Convolutional Fitted Q Iteration For Vision-Based Control Problems

Dongbin Zhao, *IEEE Senior Member,* Yuanheng Zhu, *IEEE Member,* Le Lv, Yaran Chen, and Qichao Zhang

***Abstract*—In this paper a deep reinforcement learning (DRL) method is proposed to solve the control problem which takes raw image pixels as input states. A convolutional neural network (CNN) is used to approximate Q functions, termed as Q-CNN. A pretrained network, which is the result of a classification challenge on a vast set of natural images, initializes the parameters of Q-CNN. Such initialization assigns Q-CNN with the features of image representation, so it is more concentrated on the control tasks. The weights are tuned under the scheme of fitted Q iteration (FQI), which is an offline reinforcement learning method with the stable convergence property. To demonstrate the performance, a modified Food-Poison problem is simulated. The agent determines its movements based on its forward view. In the end the algorithm successfully learns a satisfied policy which has better performance than the results of previous researches.**

***Index Terms*—Deep reinforcement learning, convolutional neural network, fitted Q iteration, vision-based control.**

I. Introduction

BIOLOGICAL systems have always inspired the development of artificial intelligence. Deep learning (DL) [1] and reinforcement learning (RL) [2] are two representatives. The former is based on the study of neurology that the increase in the depth of neural networks helps to improve the capabilities of nonlinearity and abstraction. While RL concentrates on imitating the mechanism of animals learning in environments. Both of them have gained great success in their fields.

For these years, DL has gained great success in the field of image classification and object recognition systems. Its capability of image representation is proved to be competent or even superior to the traditional machine learning methods. Its development is based on the supposition that the more layers are structured in neural networks, the approximation is more nonlinear and sophisticated, making it possible to extract more general features. Among the developed DL methods, convolutional neural networks (CNNs) are one of the highlights that have been studied for many years and have gained most exciting success. CNNs emphasize the stationarity of statistics and locality of pixel dependencies in images [3]. Compared to fully connected NNs, fewer connections and parameters are contained in CNNs, making it easier to train networks and extend the depths and breadths. Some applications of CNNs have demonstrated better performance than traditional image techniques [3].

D. Zhao, Y. Zhu, L. Lv, Y. Chen, and Q. Zhang are with the State Key Laboratory of Management and Control for Complex Systems, Institution of Automation, Chinese Academy of Sciences, Beijing 100190, China. e-mail: {dongbin.zhao, yuanheng.zhu, lvle2012, chenyaran2013, zhangqichao2014}@ia.ac.cn.

This work is supported by National Natural Science Foundation of China (NSFC) under Grants No. 61273136, No. 61573353 and No. 61533017.

In contrast to DL perceiving images, RL concentrates on strategies. It aims to make an agent learn proper actions through the interaction with the environment. At each step, the agent receives a reward which indicates the goodness or badness of its action. The target is to find a policy that gains the most rewards. According to the types of the interaction, RL is categorized into online RL and offline RL. The former one updates policies in real-time based on online observations. It is capable to solve model-unknown problems. Offline RL tunes policies on the basis of known models or collected data. Its learning procedure is stable and convergent. In traditional cases, RL mostly takes low-dimensional physical variables as input states. When considering pixel images as input states, the dimension becomes hundreds or thousands so algorithms are more vulnerable to the curse of dimensionality. Early research tackles the problem by comparing images in pixels and clustering in the Euclidean distance [4], [5], [6]. Since the image content is not really understood, the approach is rather superficial. The efficient way is to extract features from images, and learn a mapping function from features to actions directly [7].

Due to the advantages of DL and RL in their own aspects, the combination provides a novel approach to the vision-based control problems which take pixel images as input states. The first successful framework is proposed by Lange and Riedmiller in [8]. Deep auto-encoders (DAEs) are utilized to construct a low-dimensional feature space from input images. A batch-RL method is followed to train a Q function approximator which takes the feature variables instead of the pixel images as inputs. They first choose a simple task (puddle world) as their experiment object, and then extend the method to a more difficult and realistic problem (racing slot car) in [9]. A more exciting success is gained by Mnih et al. [10], [11]. They apply DRL to a series of computer games and achieve superior results than human beings. Their success ascribes to the image processing with a CNN structure and the online updating scheme with experience replay Q learning. More works are further inspired by their research [12], [13], [14]. Deep recurrent NNs are combined with RL in [15] to play text-based games. However, for online RL, it has been proved that the distribution of online observation has a deterministic influence on the performance of the learned policy. In some complicated systems, the most crucial states are sparse in the data set, making it hard to learn a satisfying policy. Besides, online algorithms also have an inclination of divergence in comparison with offline RL.

