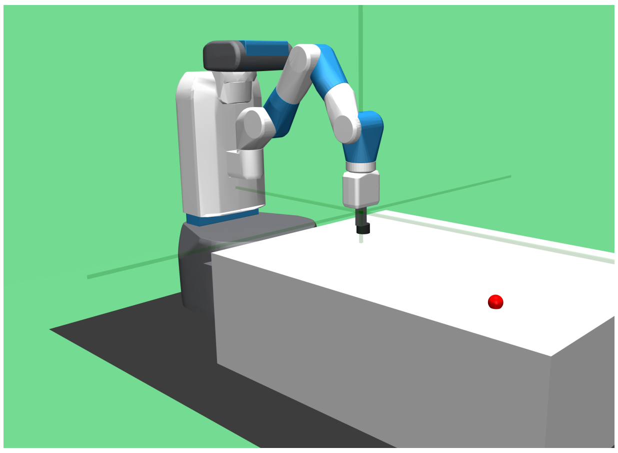
Fig. 2. FetchSlide

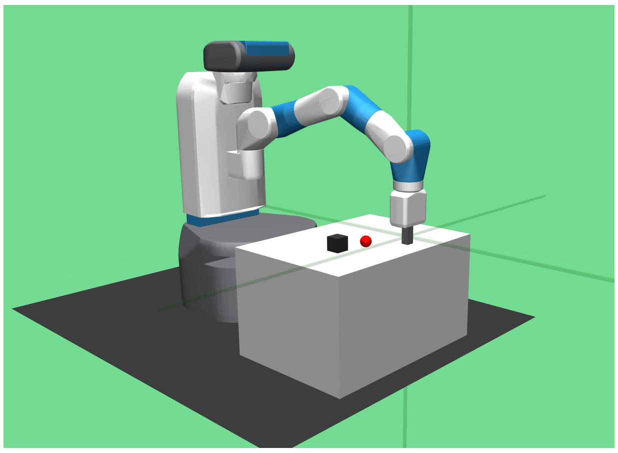
Fig. 3. FetchPush

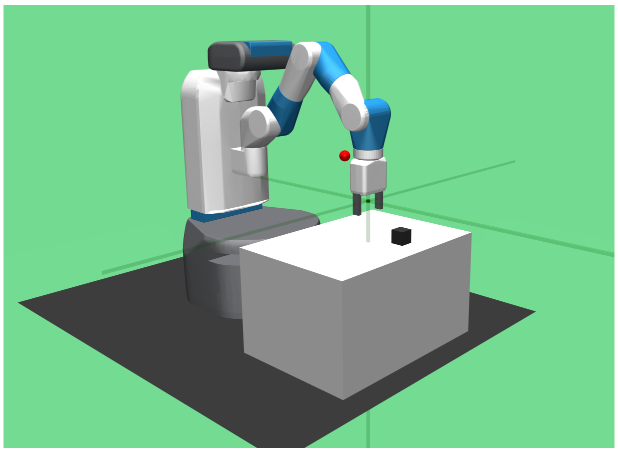

Fig. 4. FetchPick And Place

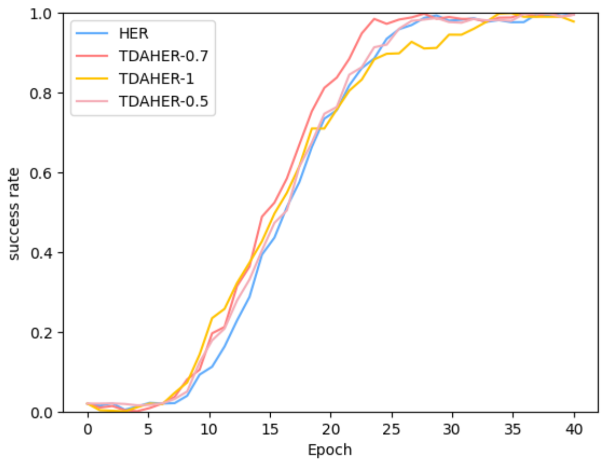

Fig. 5. FetchReach result

reward is given. The experiments were conducted on a windows-based computer with an Intel i7-10750 processor, nvidia rtx3060 graphics card, and 16GB DDR4 RAM. anaconda python 3.9 environment was used for the Python environment.

4.1.1. Expriment methodology

1. Expriment design Experimental task: In the above experimental setting we trained TDAHER separately from the standard HER algorithm in stablebaseline [20] for comparison. In each of them, different epochs were trained for different tasks according to the task difficulty, where feachreach was trained with 50 epochs, feachpush with 50 epochs, and feachslide and feachpeakandplace with 200 epochs. each of these epochs Each epoch contained 50 rounds of training.

