UDC 004.896
doi:10.31799/1684-8853-2020-5-24-32

Articles

Controlling system based on neural networks with reinforcement learning for robotic manipulator

E. B. Solovyeva[a], *Dr. Sc., Tech., Associate Professor, orcid.org/0000-0001-8204-6632, selenab@hotbox.ru*
A. Abdullah[a], *Post-Graduate Student, orcid.org/0000-0002-4024-9201*
[a]*Saint-Petersburg Electrotechnical University «LETI», 5, Prof. Popov St., 197376, Saint-Petersburg, Russian Federation*

Introduction: *Due to its advantages, such as high flexibility and the ability to move heavy pieces with high torques and forces, the robotic arm, also named manipulator robot, is the most used industrial robot.* ***Purpose:*** *We improve the controlling quality of a manipulator robot with seven degrees of freedom in the V-REP program's environment using the reinforcement learning method based on deep neural networks.* ***Methods:*** *Estimate the action signal's policy by building a numerical algorithm using deep neural networks. The action-network sends the action's signal to the robotic manipulator, and the critic-network performs a numerical function approximation to calculate the value function (Q-value).* ***Results:*** *We create a model of the robot and the environment using the reinforcement-learning library in MATLAB and connecting the output signals (the action's signal) to a simulated robot in V-REP program. Train the robot to reach an object in its workspace after interacting with the environment and calculating the reward of such interaction. The model of the observations was done using three vision sensors. Based on the proposed deep learning method, a model of an agent representing the robotic manipulator was built using four layers neural network for the actor with four layers neural network for the critic. The agent's model representing the robotic manipulator was trained for several hours until the robot started to reach the object in its workspace in an acceptable way. The main advantage over supervised learning control is allowing our robot to perform actions and train at the same moment, giving the robot the ability to reach an object in its workspace in a continuous space action.* ***Practical relevance:*** *The results obtained are used to control the behavior of the movement of the manipulator without the need to construct kinematic models, which reduce the mathematical complexity of the calculation and provide a universal solution.*

Keywords *— kinematic control, reinforcement learning, deep learning, robotic manipulator, deep neural network, deep deterministic policy gradient.*

For citation: Solovyeva E. B., Abdullah A. Controlling system based on neural networks with reinforcement learning for robotic manipulator. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2020, no. 5, pp. 24–32. doi:10.31799/1684-8853-2020-5-24-32

Introduction

The use of deep neural networks has been rising lately in solving many different technical problems in many fields, especially in robotics and automation control, where the aim is to build intelligent systems that can operate without the need of human experts. The progress of unsupervised deep learning has continued to rise in rank, aiming to learn intelligent behavior in complex and dynamic environments [1, 2]. Therefore, we will review the methods of controlling a robotic manipulator using kinematic control and machine learning control in the field of reinforcement learning and examine the problems that can face us in this procedure. Learning control policies in different systems' operations have made reinforcement learning an optimal solution where it fits well with various tasks. In addition, recent work in this field has progressed towards capturing the state of the environment through images, something developers are still trying to achieve but in a different context.

Reducing continuous spaces of action would be poorly scaled, as the number of discrete actions increases exponentially with the action dimensionality, so deep reinforcement learning operates with continuous spaces of action to be convenient with real-world control problems. Furthermore, having a parameterized policy can be beneficial, since it can generalize in the space of action [3].

Reinforcement learning is one of the most popular areas of machine learning [4–6]. Stimulated by human behavior, it allows an agent (the learner and decision-maker) to discover optimal performance by experiment and failure interactions with its enclosing environment to solve the problems of control. The environment is everything outside of the agent that can be associated with, while a learning task is the complete specification of the environment. The first method in the field of reinforcement learning is dynamic programming [6]. Dynamic programming uses value functions to structure the search for good policies but needs a perfect environment model. The need to know the complete model of the environment limits the dynamic programming method, wherein many control problems, knowing the whole aspects of the environment, could be impractical because many problems give rise to huge state sets. The next method is Monte Carlo (MC) one [6]. The model-free MC method enables us to learn from the

environment's sample sequences of states, actions, and rewards. MC method works by equating the sample returns from the environment on an episode by episode base, so we do not require the full model of the environment like in the case of dynamic programming, but in MC method the agent must consider the exploration-exploitation tradeoff in order to get information about the rewards and the environment. It needs to explore by analyzing both previously unused actions and uncertain actions that could lead to negative rewards. A new method of reinforcement learning evolves from both dynamic programming, and MC method. This method is called temporal-difference (TD) learning [6]. TD learning combines dynamic programming's ability to learn through bootstrapping and Monte Carlo's ability to learn directly from examples selected from the environment without access to the Markov decision process. Alternatively, TD method only waits until the next time-step by using temporary errors to notify us how different the new value is from the old prediction. Temporal difference learning led us to a method called state-action-reward-state-action (SARSA) [4–6]. SARSA is an on-policy TD control algorithm. This name is derived from an experience in which the agent starts in certain state performs an action, receives a reward, then transfers to a new state, and decides to do a new action. Based on SARSA, *Q*-learning method appeared but conversely to SARSA. *Q*-learning is an off-policy TD control algorithm, which directly approximates the expected reward independent of the policy being followed [4–6].