In this paper, we propose an offline DRL method to solve a vision-based Food-Poison problem. The method is applicable

978-1-5090-0620-5/16/$31.00 ©2016 IEEE

to other similar problems which take pixel images as inputs. In order to achieve a satisfying result with only a small set of data, we construct a deep network with a pretrained CNN to approximate Q functions. The weights are tuned in the scheme of fitted Q iteration (FQI) method. The learning process is fast and the results are promising. Even though the problem becomes more difficult than its original version, the results are superior. The main contribution is that a simple and efficient realization of DRL is established and a new benchmark is constructed to test the performance.

The paper is organized as follows. In Section II, we introduce fitted Q iteration, an offline RL algorithm. Next introduces CNNs to approximate Q functions and presents the convolutional fitted Q iteration algorithm. The vision-based food-poison problem is described in Section IV and the experimental study is conducted in Section V. Our discussion and conclusion is summarized in the end.

II. Fitted Q Iteration

Typically the control objects in RL can be described by 4-tuples (S, A, R, T), where S denotes the state space, A represents finite actions or action space, R is the reward function, and T is the transition function. If the problem is further a Markov Decision Process (MDP), the action a_t at instant t is determined only by the current state s_t, denoted by a policy function $\pi : S \to A$. The target is to find the optimal policy π^* that gains the highest sum of rewards with a discounted factor γ in the mathematical form

$$\pi^*(s) = \arg\max_{\pi} \sum_{k=0}^{\infty} \gamma^k r_k, \quad s_0 = s, a_k = \pi(s_k) \tag{1}$$

where s_k, a_k, r_k indicate the state, action, and reward at step k.

To solve the optimal problem, we define a Q function which estimates the current policy. It maps the state and action into a scalar value, and is defined by

$$Q(s, a) = \sum_{k=0}^{\infty} \gamma^k r_k, \quad s_0 = s, a_0 = a, a_k = \pi(s_k) \tag{2}$$

From the definition, $Q(s, a)$ represents the sum of rewards starting from the state s and action a, and followed by the policy π. The optimal policy π^* corresponds to the optimal Q function that has the highest values

$$Q^*(s, a) = \sum_{k=0}^{\infty} \gamma^k r_k, \quad s_0 = s, a_0 = a, a_k = \pi^*(s_k) \tag{3}$$

which can be rewritten in the Bellman principle

$$Q^*(s_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \tag{4}$$

After solving Q^*, π^* is available by

$$\pi^*(s_t) = \arg\max_{a_t} Q^*(s_t, a_t) \tag{5}$$

From the above analysis, it is crucial to obtain Q^* by solving (4). An efficient approach is value iteration (VI). In VI, we define an operator $\mathcal{T}$ that maps Q functions to Q functions. Starting from an initial $Q^{(0)}$, a new function is calculated at each iteration

$$\begin{aligned} Q^{(i)}(s_t, a_t) &= \mathcal{T}[Q^{(i-1)}](s_t, a_t) \\ &= r_t + \gamma \max_{a_{t+1}} Q^{(i-1)}(s_{t+1}, a_{t+1}) \end{aligned} \tag{6}$$

Since the operator $\mathcal{T}$ is contractive, after sufficient iterations the Q errors are reduced following the bound

$$\|Q^{(i)} - Q^*\|_\infty \le \gamma^i \|Q^{(0)} - Q^*\|_\infty \tag{7}$$

The corresponding greedy policy $\pi^{(i)}$ with respect to $Q^{(i)}$

$$\pi^{(i)}(s_t) = \arg\max_{a_t} Q^{(i)}(s_t, a_t) \tag{8}$$

is suboptimal in comparison with π^*. Detailed analysis is available in [16].

