Transfer in Reinforcement Learning Via Shared Q-Network

Zongze Li[a], Zijia Yan, Tingting Zhang, and Yuqiang Qu

Department of Software Engineering, Jilin University, Changchun, China

[a]lizz5516@mails.jlu.edu.cn

Keywords: Transfer, Reinforcement Learning, Q-network

Abstract: Transfer learning is one of the hot issues in the field of reinforcement learning. Many reinforcement learning algorithms can only solve a single special task, and their models and algorithms are designed for a single state space, which is not universal. When tasks change, it will cost a lot to change the model and configuration of reinforcement learning algorithm, or even it is not feasible. Transfer learning is born to solve this problem. Through the transfer of learning experience or learning method, the machine learning algorithm can keep the effectiveness and efficiency in the constantly updated tasks and data.

1. Introduction

In this paper, we will explain the transfer algorithm of deep reinforcement learning based on shared Q-network, and improve the convergence speed of new tasks. In section 2, we will introduce reinforcement learning, transfer learning and reinforcement learning transfer algorithm based on common characteristics. In section 3, we will introduce shared q-network, and in section 4, we will introduce shaping function which can be transferred from experience in shared q-network to q-network. The last section 5 is our summary.

2. Background

In this section, we present related work and background concepts such as Markov process, reinforcement learning, deep q-Learning and transfer learning.

2.1 Reinforcement Learning

A reinforcement learning environment is typically formalized by means of a Markov decision process (MDP). The Markov decision process can be expressed as a tuple(S, A, P, R, γ). Where S is the set of states in the decision-making process, A is the set of actions in the decision-making process, P is the transition probability between states, R is the return after taking one action to reach the next state, and γ is the discount factor.

In MDP, the decision can be represented by $\pi(a|s)$, which means the probability of action a in s state at step t. The goal is to learn a deterministic stationary policy π, which maps each state to an action, such that the value function of a state s, i.e., its expected return received from time step t and onwards, is maximized.

At time t, action a is selected with probability π in s state, and the state action value generated by action a is $q_\pi(s, a)$.The optimal state-action function is defined as $q^*(s, a) = max_\pi q(s, a)$, which represents the maximum state-action value in all policies.[1]

$$q^*\ (s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a max_{a'} q_\pi^*(s', a')$$

Watkins introduced an algorithm to iteratively approximate q^*.In the q-learning algorithm, a q-table containing state-action pairs is stored. Each entry in the table contains a $\hat{q}(s, a)$, which is the current estimate of the actual $q^*(s, a)$ value. $\hat{q}(s, a)$ is updated according to the following rules,

$$\hat{q}(s, a) \leftarrow (1 - \alpha_t)\hat{q}(s, a) + \alpha_t(r + \gamma max_{a'} \hat{q}(s', a'))$$

Where α t is the learning rate at time step t and r is the reward received for performing action an in state s. [2]

The purpose of reinforcement learning is to find the optimal strategy to maximize the cumulative return function.

2.2 Deep Q-learning

When the set of states for a problem is large, the algorithm described above becomes unusable. And this is, one possible way to model it is to approximate the value function. A state value function $\hat{v}$ is introduced, which is described by a parameter w and takes state s as input. After calculation, the value of state s is obtained.

$$\hat{v}(s, \mathrm{w}) \approx v_{\pi}(s)$$

Similarly, an action value function q is introduced, which is described by a parameter w and accepts state s and action a as input. After calculation, the action value is obtained.

$$\hat{q}(s, a, w) \approx q_{\pi}(s, a)$$

There are many approximation methods for the value function, the most widely used of which is the neural network. For the state value function, the input of the neural network is the eigenvector of state s, and the output is state value $\hat{v}(s, w)$. For the action value function, there are two methods: one is input state s and action a, and the corresponding action value $\hat{q}(s, a, w)$ is output; the other is the eigenvector of only input state s, and the number of actions in the action set will have as many output .

The basic idea of DQN algorithm comes from q-learning, but different from q-learning, the calculation of its q-value is not directly calculated by the state value s and the action a, but by the q-network. The input of DQN is the state vector $\emptyset(\mathrm{s})$ corresponding to state s, and the output is the action value function q of all actions in this state.

