Double-task Deep Q-learning with Multiple views

Jun Chen
chencherel@gmail.com

Tingzhu Bai
tzhbai@bit.edu.cn

Xiangsheng Huang
xiangsheng.huang@ia.ac.cn

Xian Guo
15120396@bitu.edu.cn

Jianing Yang
1100012928@pku.edu.cn

Yuxing Yao
yyxdawang78@buaa.edu.cn

Abstract

Deep Reinforcement learning enables autonomous robots to learn large repertories of behavioral skill with minimal human intervention. However, the applications of direct deep reinforcement learning have been restricted. For complicated robotic systems, these limitations result from high dimensional action space, high freedom of robotic system and high correlation between images. In this paper we introduce a new definition of action space and propose a double-task deep Q-Network with multiple views (DMDQN) based on double-DQN and dueling-DQN. For extension, we define multi-task model for more complex jobs. Moreover data augment policy is applied, which includes auto-sampling and action-overturn. The exploration policy is formed when DMDQN and data augment are combined. For robotic system's steady exploration, we designed the safety constraints according to working condition. Our experiments show that our double-task DQN with multiple views performs better than the single-task and single-view model. Combining our DMDQN and data augment, the robotic system can reach the object in an exploration way.

1. Introduction

Reinforcement learning methods have been applied to a wide range of robotic tasks from locomotion [6, 1] to manipulation [13, 15, 12, 4] and autonomous vehicle control [5]. Recently deep learning performs well in image classification and object detection especially from [7], and researchers focus on changing the architecture of neural network to tackle different tasks, [3, 9, 10]. For example, [14] extended single neural network with synthesis-loss function to complete double-task. [11] proposed deep Q-learning which combined Q-learning with deep learning to play Atari and to matched human performance. [17, 16] developed DQN to dueling-DQN and double-DQN based on [11] to reduce overestimation and split state-action value function into state value function and action advance value function. Deep reinforcement learning has become the most promising candidate algorithm for robotic auto-manipulation, because reinforcement learning equipped with convolutional neural network uses the raw pixels of images instead of robot's joint parameters.

[8] applied DQN in robotic manipulation and proposed an idea which converts search policy into supervised learning. [2] developed a way to reduce training time by parallelizing the algorithm across multiple robots. [18] defined a reward and terminate function for robotic system. But all of them used relatively simple robotic system. Moreover they defined the joint angles as action space and obtained the state information with single view angle. In practice, more complicated robotic systems have been applied for special propose, which indicates that it is difficult to use joint angles as action space of reinforcement learning due to high dimensional action space with continuous action values. In addition, using single camera can't describe the current state of robotic system and results in a high correlation coefficient between current state and next state.

In this paper, we define a new kind of discrete action space which uses target coordinate and pose angle of end-effector instead of joint angles to remedy the high dimensional action space. According to the definition of action space, we propose double-task Deep Q-Network with Multiple views, DMDQN, an off-policy and model-free control policy, based on double-DQN and dueling-DQN, as Figure 1 showed.

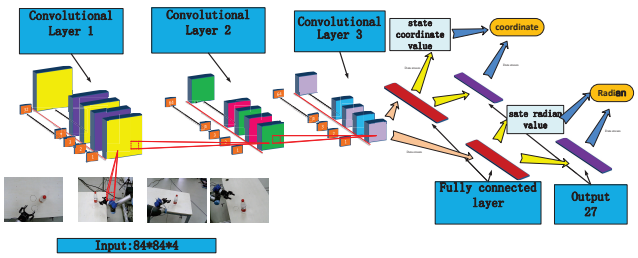

Figure 1. The architecture of DMDQN.

Double-DQN can tackle overestimations of the original DQN, and dueling-DQN is able to estimate the state value and each action-advance value. The double-task neural network shares the hyper parameter of convolutional layers, involving estimating state-coordinate action values and state-radian action values. Furthermore the double-task model can be extended to multiple tasks for more sophisticated works. To tackle the high correlation between images, we apply multi-view strategy which uses multiple cameras to sample from different view angles. Before being fed to neural network the frames are added random noise. Considering the dataset's covering a narrow part of the state space and absence of some actions, we develop a data augment policy including auto-sampling and action-overturn to update the replay memory. The auto-sampling policy requires only small dataset to pre-train our DMDQN and updates the replay memory through training and exploring repeatedly, and the action-overturn can double the replay memory without increasing required RAM. Combining DMDQN with data augment, the robotic system is able to explore the environment to attempt to complete the prior task effectively. We equipped the safety constraints to our algorithm system for steady exploration . In our experiments, the double-task model with multiple views performers better in a exploration way and the action-overturn can determinate what the terminate state of robotic system is.