In practical applications, it is common to encounter some systems that have large state sets or continuous state spaces. It is infeasible to record Q values for every state-action pair (s, a). One solution is based on approximation techniques that use fewer parameters to represent complicated Q functions. Let θ be the parameters that define the approximator of Q functions. The approximate Q value at pair (s, a) is denoted by $\hat{Q}(s, a|\theta)$. The fitted Q iteration (FQI) that solves the iteration process with approximators becomes

1) Calculate target values over a data set D

$$y^{(i)}(s_t, a_t) = r_t + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}|\theta^{(i-1)}) \tag{9}$$

2) Find a set of parameters $\theta^{(i)}$ that minimize the loss function

$$\theta^{(i)} = \arg\min_{\theta} \sum_{D} \left[\hat{Q}(s_k, a_k|\theta) - y^{(i)}(s_k, a_k)\right]^2$$

where $\theta^{(i)}$ represent the approximator parameters at the i-th iteration of FQI. D indicate a set of samples (s_k, a_k, r_k, s_{k+1}) that are used to train the approximators.

Due to errors in the approximation, the convergence and stability of QI is disturbed. [4] states that if the approximator is a nonexpansion, FQI still converges. Under this condition, the final converged approximate solution θ^* has the following suboptimal property [16]

$$\|Q^* - \hat{Q}(\theta^*)\|_\infty \le \frac{2\epsilon}{1-\gamma} \tag{10}$$

where ϵ represents the approximate precision of the approximator. Unfortunately, the above requirement is quite restrict. Most approximators like NNs, RBFs, GPs are not nonexpensions. But if the approximator has high precisions, it is helpful to stabilize the learning process and avoid divergence.

In traditional applications, RL algorithms widely choose physical variables as input states, and employ NNs as approximators due to the flexibility and universal approximation. However, when considering high-dimensional input states like images, shallow NNs are hard to learn the complex mapping functions. In order to ensure the precision of approximators and the convergence of FQI, a deep NN structure is preferred.

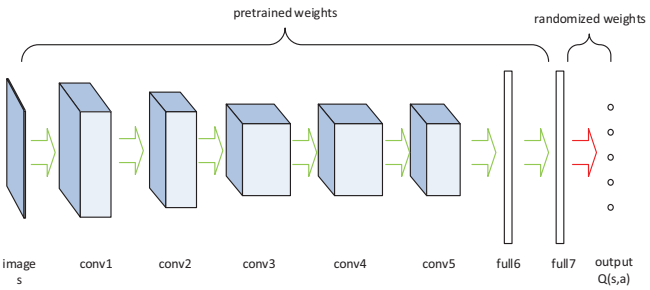


Fig. 1. The structure of Q-CNN

III. Convolutional Fitted Q Iteration

A. Approximate Q Function with CNN

To tackle the vision-based problem we adopt a convolutional neural network that maps images to Q values. The NN structure is introduced from [17] but is adapted to our tasks. The original CNN is used for an image classification challenge. It has 5 convolutional layers and 3 fully connected layers. It inputs 224×224 RGB images and outputs 1000 values on the probabilities of each class. More details are available in their literature.

For our purpose, the output layers is desired to produce a vector of values at the same number of actions. Each value indicates the Q value at the corresponding action with the input image. The rest layers remain unchanged as plotted in Fig. 1. We denote the network as Q-CNN to specify the approximation of Q functions. Given a pair (s, a) where s are now a set of RGB pixels, subtract the image mean values (calculated from the data set) from s. Then enter the result into Q-CNN and output the Q values. Denote the one with respect to action a as $Q(s, a)$.

Except the network structure is constructed following [17], we also use their trained results to initialize Q-CNN. This design is based on the facts that shallow layers mainly focus on extracting features in a statistics point, while deep layers is guided by tasks to reduce loss functions on the basis of the extracted features. To this end, it requires millions of data to train the shallow layers before finding a group of parameters that are capable to understand images in a high level. The authors train the network based on ILSVRC-2012 data set with about 1.2M images. The shallow layers are competent to the classification challenge. The reutilization of the network saves the process of learning image representation, as well as helps to concentrate the algorithm on training deep layers under the RL framework. Hence the weights in our Q-CNN are set to the trained network in [17] except the output layer. The output layer is defined to produce Q values and its parameters are initialized randomly around zero.