The main disadvantage of all of the above-mentioned methods is the need for a vast database of samples and their need to train them before we start to perform the process of control. On the other hand, we need an algorithm that can be suitable for continuous spaces, where the algorithm can learn and perform simultaneously with sensor reading and action executions. Deep deterministic policy gradient or deep *Q*-learning was developed to solve this problem and to overcome all the mentioned methods' disadvantages by developing the architecture of the agent to consist of a critic deep neural network and an actor deep neural network. The critic and the actor can work in parallel to give us the actions and the estimated rewards for these actions while the training process continues.

Hence, we study deep reinforcement learning algorithm, namely, deep deterministic policy gradient [3]. In order for robots to achieve a common advantage purpose, reaching objects is a fundamental ability to learn. Traditionally, human experts are required to analytically produce an algorithm for a particular task under adaptive control using kinematic control and supervised learning control, but this is a challenging and time-consuming approach. By applying deep learning, we overcome these restrictions in generalizing robotic control and demonstrating how building the actor-network and the critic-network based on convolution neural networks (CNN) increases the quality of the performance compared with fully connected neural networks. We begin by showing the area of interest, machine learning, focusing on deep learning and reinforcement learning, and deep deterministic policy gradient; we describe how to control a robotic arm using deep learning.

Kinematics control of a robotic manipulator

Given the joint rotation angles and the lengths of the manipulator's frames, we represent the forward kinematics [3]

$$\mathbf{X} = F(\mathbf{q}, \theta_{kin}), \quad (1)$$

where $\mathbf{X}$ is the coordinates vector containing the position and orientation of the robot's end effector; $\mathbf{q}$ is a matrix of joints' angels; θ_{kin} is a vector of fixed kinematic linking parameters. It consists of parameters describing the robotic manipulator lengths and angles, illustrating each joint axis's rotation relative to the previous joint axis.

Closed-form solutions of (1) are favored. However, there are manipulator structures, for which only iterative numerical solutions are possible. From equation (1) we get the equation of the inverse kinematics given as

$$\mathbf{q} = F^{-1}(\mathbf{X}, \mathbf{C}, \theta_{kin}), \quad (2)$$

where $\mathbf{C}$ is a vector containing some information used to select a possible solution, and another alternative is to let $\mathbf{C}$ be the previous solution and choose the new solution as the closest solution.

When the robot's end effector is in a fixed position, there will always be existing values for the joints angles which led the end effector to be in such position and direction, so a closed-form solution for the forward kinematics problem is always assured in comparing with the inverse kinematic making it more comfortable to deal. This solution defines the workspace of a manipulator. On the other hand, there can even exist an infinite number of solutions, like the case of a redundant manipulator [3].

When the dimension of the task-space is smaller than the dimension of the joint space, the kinematic structure is considered redundant. Here the interest is in the inverse problem because we can calculate the angels of the joints that will lead to reaching a point in their workspace. As we can see, the kinematic control needs to know the robot's parameters and environment, which makes it a non-uni-

versal solution for the controlling problem. Each robot needs to rebuild the complete mathematical model and recalculate all the inverse and forward kinematics metrics, which will take a lot of processing each time. The solution for finding a universal solution for the controlling process is by using unsupervised learning.

Advantages of reinforcement learning in controlling a robotic manipulator

In unsupervised learning, we get a lower complexity compared to supervised learning because we are not expected to understand and then mark the input data. This situation happens in real-time so that all the input data must be analyzed and marked, which helps us understand the various training models and sorting of raw data [4–6]. It is easier for us to get unmarked data from a computer than marked data because marked data demands human interface and understanding of the categorization of such data to use in the learning process. Experts should estimate the target output or part of it in order to achieve the learning process. The supervised learning makes this way time-consuming and not flexible for various systems where some changes to the environment or the robot could happen. Nevertheless, there are many systems where we can not estimate the output, and we can not have enough information to build the target output to achieve the learning process.

In supervised learning, we would have a set of coordination of some locations in the workspace and the corresponding angles of the manipulator's joints. We can then feed those input frames through a neural network that, at the output, can produce the angles of the motors or joints by training on the data set from the previous data of the locations and the corresponding angles. Many approaches could be used, like backpropagation, so we can train that neural network to replicate human manual control actions. However, when we want to do supervised learning, we have to create a data set to train [6–8]. On which is not always a straightforward thing to do, and on the other hand, if we train our neural network model to imitate the actions of the human control well only, then by definition, our agent can never be better at executing the right action. When we want to train a neural network to perform by itself a controlling process on a robotics manipulator, where this controlling will take place in different environments, it could face many new problems. On the other hand, the offered method of unsupervised learning achieves this goal [9, 10].