Here we introduce the structure of this paper. In Section 2, we describe and formulate the original deep Q-learning and import two research achievements, double-DQN and dueling DQN in detail. In section 3, we expatiate our algorithm system, DMDQN, including the definition of action space in Section 3.1 and the architecture of DMDQN in Section 3.2. In Section 3.3, the training policy, reward and terminate function is described in details. Data augment policy in Section 3.4 is equipped to the training policy, which can update the replay memory while training. In Section 4, we did some experiments to testify our algorithm. In section 4.1, we compared performances of the double-task model to single-task model, while multi-view policy and single-view policy were compared in section 4.2. In Section 4.3, the models using the data augment and without it were put into comparison. In Section 5, we draw conclusions about our algorithm system and described the further research about the deep reinforcement learning in robotic system. We summarize our major contributions as follows:

(1) We propose a new definition of action space of robotic system and a new architecture of DQN which we named as a double-task Deep Q-Network with Multiple views, DMDQN;
(2) We combine DMDQN with data augment as a exploration policy to train the DMDQN while updating the replay memory and we design safety constraints for steady propose.

2. Background

In this section, we will formulate current robotic reinforcement learning problem and describe the existing algorithmic foundations on which we build our DMDQN based. The goal of reinforcement learning is to control an agent attempting to maximize reward function after executing an action which, in the context of robotic skills, denotes a user-provided definition of what robot should try to accomplish. The DQN, a topical example of DRL, satisfies both the autonomy and flexibility requirements for learning from exploration. It successfully learnt to play 49 different Atari 2600 games, achieving a human-level of control [11]. The DQN uses deep convolutional neural network to approximate state-action values which is defined by inputing raw pixels of frames, receiving rewards and outputting state-action value.

In robotic grasping system domain, for example, agent perceives images s_t consisting of M image frames: $s_t = \{I_{1,t}, I_{2,t}, I_{3,t}, I_{4,t}, \ldots, I_{M,t}\}$ at state step t and chooses an action from a discrete set $a_t \in A = \{1, \ldots, |A|\}$ and observes next state s_{t+1} and reward signal r_t according to the distance from the end-effector of robotic system to the object in grasping task.

The agent seeks to maximize the expected discounted return, where the discounted return is defined as $R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$. In this formulation $\gamma \in [0, 1]$ is discount factor that trades off the importance of immediate and future rewards. The goal is to find the optimal policy which maximizes the expected sum of returns from initial state distribution. And the state-value function and state function are defined as $Q^\pi(s, a) = E[R_t | s_t = s, a_t = a, \pi]$ and $V^\pi(s) = E_{a\sim\pi(s)}[Q^\pi(s, a)]$ respectively. The preceding state-action value function (Q function for short) can be computed recursively with dynamic programming:

$$Q^\pi(s, a) = E_{s'}[r + \gamma E_{a'\sim\pi(s')}[Q^\pi(s', a')] | s, a, \pi] \quad (1)$$

Such value iteration algorithms coverage to the optimal action-value function, $Q^\pi \to Q^*$ as $i \to \infty$. [11] built two networks named as target-network and Q-network, which have the same architecture and update target-network with Q-network every t training steps. A Q-network is trained by minimizing the loss functions $L_i(\theta_i) = E_{s,a\sim\pi}[(y_i - Q(s, a|\theta_i))]^2$ where $y_i = E_{s'\sim\varepsilon}[r + \gamma \max_{a'} Q(s', a'|\theta')]$ (θ_i is the hyper-parameter of Q-network, θ' is the hyper-parameter of target-network). The loss function is differentiated with respect to the hyper-parameter and optimized by stochastic gradient descent as follows:

$$\nabla_{\theta_i} L_i(\theta_i) = E_{s,a\sim\pi;s'\sim\varepsilon}[(r + \gamma \max_{a'} Q(s', a'|\theta') - Q(s, a|\theta_i)) \nabla_{\theta_i} Q(s, a|\theta_i)] \quad (2)$$

But the application of direct deep Q-learning so far has

been limited due to its overestimation of state-action values and inability to estimate the state value. [17, 16] developed DQN to dueling DQN and double DQN respectively. Dueling DQN splits the state-action value function into state value function and action-advance value function. And double-DQN can tackle overestimation of state-action value function.

The dueling architecture consists of two streams that represent the state value and action-advance value, sharing a common convolutional feature learning module. It's lower layers are convolutional as in the original DQN's. However, the following convolutional is two sequence (or streams) of fully connected layers instead of a single sequence stream, which are constructed such that they have the capability of estimating state value function and action-advance value function. Finally, these two streams are combined to produce a single output Q function for each action as follows:

$$Q(s,a|\theta,\alpha,\beta) = V(s|\theta,\beta) + (A(s,a|\theta,\alpha) - \frac{1}{|A|}\sum_{a'\in|A|} A(s,a'|\theta,\alpha) \quad (3)$$

And the training policy of dueling network is the same as the original DQN's.