B. Convolutional Fitted Q Iteration Algorithm

The algorithmic flow of the convolutional fitted Q iteration (CFQI) is presented in Algorithm 1. First we use the trained network to define the hidden layers of Q-CNN, and structure the output layer with random parameters. A data set is prepared. At each iteration, based on the current Q-CNN,

Algorithm 1 Convolutional fitted Q iteration algorithm

```
Initialize Q-CNN with the pretrained network and randomize the output layer parameters
Prepare data set D, define the number of iterations N_i and the number of cycles N_c
for i = 1 : N_i do
    Calculate the target values y^(i) for D based on the last results θ^(i−1) by using (9)
    for j = 1 : N_c do
        Apply stochastic gradient to train θ^(i) based on the loss function (11)
    end for
    Replace Q-CNN with the current θ^(i)
end for
```

calculate the target values $y^{(i)}$ for the data set. After that, the network is trained using stochastic gradient descent with a batch size of 100 samples, momentum of 0.9, and weight decay of 0.0005. The loss function is defined by

$$L = \frac{1}{2N} \sum_{(s_k, a_k) \in D} \left[\hat{Q}(s_k, a_k | \theta^{(i)}) - y^{(i)}(s_k, a_k) \right]^2 \quad (11)$$

The training law for the parameters $\theta^{(i)}$ is

$$\begin{aligned} w_{j+1}^{(i)} &= 0.9 \cdot w_j^{(i)} - 0.0005 \cdot \alpha \cdot \theta_j^{(i)} - \alpha \cdot \left\langle \frac{\partial L}{\partial \theta_j^{(i)}} \right\rangle_{D_j} \\ \theta_{j+1}^{(i)} &= \theta_j^{(i)} + w_{j+1}^{(i)} \end{aligned} \quad (12)$$

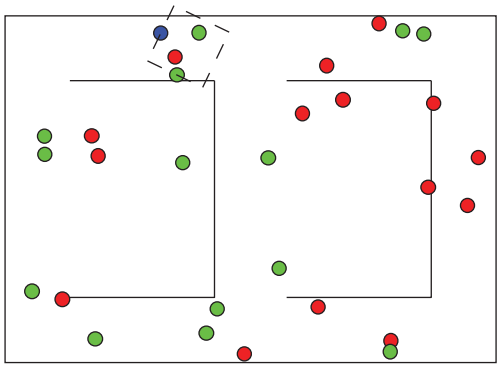
where j indicates the index of the minibatch updating, w are the momentum variables, α is the learning rate, and $\left\langle \frac{\partial L}{\partial \theta_j^{(i)}} \right\rangle_{D_j}$ represents the gradient of the loss function over the batch D_j at the current parameters $\theta_j^{(i)}$. At each iteration, the updating over the whole data set is repeated for several cycles at the same target values $y^{(i)}$. After the cycles, the learned parameters $\theta^{(i)}$ replace Q-CNN and calculate new targets $y^{(i+1)}$.

IV. Vision-Based Food-Poison Problem

A. Problem description

This part introduces a vision-based Food-Poison problem which is used as a benchmark to test our algorithm. The problem is converted from a demonstration which is first presented in [18]. An *agent* (blue) moves in a *world* area, which contains *walls* and two types of items, *foods* (red) and *poisons* (green). Five actions determine the directions of the agent by turning different angles, including *forward*, *left*, *right*, *sharp left* and *sharp right*. It is desired to design a controller that navigates the agent towards foods and away from poisons. The system keeps a fixed number of items in the world. If an item is eaten by the agent, a new one is created at a random position with a random type. In the original problem, the agent has 9 eyes pointing at different angles and each eye returns 3 values that report the distances to walls, foods and poisons separately.

In this paper, we convert the problem to a vision-based version, illustrated in Fig. 2. The controller is designed to take the forward views of the agent as inputs and output the



(a) A schematic of food-poison problem

(b) The pixel image observed by the agent

Fig. 2. The vision-based food-poison problem.

5 actions. The input images correspond to the square areas in front of the agent with the size of 85×85, which equals to the covering area of the 9 eyes in the original problem. At each step, the agent receives a reward that comprises of 4 parts, $r = r_p + r_f + r_w + r_a$. Each part is related to the current positions of the agent and its surroundings, as well as the agent's actions:

r_p	if the agent touches poisons, -6; others 0
r_f	if the agent touches foods, 5; others 0
r_w	$\min(1, 2d_w/85)$
r_a	if the action chooses forward, $0.1r_w$; otherwise 0

where d_w indicates the distance between the agent and the nearest wall. From the reward definition, the agent gains highly if it eats foods, avoids poisons, keeps away from walls and prefers to move forward. The discounted factor used in RL selects 0.7.