However, the only difference here is that now we do not know the target label. Therefore, we do not know the rotation of the manipulator's joints in any situation because we do not have a data set to train on, and in reinforcement learning, the network that transforms input frames to output actions is called the policy network [11]. The approach in policy gradients is that we start with a completely random network, we feed that network the coordinates from the environment, and it produces random action. Send that action back to the joints motors and then produces the next frame and this is how the loop continues and the network, in this case, it could be fully connected networks, but we can apply convolution network, in other words, deep neural networks and from this, we get the name "Deep learning". Allowing our agent to randomly explore the environment and discover better rewards and better behavior [12].

Within the task, the learning process is divided into episodes. We usually use each episode as a control for a finite steps model, where it is essential to operate in many steps until we reach the time where we reset and restart.

When we move from a step to another under making an action, the agent receives a reward describing the effectiveness of this step regarding the task. The objective here is to maximize this collective reward during the learning process for the robot [13]. The equation of the collective reward R is given as

$$R = r_1 + r_2 + r_3 + \cdots + r_{K+1} = \sum_{k=0}^{K} r_{k+1}, \qquad (3)$$

were r_1 is a reward for moving from the state 0 to the state 1 in the step 1; r_2 is a reward for moving from the state 1 to the state 2 in the step 2; $(K + 1)$ is the number of rewards or steps, which refers to the end of the episode; k is a counter for the sum sign; $(k + 1)$ is a step number.

As we can see from equation (3), the collective reward could increase without any conversion, making it not suitable for all the tasks [14, 15]. Therefore, we define a factor $0 \le \gamma \le 1$ that assures the conversion and determines the value of the future rewards that the robot might receive. By adding this discount factor to equation (3) we get the expected collective reward as

$$R = r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^K r_{K+1} = \sum_{k=0}^{K} \gamma^k r_{k+1}, \qquad (4)$$

where γ is the discount factor.

We must take extreme caution when choosing an appropriate value of γ in equation (4) because it can often change the form of the optimal solution where different values lead to different performances [16]. If γ is small, the agent could select cases that only increase the reward and lead to lower performance in the long-term. In contrast, if γ is big, the

agent will lose the capacity to differentiate between the policies that will get a reward in the future and those directly get a big reward [17, 18].

Controlling a robotic manipulator using deep learning

Deep learning is a field of reinforcement unsupervised learning involved with deep neural networks. We used two types of deep neural networks each time, the convolution neural network and the fully connected neural network. In convolution neural networks, the connections are between the neuron and its surrounding neurons from the previous layer, and these connections share the same weights and bias [19–21].

Let's consider the method of deep deterministic policy gradient with using deep neural networks as an actor and critic. Set up the following notations and variables: $\pi(\mathbf{s}_j|\xi)$ is the actor-network output calculated on the basis of the actor neural network having the input vector $\mathbf{s}_j$ and the parameters vector ξ; j is the current step; $\mathbf{s}_j$ is the state vector built by the observation in step j; ξ is the parameters vector of the actor-network; $Q(\mathbf{s}_{j+1}, \pi(\mathbf{s}_{j+1}|\xi)|\theta)$ is the output of the critic network having the input vector $\mathbf{s}_{j+1}$, which is obtained from state $\mathbf{s}_j$ after action $\mathbf{a}_j$ of the actor-network, and the parameters vector θ. On the basis of the TD learning, the updated parameters vector θ of the critic-network results from solving the optimization problem [6, 9]

$$L = \frac{1}{M}\sum_{j=1}^{M}\left(\mathbf{y}_j - Q(\mathbf{s}_j, \mathbf{a}_j|\theta)\right)^2 \to \min_{\theta}, \qquad (5)$$

where L is the loss function; M is the maximum count in the last step; $\mathbf{y}_j$ is the value function target, output vector of deep neural network, which gives the critic signal or the joint angels; $\mathbf{a}_j$ is the action vector in the given state $\mathbf{s}_j$. This output vector, similar to the solution of the inverse kinematics equation (2), is described as

$$\mathbf{y}_j = r_j + \gamma Q\left(\mathbf{s}_{j+1}, \pi\left(\mathbf{s}_{j+1}|\xi\right)|\theta\right), \qquad (6)$$

where r_j is the reward in step j; j is the current step.

We train the network to decrease the mean squared error concerning the Q function [22, 23]. However, the dependence of the Q targets on Q itself can lead to instabilities or even divergence during learning. We consider a policy that can be described by parameters very beneficial for control because it allows for learning when the sensory reading and actions executions belong to continuous spaces. The target value function $\mathbf{y}_j$ in (6) is proposed to be a permanent value on learning a neural network by the back-propagation algorithm.

After updating the parameters vector θ of the critic-network, to updat the parameters vector ξ of the actor-network we use the following policy gradient when maximizing the expected discounted reward (4) [6, 9]:

$$\nabla_{\xi} J \approx \frac{1}{M}\sum_{j=1}^{M}\mathbf{G}_{\pi j}\mathbf{G}_{\xi j}, \qquad (7)$$

$$\mathbf{G}_{\pi j} = \nabla_{\pi} Q(\mathbf{s}_j, \pi(\mathbf{s}_j|\xi)|\theta),$$

$$\mathbf{G}_{\xi j} = \nabla_{\xi}\pi(\mathbf{s}_j|\xi),$$

where $\nabla_{\xi} J$ is the policy gradient; $\mathbf{G}_{\pi j}$, $\mathbf{G}_{\xi j}$ are the gradient vectors of the critic's and actor's outputs with respect to an actions and parameters of the actor-network respectively; ∇_{π} is the gradient ascent with respect to the policy of the action; ∇_{ξ} is the gradient ascent with respect to the parameters vector ξ of the actor-network.