The double DQN defines a new training policy instead of changing the architecture of the original DQN. The max operator in standard Q-learning and DQN uses the same values to select and evaluate an action and makes it more likely to select overestimated values, resulting in overoptimistic value estimates. To prevent this phenomenon, [16] proposed double Q-learning as follows:

$$y_t^{DoubleQ} = r_t + \gamma Q(s_{t+1} \arg\max_a Q(s_{t+1}, a|\theta_t)|\theta'_t) \quad (4)$$

The selection of action, in the $argmax$, is still due to the hyper-parameter θ_t of the Q-network. This means that, in Q-learning, we are still estimating the value of the greedy policy according to the current value defined by θ_t. However, [16] used the hyper-parameter θ'_t of target-network to fairly evaluate the value of this policy. The target-network is updated every τ steps as the original DQN does.

3. DMDQN

3.1. Action space

[2, 18] defined action space according to joint angles, which indicates the algorithms control robotic system through changing the joint-angles. But in complicated robotic system, Figure 2 shows the structure of robotic system, it is hard to control joint angles precisely, the relation of joint-angles is not arbitrary, which implies the joint-angles is dependent. If defining the action space through joint-angles, we must build constraint model and solve it accordingly which undoubtedly increases the complexity of our algorithm.

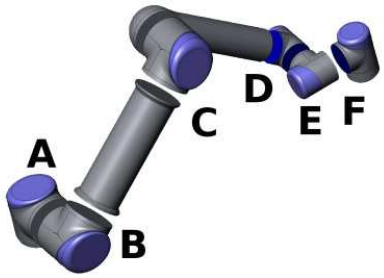

Figure 2. The structure of Joints of the robotic system: A: Base, B: Shoulder, C: Elbow and D, F, E: Wrist 1, 2, 3.

Our robotic system offers auto-movement policy which auto-builds stationary Cartesian Coordinate and we only need input the target coordinate (x_t, y_t, z_t) and pose angle (Rx_t, Ry_t, Rz_t) of end-effector to the robotic system then it moves to the target state automatically and safely. In order to make full use of it, we discretize the continuous space (coordinate and radian) to state point and the discrete distance is $coordinate\_step$ and $radian\_step$ which can be set on the basis of working condition of robotic system. In three-dimensional system of coordinate, each axis has three candidate state $\{-1, 0, 1\}$ (-1: negative direction; 0: no change; 1: positive direction). So we can get the next state(coordinate and radian) as follows:

$$\begin{cases} x_{t+1} = x_t + coordinate\_step * dx \\ y_{t+1} = y_t + coordinate\_step * dy \\ z_{t+1} = z_t + coordinate\_step * dz \end{cases} \quad (5)$$

$$\begin{cases} Rx_{t+1} = Rx_t + radian\_step * dRx \\ Ry_{t+1} = Ry_t + radian\_step * dRy \\ Rz_{t+1} = Rz_t + radian\_step * dRz \end{cases} \quad (6)$$

(dx, dy, dz) and (dRx, dRy, dRz) are the offset state of coordinate and radian respectively, dx, dy, dz, dRx, dRy, $dRz \in \{-1, 0, 1\}$.

The action spaces of coordinate and radian respectively have 27 candidate states. As Figure 3 shows the central point (red point) is the current state and the others are the candidate state.

3.2. DMDQN architecture

Due to the limits of DQN and it's variants mentioned above, this paper developed a double-task DQN with Multiple views. Recently most of reinforcement learning algorithms, such as [2, 18], based on computer vision applied in controlling robotic system, employ single camera, which leads to high correlation coefficient between current state and next state, because single camera is not enough

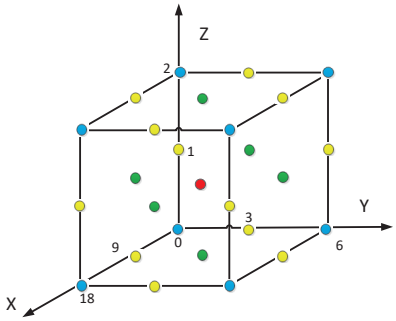

Figure 3. The red point is the current state, the others are the candidate states

to describe the current state of robotic system. Therefore in our robotic system we employ multi-view policy. Multi-camera with different observation angles records the process of robotic system reaching the object. In practice we used four cameras to sample data from four different directions. Figure 4 shows our robotic system equipped with four cameras.