B. Data set preparation

From the above description of CFQI, the data set D is crucial in the implementation. In addition, existing literatures have verified that the distributions of the samples impact the performance of the final learned policies. Considering the food-poison problem, the agent is expected to achieve 4 types of goals, which are listed according to their significance:

G1 avoid touching poisons
G2 move towards foods and eat them
G3 keep away from walls
G4 try to move forward

The data set is designed in the view of the 4 goals. Note that since no policies have been learned, all the samples are produced under random actions. 5 kinds of samples are selected to form the data set. Some of them correspond to a single tuple (s_k, a_k, r_k, s_{k+1}) at one instant, while some are several sequential tuples. The detailed descriptions are given as follows:

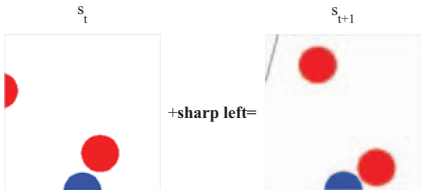
D1 samples at one instant that the agent touches poisons
D2 samples at one instant that there exist poisons close to the agent
D3 sequential samples that the agent moves 4 steps and eats foods
D4 sequential samples that the agent moves 8 steps and eats foods
D5 arbitrary samples at one instant

D1 and D2 correspond to the goal of G1. D1 makes agent aware of the penalty when touching poisons. D2 extends the influence of the penalty to the situation where the agent is close to poisons. Continuing to get close to poisons may result in worse punishment just like the case in D1. D3 and D4 satisfy G2 that expects the agent to move towards foods from distant positions. D5 includes the expectation of G3, because agent may be randomized to a position that is close to walls. G4 is embodied in D1–D5, since forward action always indicates better rewards in comparison with the other actions. We choose the size of the 5 sets as 1500, 1400, 400, 1600, 5000, so there is 9900 samples in the data set.

In order to augment the effect of the data set, we apply *flipping* to the samples. The symmetry of the problem underlies the data augmentation, like the illustration given in Fig. 3. At the state s_t given in subfigure (a), after taking the action *sharp left*, the agent observes the image of s_{t+1}. If we horizontally flip s_t as depicted in subfigure (b), it is reasonable to infer that after taking the action *sharp right*, the next image is also horizontally flipped.

V. Experimental Results

The experiment is performed in the MATLAB code with the toolbox MatConvNet [19]. We use a GeForce GTX Titan X GPU to accelerate the training process. 50 iterations are conducted and each iteration contains 5 cycles of updating. It takes about 6 hours to end the algorithm. The loss errors at each cycles are depicted in Fig. 4. 5 cycles at the same iteration is plotted in the same color. It is observed that at the first several iterations, the loss errors reduce rapidly under the framework of RL. The reduction is benefitted from the initialization of Q-CNN by the pretrained results from [17]. The learning process is concentrated on the updating of deep layers. In the following iterations the curve is quite flat, indicating the Q iteration mechanism plays a more major role in updating the policies. We further calculate the errors of Q values at two successive iterations shown in Fig. 5. The curve generally reflects the updating of Q-CNN. The curve is first stabilized to a small error bound at about 10th iteration. After that, tiny vibrations remain existing in the following iterations due to the approximation errors. The convergence



(a) A transition of states at action 'sharp left'.

(b) The equivalence of the transition at action 'sharp right'.

Fig. 3. An illustration of horizontal flipping.

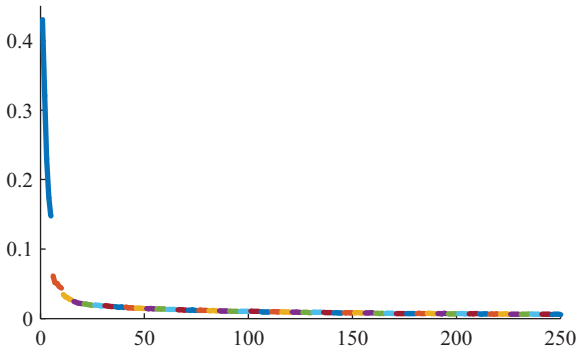


Fig. 4. The curve of loss errors along updating cycles.

rate is exciting since the actual needed implementation time is about 2 hours.