From equation (7), the resulting policy gradient increases the expected discount reward, and we use it to update the actor-network weights and bias. On the other hand, from equation (5), after minimizing the loss function over all the experiences, we use it to update the critic-network weights and biases. In consideration of the observation, we evaluate the gradient of the critic-network output and the gradient of the output of the actor-network. We perform a smoothing process to update the weights and the biases of the target actor-network and the critic-network:

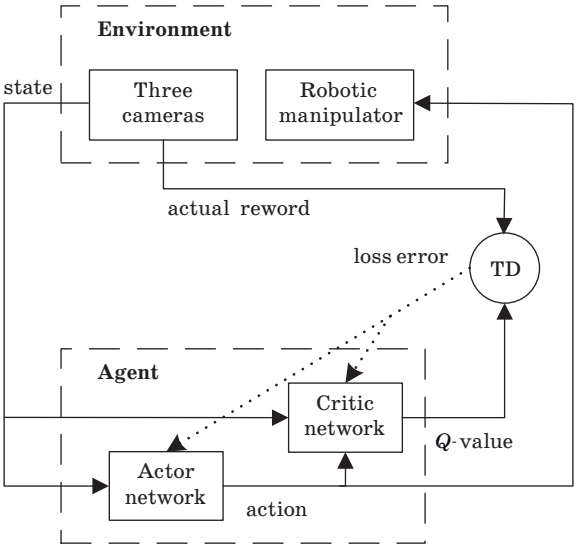
$$\bar{\theta} = \tau\theta + (1-\tau)\bar{\theta}, \quad \bar{\xi} = \tau\xi + (1-\tau)\bar{\xi},$$

where τ is a smoothing factor equal to less than one; $\bar{\theta}$, $\bar{\xi}$ are the updated parametes vectors of the critic and actor networks correspondently.

Finally, we repeat performing equation (6) after we get the new observation for a new step in the training until the end of the episode and the beginning of new training episode.

The practical experiment of controlling the robot manipulator

As illustrated in Fig. 1, the learning process starts with taking the coordinates of the end effector and the cube on the three-axis from the cameras, generating the state vector. We send the state vector to the actor-network. The critic network takes both the action generated by the actor-network and the state and gives us the expected reward from this action or the Q-value. In the next steps to improve the performance, when we are in a state, and the actor is proposing a particular action, we take a slightly different action and see if the Q-value a lit-



■ ***Fig. 1***. The block-scheme of the deep deterministic policy gradient used to control the robotic manipulator

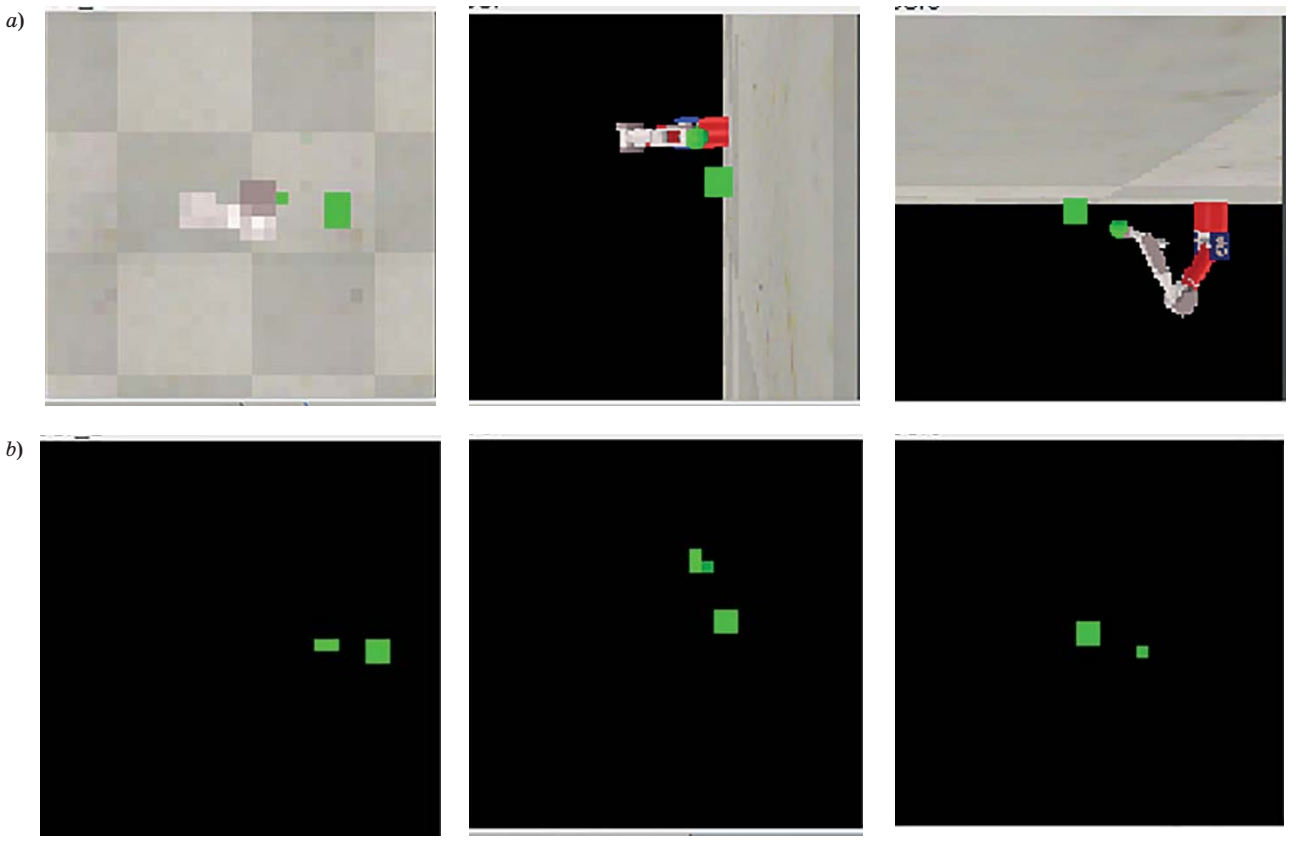
tle higher we change the action to the new proposed action. The robotic manipulator moves when it gets the action signal. The cameras make the observations, generating a new state vector describing the new situation of the environment. On each axis, if the distance between the end effector and the cube gets smaller than before, we add a positive reward.