The main technique used by DQN is experience replay, that is, to save the rewards and status updates from each interaction with the environment for future q-value updates. The target q-value obtained through the experience replay and the q-value obtained through the calculation of the q-network must have errors, so we can update the parameter w of the neural network through the reverse propagation of the gradient. When w converges, we get the approximate q value, and then the greedy strategy is solved. [3]

2.3 Transfer Learning

Transfer learning is an algorithm that gives the source domain and the source task, the target domain and the target task, and USES the source domain to obtain some knowledge in solving the task to improve the target task.

2.3.1 Related Task Share Common Features

Transfer learning needs to be carried out between similar but different tasks, which need to satisfy the need to be able to apply the experience of one task to another to a certain extent. Shared features are special states have same semantic between all the tasks.

For example, each robot has a heat sensor that allows the robot to collect heat information from the room. Or there is a signal receiver that can pick up signals from several sources in the room. If the heat information and signals in the example above are somehow related to the object being sought, then heat and signals can be seen as shared features in both rooms. Thus, the i_{th} Markov process M_i can be defined as:

$$M_i =< S_i, A_i, P_i, R_i, D >$$

Where D is the shared feature space. All markov processes share a D and use it as a bridge for the transfer of experience. When a valued function in the state space is updated, a valued function in the shared feature space is also updated.[4]

2.3.2 Shaping

Shaping function is used to transfer experiences in the shared feature space to the state space of the new task. The i_{th} procedure in M_j is defined as:

$$\sigma_i^j =< s_i^j, d_i^j, r_i^j, v_i^j >$$

Where s is the state, d is the feature in the shared feature space, r is the reward of the current state, v is the value of the value function, thus the mapping from s to d can be established:

$$F: s_i^j \leftrightarrow d_i^j$$

Similar to the value function:

$$V_j: s_i^j \rightarrow v_i^j$$

Mapping of shared features to value functions can be established:

$$L: d_i^j \rightarrow v_i^j$$

During the empirical transfer, the corresponding feature of the state in F is found first, and then the value function value of the feature is obtained by L, and finally the value function of the state space is initialized by V with the newly obtained returns. So that the value function is closer to the convergent value. [4]

3. Knowledge Transfer

In this section, we show that after the agent has processed multiple related tasks, the agent can transfer the existing knowledge to the transfer task through the shaping function, and the performance of the agent dealing the transferred task will be significantly improved.

We use an experiment (maze task with an artificial shared feature space) to demonstrate the effect of knowledge transfer in dealing with a series of related tasks.

3.1 Shared Q-network

When dealing with a series of tasks, the previous experience does not apply to new tasks due to the different environments of the tasks. Each task should have a separate q network. When dealing with new tasks, these q-networks are built in different environments and cannot be shared or transferred. The experience in the shared feature space can be applied to all states, so we need to use the experience in this space to facilitate transfer to new tasks.

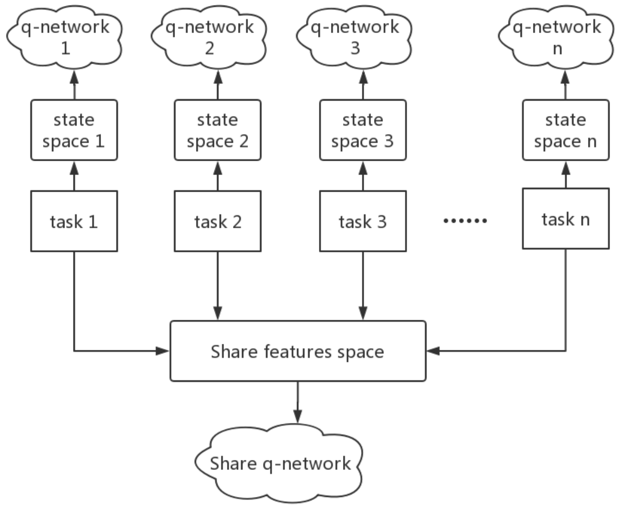

Figure 1. Structure of state space and share q-network.