   In the Feachreach task, the experiments were designed to simultaneously evaluate the standard HER algorithm against the TDAHER algorithm with different decay participation factors, with 1 full training session for each algorithm.

   In the other three tasks, the TDAHER algorithm used the best-performing attenuation factor, with 10 complete training sessions with each of the standard HER algorithms.

   Evaluation metrics: The main metric used to evaluate the algorithm in the experiments is the success rate, and the algorithm is used to make one attempt to complete the task in each epoch. The success rate is obtained by dividing the number of successes in each round by the total number of rounds attempted. Also, to evaluate the efficiency of the algorithms. The average time taken by each algorithm to reach a success rate of 0.8 was calculated to visualize the operational efficiency of different algorithms.

2. Expriment Hyperparameters All implementations are trained by the same random seeds under which the original HER algorithm is trained with the transformed data augmentation HER algorithm (TDAHER), and for comparison purposes, all training hyperparameters are kept the same. The capacity of the replay buffer is 10^6 and each batch contains 256 samples. In contrast, the networks of both Actor and critic have four layers of linear neural networks, each layer contains 256 neurons, and ReLU is used as the activation function. Its actor network passes a tanh activation function when outputting actions. Both networks use adam opimizer for optimaze and a learning rate of 0.001 is used. Also, in order to accelerate the training, a multi-process simultaneous training is used in the training process, and eight processes will be trained together at the same time.

3. Expriment Result

   In the Fetchreach task, result showed in Fig. 5, both the original HER algorithm and the TDAHER algorithm can be trained to complete the task relatively quickly due to the simplicity of the task. Here the comparison is made on what percentage of the update of the participation factor will be set to the highest reward value. As seen in the figure, if the target reward value is set to the same value as the highest reward value, it will cause a plateau period in the late training period, which will affect the training efficiency. If the target reward value is set to 0.5 of the highest reward value, the improvement is limited, although it is also improved in the early stage compared with the origi-