On the contrary, if the distance gets bigger, then before we add a negative reward. We sum the rewards to get the actual reward of the current action. We calculate the loss by achieving a TD between the actual reward and the estimated reward generated by the critic (the Q-value). We use the loss in computing the gradient of the Q-value with respect to the action. We back propagate the gradient of the Q-value with respect to the action to train and update the neural networks and to evolve our actor in the right direction.

The simulation environment is built using V-REP program. A robot arm model with seven degrees of freedom (7 DoF manipulator) is used. In addition, three cameras are added in different places to use them in determining the state and the value of the reward function. A linkage library called Remote API is used to connect V-REP program with MATLAB language. This library provides a way to create a connection between the programming language and the simulation program, so it becomes possible to take pictures from the cameras and process them using MATLAB image processing functions. The connection mechanism also allows sending motion commands to the motors on the robot joints, which makes the simulation environment so close to the real implementation environment.

We want to enable our agent to learn entirely by itself. The only feedback that was going to give it is the distance between its end effector and the object we want the manipulator to reach. Whenever our agent manages to make this distance smaller, it will receive a positive reward. If the gap gets bigger than before, then our agent will receive a penalty of negative reward. The target of the agent is to optimize its policy to earn as much reward as possible. To train our policy network, we will collect a bunch of experiences by selecting random actions to feed them back into the actor and create a whole bunch of random movements in the environment. Since our agent has not learned anything useful yet, it is going to make arbitrary, not accurate movements. Sometimes our agent might get lucky while it is going to random select of action. In this case, when a sequence minimizes the distance, our agent will receive a reward. We should note that every episode, regardless of whether we want a positive or a negative reward, we can compute the gradients that would make the actions that our agent has chosen more likely in the future. Therefore, what policy gradients are going to do is that for every episode where we have a positive reward, we will use the normal gradients to increase the probability of those actions in the future. Whenever we got a negative one, it is going to apply the same gradient, but we are going to multiply it with minus one, and this minus sign will make sure that in the future, all the actions that we took in a bad episode are going to be less likely. The result is that while training our policy network, the actions that lead to negative rewards are slowly going to be reduced, and the actions that lead to positive rewards will increase.

To make observations of the state for the agent of the reinforcement learning (the robotic manipulator), we have used three web cameras that take photos of the manipulator workspace from three perspectives so we can analyses a 3D vision for the environment on the three axes. Because of the limitation of the hardware, we do not have the ability to give the images as input to the neural networks, so we performed image processing using MATLAB to get the coordinates of both the robot and the end effector. We added as well the angles of rotation for each joint to form the state vector (the observations). Thus, using simulated webcams with 128×128 resolution, the environment was observed. Before each action, a picture is taken from each cam. Then using image processing methods of segmentation and detection in MATLAB system, we get the coordinates of the end effector and the object we want to reach to form our state. The original photos from three vision sensors in the environment of V-REP with a resolution of 128×128 showing the robotic arm as a whole and the box we want to reach are depicted in Fig. 2, *a*. The images obtained after the photos processing in MATLAB and showing two green pints,



■ ***Fig. 2.*** The original photos from three vision sensors in the environment of V-REP with a resolution of 128 × 128 to simulate real life web cameras (*a*) and the images obtained after the photos processing (*b*)

which are the end effector of the robot and the box, we aim to reach, are depicted in Fig. 2, *b*.