Figure 4. The left is overlook about our robotic system. The right shows the position and observation of the four cameras

Although we define 27 candidate state action for coordinate and radian respectively, how to combine the action spaces of coordinate and radian is still of great deal. Generally, we combine them by multiply operation, generating 729 ($27*27$) candidate actions, which increases the computation of fully connected layers and the difficulty of training tremendously.

In convolutional network, convolutional layers aim to extract feature of images and state-action values are computed in the fully connected layers. So we constructed a double-task structure with sharing the hyper-parameters of convolutional layers.

In the DMDQN algorithm, as Figure 1 shows, four images from four different cameras is fed to the neural networks. At the end of network, there are two sets of fully connected layers, coordinate and radian respectively, which have the same structures but different hyper-parameters. The single network maps the raw pixels images to the state-action values of coordinate and radian respectively. We can execute $argmax$ operator to choose coordinate-action and radian-action.

3.3. Training Detail

To ease the training, the dataset, N experience tuples, is loaded to the replay memory. In training iteration, the stochastic mini-batch from the relay memory, which contains $\{s_t, s_{t+1}, a_{coor,t}, r_{coor,t}, a_{rad,t}, r_{rad,t}, t_t\}$ is fed to the neural network. The uniform sampling from the replay memory gives equal importance to all transition in the replay memory. s_t and s_{t+1}, containing four channels which are generated respectively by four cameras with different visual angle, refer to the current state and next state of robotic system; $a_{coor,t}, a_{rad,t}$ denote executing the action of coordinate and radian in state s_t; $r_{coor,t}, r_{rad,t}$ refer to feedback from robotic system after executing $a_{coor,t}$ and $a_{rad,t}$; t_t denote whether s_t is terminate state.

Our DMDQN uses double-task which maps the state of robotic system to the action of coordinate and radian, based on double-DQN and dueling-DQN. In the following description about our training detail, the actions mentioned below both include the coordinate action and radian action.

s_{t+1} is fed to the Q-network and the maximum state-action value is selected as follows:

$$\begin{aligned} a'_{coor,t+1} &= \arg\max Q_{coor,q}\left(s_{t+1}, a_{coor}|\theta_\tau\right) \\ a'_{rad,t+1} &= \arg\max Q_{rad,q}\left(s_{t+1}, a_{rad}|\theta_\tau\right) \end{aligned} \tag{7}$$

And s_{t+1} is input to the target-network and the state-action value $Q_{coor,target}(s_{t+1}, a'_{coor,t+1}|\theta_\tau)$ and $Q_{rad,target}(s_{t+1}, a'_{rad,t+1}|\theta_\tau)$ are choosen.

Then target values of Q-network is generated

$$\begin{aligned} {y'}_{coor} &= r_{coor,t} + \gamma Q_{coor,target}(s_{t+1}, a'_{coor,t+1}|\theta') \\ {y'}_{rad} &= r_{rad,t} + \gamma Q_{rad,target}(s_{t+1}, a'_{rad,t+1}|\theta') \end{aligned} \tag{8}$$

And the prediction value of Q-network, $Q_{coor,q}(s_t, a_{coor,t}|\theta_\tau)$ and $Q_{rad,q}(s_t, a_{rad,t}|\theta_\tau)$, is computed

We generate loss functions of coordinate and radian and synthesis-loss function, α is the loss coefficient

$$\begin{aligned} L_{coor}\left(\theta_\tau\right) &= \left[\left({y'}_{coor} - Q_{coor,q}\left(s_t, a_{coor,t}|\theta_\tau\right)\right)\right]^2 \\ L_{rad}\left(\theta_\tau\right) &= \left[\left({y'}_{rad} - Q_{rad,q}\left(s_t, a_{rad,t}|\theta_\tau\right)\right)\right]^2 \end{aligned} \tag{9}$$

$$L\left(\theta_\tau\right) = \alpha L_{coor}\left(\theta_\tau\right) + (1-\alpha) L_{coor}\left(\theta_\tau\right) \tag{10}$$

The synthesis-loss function is differentiated with respect to the hyper-parameter and optimized by stochastic gradient descent.

In reinforcement learning, the agent obtains a feedback and terminate signal from the environment after executing an action. The feedback in context of a robotic skill is a user-provided definition of what the task is.In reaching object domain, the distance to the terminate state can be computed easily, because each state information can be obtained from the robotic system. Some tricks about the reward and

terminate function were tried to our algorithm system, such as the distance directly, the functions with respect to the distance and so on. Finally we equipped the DMDQN with Algorithm 1.