Now apply the learned result to the problem and observe the performance. Fig. 6 presents a typical trajectory. All the input images are presented in Fig. 7. At the beginning, a food is observed in the visual area. The agent moves towards the food and successfully eats it (row 3, column 8). After that, a poison nearby is immediately perceived, so the agent turns sharply to keep away from it. In the end the agent navigates perfectly through the gap between another two poisons. At each instant, Q-CNN provides the proper greedy actions under the current views to achieve the 4 goals listed above. The curves of the Q values in the trajectory are drawn in Fig. 8. A noticeable peak appears in the maximum Q curve. The peak point indicates the moment that the agent is about to eat the food. At the other instants, the curve is relatively flat. But in the second subfigure that depicts all Q values, at some instants some actions have much worse values than the others. These are the moments that the agent gets very close to poisons or walls. If taking the mentioned actions, a worse punishment will occur. So their Q

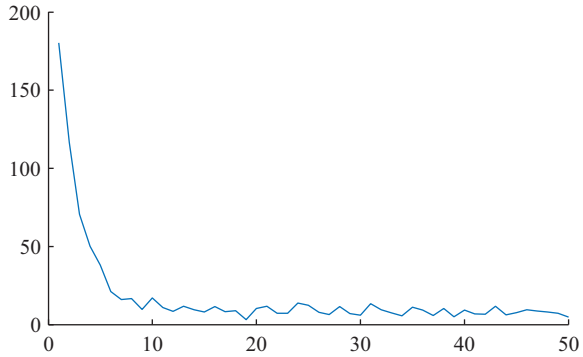


Fig. 5. The curve of Q errors along iterations.

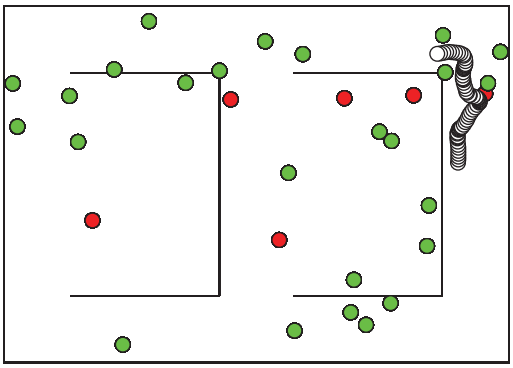


Fig. 6. A typical trajectory in the experiment.

values are lower and the greedy policy does not choose them.

In the end we compare our results with the original experiment presented in [18]. The latter takes 1D sensor values as inputs and uses experience-replay Q learning to train the network. The experiment is repeated for 5 times and the

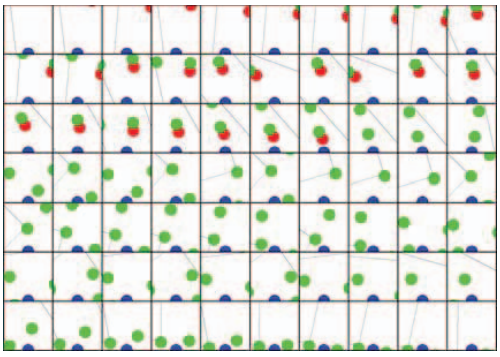
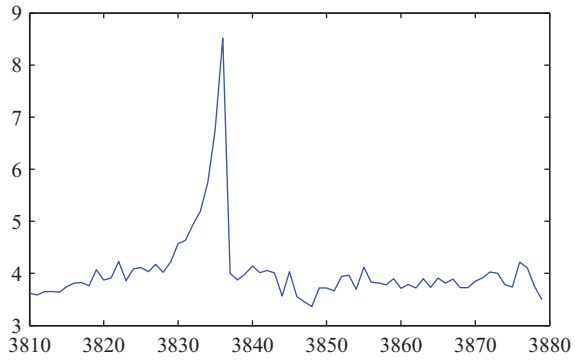
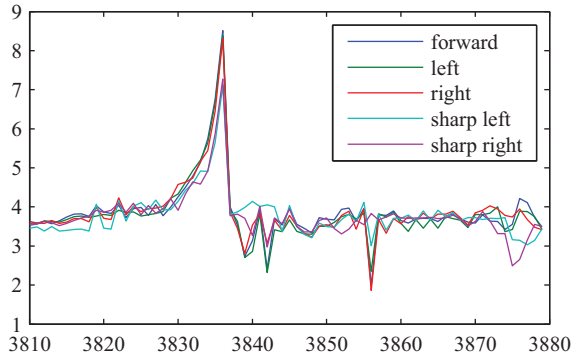


Fig. 7. The visual images during the trajectory. The images are arrayed from left to right, top to bottom.