Let us consider the simulation environment. The robotic manipulator has seven motors; each motor is related to two adjacent digits, respectively. The motor rotates in a clockwise direction if the first digit is bigger than the second one and vice versa. Thus, the number of the action signal parameters is 14. After that, we start taking pictures from the cameras and discovering the green areas in them, because the end effector and the element we want to reach are made in green. Then, we determine the coordinates of these two elements in the three images, as seen in Fig. 2, *b*. The state contains seven values that reflect the motors' angles, and 12 values reflect a pair of coordinates in each of the three images. The total number of the state signal parameters is 19.

Two kinds of neural networks are chosen to build the actor and the critic networks. In the first case, the actor and critic networks are designed as CNN and, in the second case, as the fully connected neural networks. The actor-network includes four layers and has a size of 200 × 200 × 38 × 14, where every number means the number of neurons in a layer. The actor-network has 14 output signals to give the rotation of the joints with two pairs for each of the seven joints. The critic-network comprises four layers and has a size of 200 × 200 × 10 × 1. The output signal is a value of the Q-function.

The training process of the robot consists of a maximum number of 600 episodes when using convolution neural networks and 1000 when using the fully connected networks, where the episode is all the actions, and the states that come in between an initial-state and a terminal-state and each taken action and state is considered to be one step inside the episode. The episode consists of a maximum number of 600 steps when using the convolution neural networks and 1000 when using the fully connected networks. Each episode ends when the robot achieves the task of reaching the cube or when it reaches its maximum number of steps without being able to reach the cube. After that, a new episode begins with a new initial position for the cube in the workspace of the robot. As a start of the training process during the first step in the first episode, the state vector is constructed by taking the coordinates of the end effector of the robot and the cube (the object we aim to reach) from three images in the initial observations. We send the state vector to the actor-network as its input. The actor-network generates the first action as an output, and we send it to the motors' joints to perform the first movement.

We take the new observations constructing the new state vector, and we send it to the critic-network with the action vector. The critic-network takes the state vector, and the action vector as an input and generates the *Q*-value is output. At the end of this step, the reward of the taken action is demonstrated by calculating the distance in the three images. Depending on the *Q*-value and the reward, we update the parameters of the actor-network and the critic-network entering the next step. In the next step, the actor-network gives the new action signal for the join's motor to rotate depending on the current state vector, and the new observations are taken constructing the new state vector. The new state vector is sent with the action vector to the critic-network repeating the same process. If the taken action in the step leads the robot to move far from the cube, the reward takes a negative value in order to reduce the repetition of such actions, and if the action in the current step reduces the distance moving the robot towards the cube, the rewards take a positive value to ensure repeating such actions. The task is considered to be accomplished when the distance is reduced to a certain level (very small distance) in the three images at the same time. The episode ends, and we give the reward a big positive value, and the robot returns to its initial position, a new position is determined randomly for the cube (the element we want to reach). A new episode starts, and a new action is taken from the output of the actor-network repeating the same process in the previous episodes. The training stops in the final episode, where the parameters of the actor-network always lead the robot to execute the actions, those making it reaching the cube.

In our experiment, we achieved two training process first using convolution neural networks to build the actor and critic networks. The results are represented in Table 1. Then, we repeat the training using the fully connected neural networks to build the actor and critic networks. The results are noted in Table 2.

In table 1 is clear that, the larger the network's size, the higher the learning parameters, the higher the average reward, which leads the robot to execute more likely good actions to reach the cube and to reaching high accuracy and lower elapsed time quickly.

As follows from the analysis of tables 1 and 2, the architecture of the convolution neural networks gives the character to be more specialized and efficient than the fully connected networks. In the architecture of the fully connected neural networks, there are connections between all the neurons in previous layers with each neuron in the next layer, with a unique weight to each connection. This connection pattern increases the network parameters and makes no assumptions about the data's features, increasing the expenses of the memory and the computation. On the other hand, in the convolution neural network's architecture due to its convolutional layers, the connections are between the neuron and its surrounding neurons from the previous and the next layer, and these connections share the same weights. This connection pattern decreases the number of the network parameters affecting its memory use (less memory use), the computation (less time), and increases the accuracy by producing more likely actions that increase the average reward.

■ ***Table 1***. The number of parameters, average reward and elapsed time on learning the actor and critic networks in the form of convolution neural networks

Sizes of actor and critic-networks	Number of parameters	Average reward	Elapsed time, h
200 × 200 × 38 × 14 200 × 200 × 10 × 1	3040 412	100.02	5.10
190 × 190 × 38 × 14 190 × 190 × 10 × 1	2890 392	99.23	5.24
180 × 180 × 38 × 14 180 × 180 × 10 × 1	2740 372	98.01	5.37

■ ***Table 2***. The number of parameters, average reward and elapsed time on learning the actor and critic networks in the form of fully connected neural networks

Sizes of actor and critic-networks	Number of parameters	Average reward	Elapsed time, h
200 × 200 × 38 × 14 200 × 200 × 10 × 1	48 384 42 221	98.01	5.21
190 × 190 × 38 × 14 190 × 190 × 10 × 1	44 094 38 211	94.21	5.40
180 × 180 × 38 × 14 180 × 180 × 10 × 1	40 004 34 401	90.46	5.59

We trained the networks using back propagation of the loss error by taking the gradient of the *Q*-value with respect to the input. However, the training process time depends on many factors; for example, each neural network is initialized with random values of biases and weights, giving different starting points per simulation during the training process. Due to its architecture, fewer connections and weights make convolutional layers relatively cheap in memory and computation (less time). In other words, CNN has a lower number of parameters, making it quicker to achieve the training target.