Algorithm 1 Reward and terminate algorithm (Coordinate and Radian respectively)

for Sequence $j = 1, S$ **do**
  for State $i = 1, T$ **do**
    $\text{Dis}_i^j = \text{ComputerDistance}(P_i^j, P_T^j)$
    DisChange=Dis – PreviousDis
    $$R_i^j = \begin{cases} -1 & DisChange > 0 \\ 0 & DisChange = 0 \\ 1 & DisChange < 0 \end{cases}$$
    $R_c = R_i^j + R_{i-1}^j + R_{i-2}^j + R_{i-3}^j$
    if $R_c < 0$ **then**
      $t_i^j = 1$
    else
      $t_i^j = 0$
    end if
  end for
end for

3.4. Data augment and safety constraints

In our robotic system, we sampled dataset through controlling VR/AR device. Due to personal error, some trouble might happen, for example, changing the position of the object during sampling and the unstable velocity of robotic system lead to wrong dataset. In addition the dataset sampled artificially only covers a narrow part of the state space and several actions are absent. In reinforcement learning, the algorithm should allow the robot to explore the environment to finish special task and update the experience. In this section we present our data augment policy including auto-sampling and action-overturn. In practice we combine our DMDQN with data augment policy to form an exploration policy of robotic system, described as Algorithm 2.

We trained our DMDQN using small data set generated by individual controlling VR/AR device, which we named as pre-training. Then we applied the pre-trained model to control the robotic system to update the experience of robotic system, and the DMDQN is fine-tuned by the updated replay memory in return. Finally, the two processes continue repetitively until the distribution of the Q-network's hyper parameter changes slightly.

The histogram of action demonstrates that in replay memory the distribution of action space focuses on a few specific actions and some action did not appear during the entire replay memory because robotic system reached the object toward specific direction. So the replay memory is incomplete owing to the imbalance distribution of action histogram even the absence of several actions. According to our definition of action space, half of the actions is in the same line but opposite direction for example 0th and 26th, 14th and 12th, as Figure 5 shows. So we proposed action-overturn policy to balance the distribution of action. We swapped the current state s_t and the next state s_{t+1}, and changed the reward and action to $-r_t$ and $27 - a_t$.

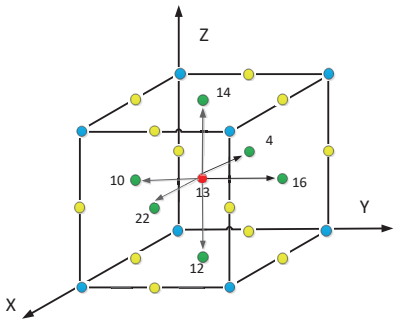

Figure 5. 10th and 16th, 22th and 4th, 12th and 14th are both in the same lines but opposite directions

In practice we sampled mini-batch, $\{s_t, s_{t+1}, a_{coor,t}, a_{rad,t}, r_{coor,t}, r_{rad,t}, t_t\}$ from replay memory uniformly at random and generated a random number which indicates whether using action-overturn. If the random number is greater than 0.5 we swap the sampled mini-batch to $\{s_{t+1}, s_t, 27 - a_{coor,t}, 27 - a_{rad,t}, -r_{coor,t}, -r_{rad,t}, t_t\}$. Figure 6 shows the distribution of action. After using the action-overturn policy, it can be seen that the distribution of action is balanced partically.

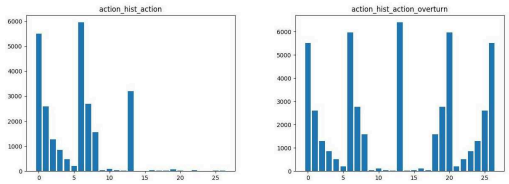

Figure 6. The left is the distribution without action-overturn. The right is generated by action-overturn

To make robotic system able to tackle task in an exploration way, we proposed a exploration policy combining the DMDQN and data augment, which trains the DMDQN while updating the replay memory. Every $test\_interval$ iterations of training, we applied the trained Q-network model of DMDQN to control the robotic system to explore environment to reach object while updating the replay memory. At each state the reward and terminate function is computed until terminate signal is received.

Our double-task DQN based on dueling DQN and double DQN with multiple views, using replay memory and data augment, updates θ_τ by stochastic gradient descent and the entire pseudocode of the exploration policy as Algorithm 2 shows.