(a) The maximum of Q values



(b) Q values at each action

Fig. 8. The curves of Q values during the trajectory.

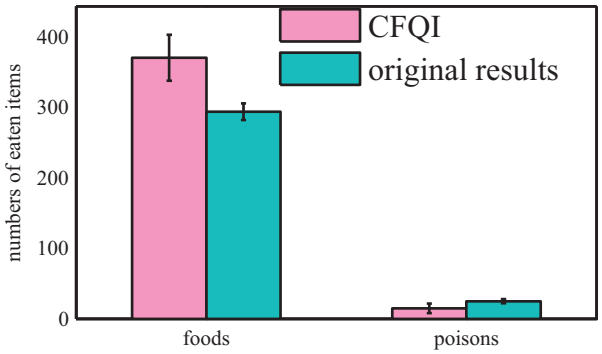


Fig. 9. Statistics of eaten items within 100000 steps by the results of CFQI and [18] over 5 trials.

numbers of eaten foods and poisons in 100000 steps are counted. The statistics results are presented in Fig. 9. It is revealed that our method is capable to learn policies that eat more food and less poisons, even though the problem is complicated by taking pixel images as input states.

VI. Conclusion and Discussion

The proposed convolutional fitted Q iteration algorithm combines RL and DL successfully. It is capable to solve one kind of control problems that take pixel images as inputs. CNNs are adapted to approximate Q functions with the output layer producing Q values. A pretrained classification network that has been trained on a vast set of images, initializes our Q-CNN so that it has the excellent capability of image representation. Such reutilization helps to speed up the learning process and reduce the dependence on large training sets. With the defined data set, the algorithm converges rapidly in the framework of the iterative RL method. A complicated food-poison problem is simulated to verify the performance. Visual images in front of the agent are taken as the input states. Policies determine actions based on the inputs to avoid poisons and eat foods. The new task increases the learning difficulty but CFQI reveals better performance than the original experiment.

References

[1] Y. LeCun and M. Ranzato, "Deep learning tutorial," in *Tutorials in International Conference on Machine Learning (ICML13)*, 2013.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[4] G. J. Gordon, "Stable function approximation in dynamic programming," Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep., 1995.

[5] D. Ernst, R. Marée, and L. Wehenkel, "Reinforcement learning with raw image pixels as input state," in *Proceedings of the 2006 Advances in Machine Vision, Image Processing, and Pattern Analysis International Conference on Intelligent Computing in Pattern Analysis/Synthesis*, ser. IWICPAS'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 446–454.

[6] S. Jodogne and J. H. Piater, "Closed-loop learning of visual control policies," *J. Artif. Int. Res.*, vol. 28, no. 1, pp. 349–391, Mar. 2007.

[7] R. Legenstein, N. Wilbert, and L. Wiskott, "Reinforcement learning on slow features of high-dimensional input streams," *PLoS Comput Biol*, vol. 6, no. 8, p. e1000894, 08 2010.

[8] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, July 2010, pp. 1–8.

[9] S. Lange, M. A. Riedmiller, and A. Voigtländer, "Autonomous reinforcement learning on raw visual input data in a real world application," in *The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, June 10-15, 2012*, 2012, pp. 1–8.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. a. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[12] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," *CoRR*, vol. abs/1509.06461, 2015.

[13] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," *CoRR*, vol. abs/1507.04296, 2015.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[15] K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," *CoRR*, vol. abs/1506.08941, 2015.

[16] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 2010.

[17] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *British Machine Vision Conference*, 2014.

[18] (2015) ConvNetJS Deep Q Learning Demo. [Online]. Available: http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html

[19] A. Vedaldi and K. Lenc, "MatConvNet – convolutional neural networks for MATLAB," 2015.