Convolution neural networks have a connection pattern that increases its accuracy in comparison

with the fully connected network because this pattern provides the characteristic of feature extraction allowing the data to be represented as spatial with the locally and equally possible to occur extracted features at any input. This feature extraction quality produces a lower rate of decreasing the average reward when reducing the parameters of CNN over reducing the parameters of the fully connected networks, which lack this property of the feature extraction.

Conclusion

In robotic control, we trained the robotic arm, using a typical robot with seven joints to move in high action space, using deep reinforcement learning algorithms and deep deterministic policy gradients. These methods have the advantage of allowing our robot to perform actions and train at the same moment. Therefore, we used them to control our robotic manipulator to reach a cube in its workspace; since these methods give the robot the ability to perform in continuous space action (the sensory reading and actions executions belong to continuous spaces).

The results of the investigation show that using convolution neural networks for designing the actor and the critic-networks has the following advantages over the fully connected networks:

— higher accuracy by driving the actor-network to perform more likely actions that led the robot to increase the average reward during the training;

— the robustness over decreasing the network parameters due to the feature extraction property, where decreasing the network's parameters, did not affect the convolution neural network in the same way it affects the fully connected networks.

— a quicker performance using less computation power and less memory than the fully connected neural networks.

The obtained results expand the possibilities of reinforced learning used in control systems, in particular when controlling a robotic manipulator, by combining the method of deep deterministic policy gradient with deep neural networks. An essential aspect of the research results is the demonstrated effectiveness of using cellular neural networks for reinforcement learning on controlling a robotic manipulator. This fact is important because cellular neural networks are very popular and they are part of many different structures of deep neural networks.

References

1. **Beysolow T. II** *Introduction to deep learning using R. A step-by-step guide to learning and implementing deep learning models using R*. Berkeley, Apress, 2017. 227 p. doi:10.1007/978-1-4842-2734-3
2. **Ketkar N.** *Deep learning with Python*. Berkeley, Apress, 2017. 226 p. doi:10.1007/978-1-4842-2766-4
3. **Polydoros A. S., Nalpantidis L., Kruger V.** Real-time deep learning of robotic manipulator inverse dynamics. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, September 28–October 2, 2015, pp. 3442–3448. doi:10.1109/iros.2015.7353857
4. **LeCunn Y., Bengio Y., Hinton G.** Deep learning. *Nature*, 2015, vol. 521, no. 5, pp. 436–444. doi:10.1038/nature14539
5. **El-Amir H., Hamdy M.** *Deep learning pipeline*. Berkeley, Apress, 2019. 551 p. doi:10.1007/978-1-4842-5349-6
6. **Lapan M.** *Deep reinforcement learning hands-on*. Birmingham, Packt Publishing, 2020. 800 p.
7. **Solovyeva E.** Behavioural nonlinear system models specified by various types of neural networks. *Journal of Physics: Conference Series*, 2018, vol. 1015, no. 3, 032139, pp. 1–6. doi:10.1088/1742-6596/1015/3/032139
8. **Solovyeva E.** Recurrent neural networks as approximators of nonlinear filters operators. *Journal of Physics: Conference Series*, 2018, vol. 1141, no. 1, 012115, pp. 1–10. doi:10.1088/1742-6596/1141/1/012115
9. **Brown B., Zai A.** *Deep reinforcement learning in action*. New York, Manning Publications Co., 2020. 360 p.
10. **Sejnowski T. J.** *The deep learning revolution*. Cambridge, The MIT press, 2018. 352 p. doi:10.7551/mitpress/11474.001.0001
11. **Angelova A., Carneiro G., Sünderhauf N., Leitne J.** Special issue on deep learning for robotic vision. *International Journal of Computer Vision*, 2020, vol. 128, pp. 1160–1161. doi:10.1007/s11263-020-01324-z
12. **Gupta A., Eppner C., Levine S., Abbeel P.** Learning dexterous manipulation for a soft robotic hand from human demonstrations. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea, October 9–14, 2016, pp. 3786–3793. doi:10.1109/iros.2016.7759557
13. **Rajeswaran A., Kumar V., Gupta A., Vezzani G., Schulman J., Todorov E., Levine S.** Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *14th Robotics: Science and Systems XIV*, Pittsburg, USA, June 26–30, 2018, pp. 1–9. doi:10.15607/rss.2018.xiv.049
14. **Pervez A., Mao Y., Lee D.** Learning deep movement primitives using convolutional neural networks. *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Birmingham, UK, November 15–17, 2017, pp. 191–197. doi:10.1109/humanoids.2017.8246874