Algorithm 2 double-task DQN with Multiple views using replay memory and data augment

Load dataset to replay memory
Initialize action-value function Q_{coor} and Q_{rad} with random hyper-parameter θ
Initialize target action value function $\hat{Q}_{coor}$ and $\hat{Q}_{rad}$ with $\theta' = \theta$
for episode $= 1, M$ **do**
  for train $= 1, T$ **do**
    Sample from replay memory using action-overturn and get $e_t = \{s_t, s_{t+1}, a_{coor,t}, a_{rad,t}, r_{coor,t}, r_{rad,t}, t_t\}$
    Select $a'_{coor,t+1} = \arg\max Q_{coor,q}(s_{t+1}, a_{coor}|\theta)$ and $a'_{rad,t+1} = \arg\max Q_{rad,q}(s_{t+1}, a_{rad}|\theta)$
    Set

$$y'_{coor,t} = \begin{cases} r_{coor,t} & \text{if } t_t = 1 \\ r_{coor,t} + \gamma Q_{coor,target}\cdot(s_{t+1}, a'_{coor,t+1}|\theta') & \text{otherwise} \end{cases}$$

    and

$$y'_{rad,t} = \begin{cases} r_{rad,t} & \text{if } t_t = 1 \\ r_{rad,t} + \gamma Q_{rad,target}\cdot(s_{t+1}, a'_{rad,t+1}|\theta') & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on $L_{coor}(\theta)$ and $L_{rad}(\theta)$ with respect to the network parameters θ
    Synthesis-loss function $\alpha L_coor(\theta) + (1-\alpha) L_{coor}(\theta)$ and update θ by stochastic gradient descent
    Every C steps reset $\theta' = \theta$
  end for
  Execute action-value function with current parameter θ
  for test iteration $j = 1, P$ **do**
    Initialize robotic system position
    while $t_i^j \neq 0$ **do**
      Get current state $s_i^j \in \{I_{1,i}^j, I_{2,i}^j, I_{3,i}^j, I_{4,i}^j\}$
      Select and execute $a_{coor,i}^j = \arg\max Q_{coor,q}(s_i^j, a_{coor}|\theta)$ and $a_{rad,i}^j = \arg\max Q_{rad,q}(s_i^j, a_{rad}|\theta)$
      Execute security detection function and get security coefficient C_i^j
      if $C_i^j \geq threshold\ value$ **then**
        Pass
      else
        Break
      end if
      Calculate $r_{coor,i}^j$, $r_{rad,i}^j$, t_i^j and store s_i^j, $a_{coor,i}^j$, $a_{rad,i}^j$ in the replay memory and disk
    end while
  end for
end for

Ensuring safe exploration poses a significant challenge for DMDQN and data augment, due to robotic system's working different environmental conditions. In our robotic system, we proposed a set of safety constraints and set an interface for special purpose. we set a maximum commanded velocity to allow per joint as well as strict position limits for each joint. In addition to joint limits, we used a bounding sphere for the end-effector position to avoid the end-effector to crash other joints. Our safety constraints are safe enough to ensure robotic system exploring steadily. However they are strict and restrict the motion space of end-effector. Much proper safety constraints should be studied further.

4. Experiments and Results

To evaluate the feasibility of the DQN-based system, DMDQN algorithm and data augment policy in learning target reaching, we did some experiments in both simulation and real-world scenario. In these experiments, our double tasks model was able to train large neural network using the DQN-based system and stochastic gradient in stable manner. And we compared our results with the best performing methods from the RL literature. The experiments consist of some parts as follows:

(1) Comparison of double tasks DQN with multiple views and single task DQN with multiple views during training and testing;
(2) Comparison of double tasks DQN with multiple views and double tasks DQN with single view in simulation and real world;
(3) Comparison of the model using the data augment and the model without data augment.

To ensure the experiment (1) and (2) working effectively, we used VR/AR device to control the robotic system to sample dataset which satisfies the DMDQN and DQN algorithm. We sampled 1500 sequences, each of which recorded the robotic system from initial position to terminate position. And every sequence of the dataset includes about 600 states of our robotic system. As mentioned above, each state contains four images from four different cameras. So we prepared about 4 million images which indicated 1million states about the robotic system for replay memory. Before being fed into the neural network, all of the images will be resized from 160×320 to 84×84 and added random noise on each image.

In our experiments, we used the $RMSPProp$ algorithm with mini-batch of size 32. And discount factor of $\gamma = 0.99$ and the loss coefficient of $\alpha = 0.6$ with the base learning rate of either 0.0001 or 0.001 were used for all the experiments. The behavior policy during auto-sampling was ϵ-greedy with ϵ annealed linearly from 1 to 0.1 over the first million frames, and fixed at 0.1 thereafter. We trained all of million states and used a replay memory of one million

most recent states.

4.1. double tasks vs single task

The content in this section is to compare double tasks DQN with multiple views and single task DQN with multiple views. We used the dataset to train our double-task model and single-task model respectively, then the trained models were applied to complete the reaching target. Figure 1 and Figure 7 show the architectures of double-task and single-task model respectively. In this experiment, the coordinate task is used to our single-task model.