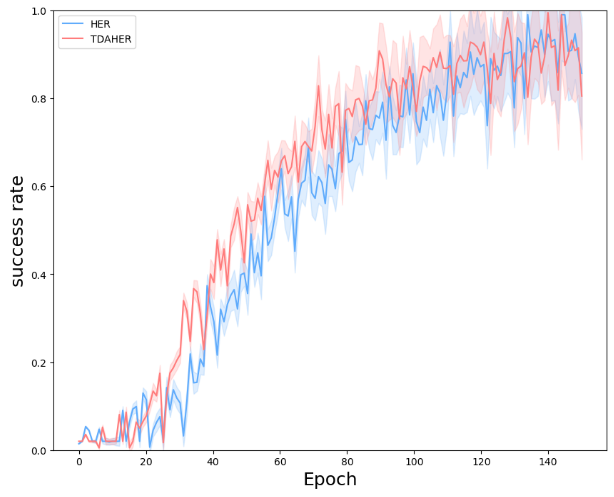

Fig. 6. FetchPush result

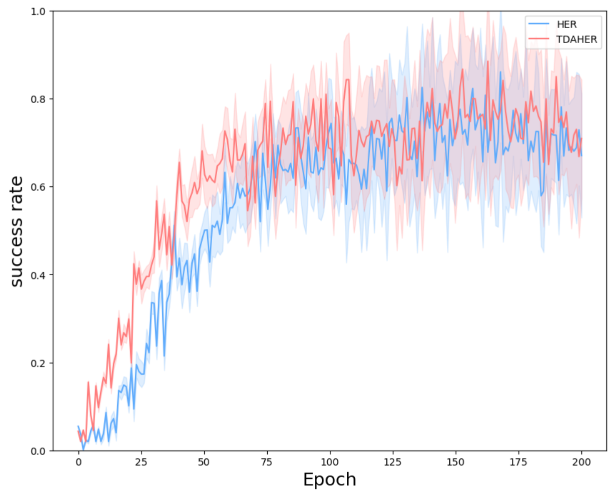

Fig. 8. FetchPickAndPlace result

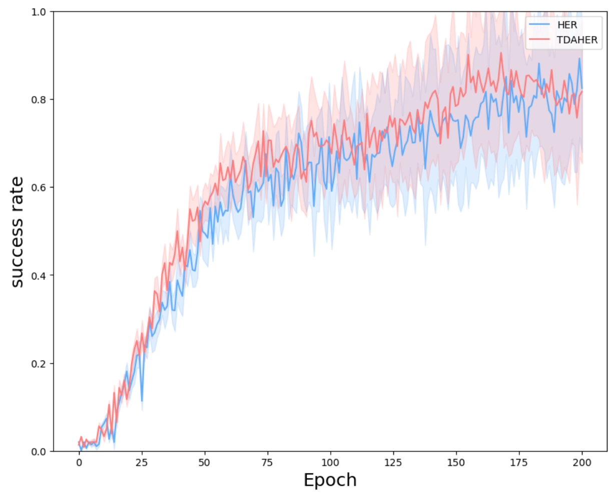

Fig. 7. FetchSlide result

nal HER algorithm; when the target reward value is set to 0.7 of the highest reward value, the improvement is most obvious in Fetchreach compared with the original HER algorithm. A success rate close to 1 is reached at 23epoch, while the original HER algorithm does not reach until after 26epoch.

In the fetchpush task, result showed in Fig. 6 , the TDAHER algorithm shows greater superiority compared to the original HER algorithm, achieving a high success rate of about 0.8 at 80 epoch, while the original HER algorithm takes up to 100 epoch to achieve.

In the Fetchslide experiments, result showed in Fig. 7, both algorithms take a considerable amount of time to reach a high success rate due to the increased difficulty of the task. At the beginning of the task training, the TDAHER algorithm is able to improve to a success rate of about 0.6 relatively quickly compared to the original HER algorithm, but after 70 epochs, the training speed of both TDAHER and HER algorithms starts to slow down. The success rates are close in this phase, but the TDAHER algorithm still maintains its superiority and has a relatively high success rate in the final phase of training.

In Fetchpickandplace, result showed in Fig. 8, it also takes 200 epochs to reach a high success rate, but in this task, TDAHER shows a higher superiority, especially in the early stage, compared with the original HER, TDAHER's training efficiency improves significantly, slows down in the middle stage, grows similarly to the original in the plateau stage, and reaches a higher success rate in the late The time to reach higher success rate in the later stage is also earlier than the original version. The variance of TDAHER data is also smaller and more stable than the original version.

We compare the average time to reach a success rate of 0.8, the average success rate of each Epoch, and the average highest success rate of the two algorithms in separate tables. Tables 1 to 4 show the performance of TDAHER and HER algorithms in FetchPush, FetchSlide, and FetchPickAndPlace respectively. As we can see, the highest success rate of evaluation is similar for FeachPush and FeachSlide, and TDAHER is higher for FeachPickAndPlace. In terms of average time and average success rate, TDAHER has a significant advantage over HER, with faster time and higher average success rate of the whole trajectory, which proves that TDAHER has higher training efficiency than HER.

4.2. Result analyse

The above experimental results show that TDAHER is more obvious in the improvement of training efficiency com-

Table 2. Feachpush Result

	FeachPush		
algorithm	Average time to reach 0.8 Success rate (epoch)	Average success rate	Average highest success rate
TDAHER	80.8	0.604	0.995
HER	87.3	0.538	0.996

Table 3. Feachslide Result

	FeachSlide		
algorithm	Average time to reach 0.8 Success rate (epoch)	Average success rate	Average highest success rate
TDAHER	148.9	0.617	0.905
HER	173.5	0.575	0.892

pared with the original HER algorithm, especially in the early stage of training, due to the more efficient use of data samples, which can make the agent faster to improve the success rate. In the middle and later stages of training, due to the difference between the data augmentation samples and the real samples, the inclusion of samples will instead affect the training efficiency, and by adjusting the decay of the participation factor, this effect can be weakened and more real samples can be used for training. In sum, data augmentation can improve the training efficiency of HER algorithm.

At the same time, the experiment also exposed a certain amount of problems. The TDAHER algorithm has limited improvement on the HER algorithm in the late training period, and the effective degree and variance of the TDAHER algorithm vary in the face of different tasks, which affects the effectiveness of the algorithm.

5. Conclusions

In this paper, we propose a method for data augmentation of robot control reinforcement learning task samples from the problem of efficient sample utilization for reinforcement learning algorithms, and based on this method, we propose a novel algorithm based on data augmentation in combination with HER algorithm. The algorithm is subjected to experimental analysis, and it is proved that the algorithm can obtain higher reward values more quickly and improve the training efficiency of HER algorithm effectively.

At the practical application level, the reinforcement learning training task for robot control, where the rewards are often sparse and the time to obtain a trajectory is long, can be effectively reused for samples using this algorithm. Improving the training efficiency of the reinforcement learning algorithm, shortening the training time, and enhancing its applicability in the field of robot control.