15. **Widmaier F., Kappler D., Schaal S., Bohg J.** Robot arm pose estimation by pixel-wise regression of joint angles. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 16–17, 2016, pp. 616–623. doi:10.1109/icra.2016.7487185
16. **Shi J., Xu J., Yao Y., Xu B.** Concept learning through deep reinforcement learning with memory-augmented neural networks. *Neural Networks*, 2019, vol. 110, pp. 47–54. doi:10.1016/j.neunet.2018.10.018
17. **Rahmatizade R., Abolghasemi P., Boloni L., Levine S.** Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 21–25, 2018, pp. 3758–3765. doi:10.1109/icra.2018.8461076
18. **Aref M. M., Mattila J.** Deep learning of robotic manipulator structures by convolutional neural network. *2018 Ninth International Conference on Intelligent Control and Information Processing (ICICIP)*, Wanzhou, China, November 9–11, 2018, pp. 236–242. doi:10.1109/icicip.2018.8606719
19. **Peiyuan Liao.** Deep neural network based subspace learning of robotic manipulator workspace mapping. *2018 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)*, Athens, Greece, May 19–21, 2018, pp. 109–120. doi:10.1109/iccairo.2018.00027
20. Chien J.-T. *Source separation and machine learning*. Cambridge, London, Elsevier Inc., 2019. 384 p. doi:10.1016/C2015-0-02300-0
21. **Solovyeva E.** Cellular neural network as a non-linear filter of impulse noise. *2017 20th Conference of Open Innovations Association FRUCT (FRUCT20)*, Saint-Petersburg, Russia, April 3–7, 2017, pp. 420–426. doi:10.23919/FRUCT.2017.8071343
22. **Xiang J., Li Q., Dong X., Ren Z.** Continuous control with deep reinforcement learning for mobile robot navigation. *2019 Chinese Automation Congress (CAC)*, Hangzhou, China, November 22–24, 2019, pp. 1501–1506. doi:10.1109/cac48633.2019.8996652
23. **Ravichandiran S.** *Hands-on reinforcement learning with Python. Master reinforcement and deep reinforcement learning using OpenAI Gym and TensorFlow*. Birmingham, Packt Publishing, 2018. 320 p.

УДК 004.896
doi:10.31799/1684-8853-2020-5-24-32

Система управления на основе нейронных сетей при обучении с подкреплением для робота-манипулятора

Е. Б. Соловьева[a], доктор техн. наук, доцент, orcid.org/0000-0001-8204-6632, selenab@hotbox.ru
А. Абдуллах[a], аспирант, orcid.org/0000-0002-4024-9201
[a]Санкт-Петербургский государственный электротехнический университет «ЛЭТИ», Профессора Попова ул., 5, Санкт-Петербург, 197376, РФ

Введение: в силу высокой гибкости и способности перемещать тяжелые предметы с большими вращающими моментами и усилиями роботизированная рука, называемая роботом-манипулятором, является часто используемым промышленным роботом. **Цель:** повысить качество управления роботом-манипулятором с семью степенями свободы, представленным в среде симулятора V-REP, применяя метод обучения с подкреплением для глубоких нейронных сетей. **Методы:** оценка сигнала политики действия посредством построения численного алгоритма с использованием глубоких нейронных сетей. Сеть актора отправляет сигнал действия в роботизированный манипулятор, а сеть критика выполняет численную аппроксимацию для вычисления оценки функции (*Q*-оценки). **Результаты:** мы создаем модель робота и его окружающую среду, используя библиотеку обучения с подкреплением в MATLAB и направляя выходной сигнал (сигнал действия) к симулятору робота в программе V-REP. Робот обучается достижению объекта в рабочем пространстве при взаимодействии с окружающей средой и при расчете вознаграждения за это взаимодействие. Модель наблюдения создана с применением трех видеосенсоров. С помощью метода глубокого обучения модель агента, представляющего собой робот-манипулятор, построена на базе четырехслойных нейронных сетей актора и критика. Модель агента обучалась в течение нескольких часов до момента достижения роботом объекта в своем рабочем пространстве с приемлемой точностью. Основное преимущество предлагаемого управления над управлением с учителем заключается в том, что робот одновременно обучается и выполняет перемещение в непрерывном пространстве действий. **Практическая значимость:** полученные результаты применяются для управления движением робота-манипулятора без конструирования кинематических моделей, в результате уменьшается сложность расчетов и обеспечивается универсальность решения.

Ключевые слова — кинематическое управление, обучение с подкреплением, глубокое обучение, робот-манипулятор, глубокая нейронная сеть, глубокий детерминированный градиент политики.

Для цитирования: Solovyeva E. B., Abdullah A. Controlling system based on neural networks with reinforcement learning for robotic manipulator. *Информационно-управляющие системы*, 2020, № 5, с. 24–32. doi:10.31799/1684-8853-2020-5-24-32
For citation: : Solovyeva E. B., Abdullah A. Controlling system based on neural networks with reinforcement learning for robotic manipulator. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2020, no. 5, pp. 24–32. doi:10.31799/1684-8853-2020-5-24-32