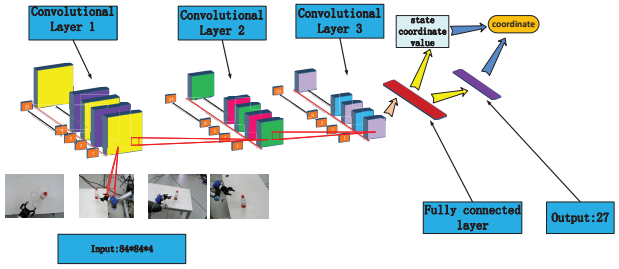

Figure 7. The single-task DQN model.

After training the double-task model and the single-task model respectively, we used the trained models to control our robotic system for thousands of episodes. Figure 8 shows the terminate states of one of testing episode. In double-task model, the coordinate and the pose angle of end-effector can be adjusted. However in single-task model, the reaching task is determined by it's initial pose angle which is fixed all the time. In our experiment, the model with multiple tasks reached object at 98.6% of the average rate of success compared with 74.2% using signal task

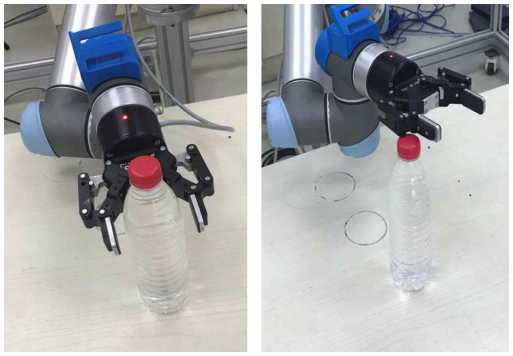

Figure 8. The left one used the double-task model and the right one used the single-task model

Our experiment proves that in the robotic system, we can design the double-task model according to the structure of system and the special tasks. We can define multiple action spaces to work together effectively. Meanwhile, the multi-task model can reduce the computation of the neural network. Figure 9 shows the multi-task more than two tasks. And the multi-task model is not only suit for the DQN-based algorithm but also for other reinforcement learning method.

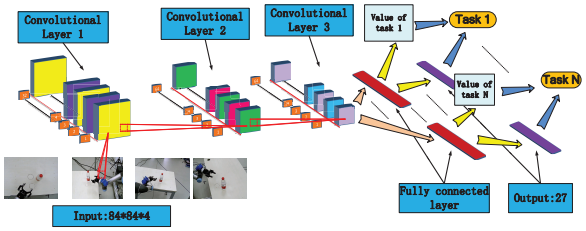

Figure 9. The architecture of the Multi-task DQN with multiple views.

4.2. multiple views vs single view

To compare the performance of multiple views and single view, we did several experiments. In this section, we used four observation angles together and the four observation angle respectively to feed the DMDQN. So there were five sets of experiment named as A, B, C, D, E.

Experiment A used the multi-view setting; Experiment B, C, D, E used the single-view settings respectively. The observation angles of B, C, D, E are $(45°, 90°, 135°)$, $(90°, 90°, 180°)$, $(135°, 90°, 135°)$, $(90°, 45°, 135°)$ respectively. Figure 10 shows the image used to train Experiment B, C, D, E is from the left to the right respectively. And Experiment A used all of them as training data.

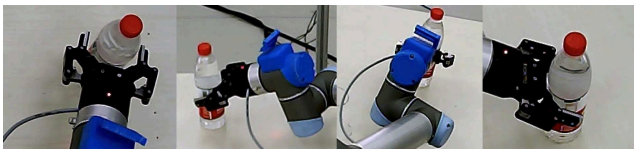

Figure 10. The images from left to right were used by Experiment B, C, D, E respectively

We used the corresponding images to train the DMDQN respectively and used the trained policy to control robotic system to complete the reaching target. We trained the five models for 7 days using Tesla GPU. Table 1 shows the results of five experiments. After training these models, we used them to control robotic system from initial position to terminate position for 10000 times. As the result implies, Experiment A achieves the highest score, 98.6%, in five experiments.Compared with Experiment A, Experiment C gets score 97.5%, competitive with Experiment A. Experiment B and D have similar scores, 85.4% and 86.8% due to symmetrical views. While Experiment E has obvious difference to the others because the data used to train the neural network is the third image in Figure 10, not containing the end-effector and losing most of information. These sets of experiments indicate that the view angle right above robot

acquire most of state information, and other three view angle provide auxiliary information.

Table 1. The average rate of success of five experiments.