At the same time the algorithm also has some limitations, its data augmentation algorithm is based on the special environment of robot control task, in the reinforcement learning task, there are also a variety of lesser forms of state space, how to combine different state spaces, design a more general reinforcement learning data augmentation method, is the future research direction. In addition, the algorithm has only been trained in the simulator and has not been tested in a real robot, which is something that needs to be done in the future.

Acknowledgment

This study is supported by the Key-Area Research and Development Program of Guangdong Province (Grant No. 2019B090918003), the National Natural Science Foundation of China (NSFC, Grant Nos. 51875205 and 51875216), and International Science and Technology Cooperation Program of Guangdong Province (Grant No. 2017A050501004).

References

[1] N. Kohl and P. Stone. “Policy gradient reinforcement learning for fast quadrupedal locomotion”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*. **3**. IEEE. 2004, 2619–2624. DOI: 10.1109/robot.2004.1307456.

[2] E. Theodorou, J. Buchli, and S. Schaal. “Reinforcement learning of motor skills in high dimensions: A path integral approach”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, 2397–2403. DOI: 10.1109/ROBOT.2010.5509336.

[3] D.-H. Chun, M.-I. Roh, H.-W. Lee, J. Ha, and D. Yu, (2021) *“Deep reinforcement learning-based collision avoidance for an autonomous ship”* **Ocean Engineering *234***: 109216. DOI: 10.1016/j.oceaneng.2021.109216.

Table 4. Feachpickandplace Result

	FeachPickAndPlace		
algorithm	Average time to reach 0.8 Success rate (epoch)	Average success rate	Average highest success rate
TDAHER	**115.6**		
HER	127.4	**0.627**	**0.884**

[4] P. Rauber, A. Ummadisingu, F. Mutz, and J. Schmidhuber, (2021) *"Reinforcement Learning in Sparse-Reward Environments With Hindsight Policy Gradients"* **Neural Computation** *33*(6): 1498–1553. DOI: 10.1162/neco_a_01387.

[5] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, (2017) *"Hindsight experience replay"* **Advances in neural information processing systems** *30*:

[6] R. S. Sutton, A. G. Barto, et al., (1999) *"Reinforcement learning"* **Journal of Cognitive Neuroscience** *11*(1): 126–134.

[7] 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., (2015) *"Human-level control through deep reinforcement learning"* **nature** *518*(7540): 529–533. DOI: 10.1038/nature14236.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, (2015) *"Continuous control with deep reinforcement learning"* **arXiv preprint arXiv:1509.02971**:

[9] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, (2016) *"Safe and efficient off-policy reinforcement learning"* **Advances in neural information processing systems** *29*:

[10] D. A. Van Dyk and X.-L. Meng, (2001) *"The art of data augmentation"* **Journal of Computational and Graphical Statistics** *10*(1): 1–50. DOI: 10.1198/10618600152418584.

[11] C. Shorten and T. M. Khoshgoftaar, (2019) *"A survey on image data augmentation for deep learning"* **Journal of big data** *6*(1): 1–48. DOI: 10.1186/s40537-019-0197-0.

[12] M. Bayer, M.-A. Kaufhold, B. Buchhold, M. Keller, J. Dallmeyer, and C. Reuter, (2023) *"Data augmentation in natural language processing: a novel text generation approach for long and short text classifiers"* **International journal of machine learning and cybernetics** *14*(1): 135–150. DOI: 10.1007/s13042-022-01553-3.

[13] M. Laskin, A. Srinivas, and P. Abbeel. "Curl: Contrastive unsupervised representations for reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2020, 5639–5650.

[14] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, (2020) *"Reinforcement learning with augmented data"* **Advances in neural information processing systems** *33*: 19884–19895.

[15] I. Kostrikov, D. Yarats, and R. Fergus, (2020) *"Image augmentation is all you need: Regularizing deep reinforcement learning from pixels"* **arXiv preprint arXiv:2004.13649**:

[16] Y. Matsuo, Y. LeCun, M. Sahani, D. Precup, D. Silver, M. Sugiyama, E. Uchibe, and J. Morimoto, (2022) *"Deep learning, reinforcement learning, and world models"* **Neural Networks**: DOI: 10.1016/j.neunet.2022.03.037.

[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, (2016) *"Openai gym"* **arXiv preprint arXiv:1606.01540**:

[18] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, et al., (2018) *"Multi-goal reinforcement learning: Challenging robotics environments and request for research"* **arXiv preprint arXiv:1802.09464**:

[19] E. Todorov, T. Erez, and Y. Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[20] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. *Stable baselines3*. 2019.