Experiment ID	View angle	The rate of success
A	All of the following angles	98.6%
B	(45°,90°,135°)	85.4%
C	(90°,90°,180°)	97.5%
D	(135°,90°,135°)	86.8%
E	(90°,45°, 135°)	47.3%

4.3. data augment vs without data augment

As mentioned above, the data augment policy including action-overturn and auto-sampling is equipped with our algorithm system. In the deep reinforcement learning, the replay memory is updated by exploring instead of sampling artificially. To consider this the data augment policy is developed.

In this section, we analyze our experiment below to explain the effectiveness of data augment, especially the action-overturn. In this experiment, we trained the DMDQN using the action-overturn mentioned above and without it respectively until the loss function changes slightly. The two trained models controlled robotic system to reach the object for thousands of episodes and we analyze the distribution of the coordinate-action and radian-action. As Figure 11 shows, the distribution of action can reflect the dataset in some way during exploring.The action-overturn can remedy the non-uniform distribution of the replay memory partially. Furthermore, the model trained using the action-overturn can generate the terminate signal in appropriate position effectively and the other model control the robotic system to unsafe state as Figure 12 shows. Due to the action-overturn policy, the model is able to determine whether the current state is the terminate state.

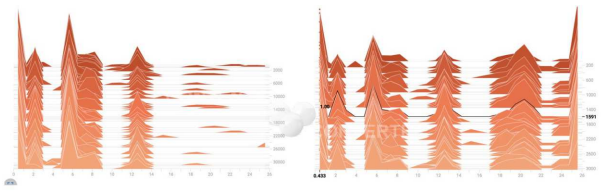

Figure 11. The left and the right is the distribution of using the action-overturn and without action-overturn respectively

During testing, we also noticed even equipped with data augment, the exploration policy controlled the robot to an unsafe position. The safety constraints mentioned above endow the system able to detect potential danger and initialize the position. However the imperfect safety constraints limit the space of exploration of the robotic system, thus

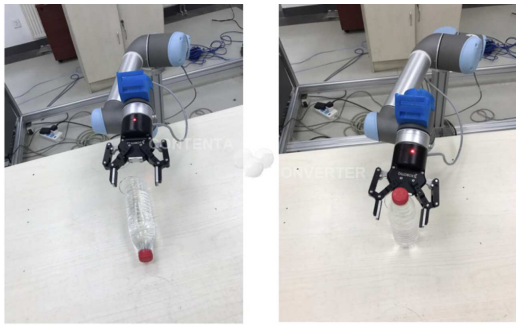

Figure 12. The left state was generated without action-overturn. The right is obtained using it

the safety constraints will be further studied. In our contrast experiment, 30% of test episode couldn't generate terminal signal and exposed robot in an unsafe state without data augment and the score would decrease to under 5% using data augment in an exploration way.

5. Conclusion and Future Research

In this paper, we pose an algorithm system which includes the definition of action space, double-task Deep Q-Network with multiple views, data augment policy and safety constrains. The definition of action spaces can be applied to complicated robotic systems, which can reduce the computation of algorithm. The double-task Deep Q-Network with multiple views performs better than current control policy, [8, 2, 18], all of which used the single-task Deep Q-Network with single view. The approach of double-task model can be extended to multi-task model, which will be used to handle more sophisticated missions of robotic system for further research. Compared with those single-view policies, multi-view policy can record more information of each state of robotic system but the disadvantage is that the observation angles must be fixed and the dataset can't be general. In further research, we would like to equip cameras to robotic system instead of environment. Data augment containing action-overturn and auto-sampling can update the replay memory effectively. The action-overturn policy can balance the distribution of actions and decrease required RAM and generate the terminate signal according to the coordinate and radian information. The auto-sampling can sample data only on the base of small dataset. In practice we combined the DMDQN with data augment to form training policy which can update the replay memory while training, as Algorithm 2 describes. The safety constrains can ensure the training policy working steadily and protect robotic system, but the it is not perfect and restrict possible positions of robotic system. The safety constrains will be developed in future work.

References

[1] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng. Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot. *International Journal of Robotics Research*, 27(2):213–228, 2008.

[2] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *arXiv preprint arXiv:1610.00633*, 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. pages 770–778, 2015.

[4] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal. Learning force control policies for compliant manipulation. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4639–4644. IEEE, 2011.

[5] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[6] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA*, pages 2619–2624 Vol.3, 2004.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37, 2016.

[10] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional models for semantic segmentation. In *CVPR*, volume 3, page 4, 2015.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[12] J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. In *AAAI*, pages 1607–1612. Atlanta, 2010.

[13] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[14] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[15] E. Theodorou, J. Buchli, and S. Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2397–2403. IEEE, 2010.

[16] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[17] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. pages 1995–2003, 2015.

[18] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke. Towards vision-based deep reinforcement learning for robotic motion control. *Computer Science*, 2015.