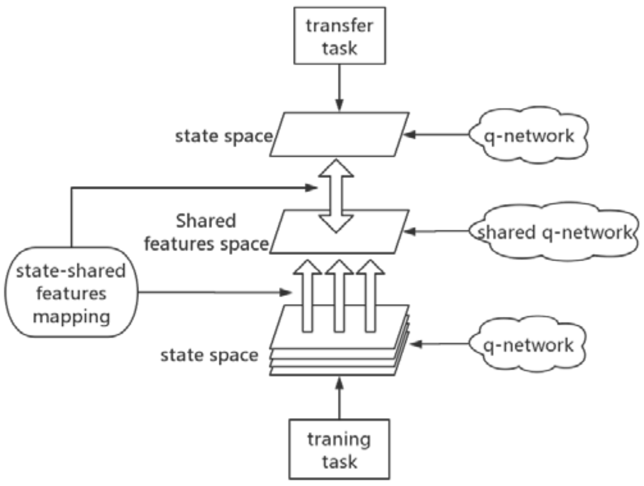

Figure 2. Mapping structure.

We use shared q-network built on a shared feature space to solve this problem. As shown in Fig.1, when the agent processes the task, the current state, action, reward, and arrival status are stored as experience in the q-network experience pool. At the same time, the current shared feature, actions, reward, and arrival shared feature are stored in the experience pool of the share q-network. Similarly, when q-network is trained in the extraction experience, the shared q-network is also trained from the experience of the shared feature experience pool.

The share features are not included in a single task, and each task contains only a subset of all the shared features (this problem will be explained in the experiment). Therefore, training a share q-network with only one task will only make it converge on the share features included in this task, but not for new tasks. So we need to use multiple tasks to train the share q-network until it converges on almost all share features.

3.2 Learning Shaping Function

In the traditional reinforcement learning algorithm, the q value is directly saved by the q-table. Q-table can be directly initialized by the shaping function mentioned in 2.4.2, and the knowledge transfer is conveniently performed. However, the DQN uses a q network that requires an input state to obtain a q value, which makes it impossible to initialize directly. Furthermore, enumerating all the states for initialization when using DQN is an unrealistic task.

Similarly, we can get the states, their corresponding shared features and their q-value tables from the three functions F, V, and L mentioned in 2.4.2. Similar to DQN, we directly optimize the q-network of the transfer task to make its q value in this state close to share q-network. The difference is that DQN approximates the q value corresponding to the action selected by the agent, and the loss of other actions is set to 0. In the transfer, the q values of all actions are approximated. So after getting two q-tables, you can train the network without modification.

In order to solve the problem of not being able to enumerate all the states, the approach we take is to transfer during the training process. In other words, not all the states are initialized before training. When the transferred task is processed, the agent determines whether the state needs to be transferred after each state is reached. This avoids the problem of enumerating unknowns and enumerating all states.

3.3 Maze Experiment

In this section, we will test the effect of knowledge transfer by taking a maze experiment. In the maze experiment, shared features are added manually to ensure that knowledge can be transferred between tasks.

Each maze is a square space divided into several grids, some of which have an agent, trap or target with a size of 1 grid. The trap position in each maze is random. The start position of the agent is fixed at the upper left corner, and the target is fixed at the lower right corner. If the agent moves to any traps, it will return to the starting point, and the goal of the task is to move the agent to the target. Fig. 3 is an example of a 10*10 scale.

Each state in the maze can be represented by two coordinates, x and y. The agent can move in one of four directions: up, down, left, and right, one grid at a time. If the agent moves outside the boundary, it will not move in place. The rewards for agents moving to traps, targets, and blank grids are -1, 10, -0.1, respectively.

But in this task, there is no shared feature space for all random maze, so we add 4 different beacons that can emit signals in the maze. The agent can receive signals and calculate the distance to the four beacons. In other words, by assigning 4 beacons from 0 to 3, the agent can get the distance to the four beacons. If there is some relationship between these distances and the target, the distances from the agent to the beacons can be taken as the shared feature space.

3.3.1 Experimental Structure

In order to record the convergence of the experiment, the agent needs to reach the target 50 times for each execution of the task. First, several training tasks are performed to train the shared q-network. Then several transfer tasks are executed. These tasks need to be executed twice and the shared q-network cannot be trained. In the first round, only q-network was trained. In the second round, Shared q-network was used to complete knowledge transfer, and then q-network was trained.

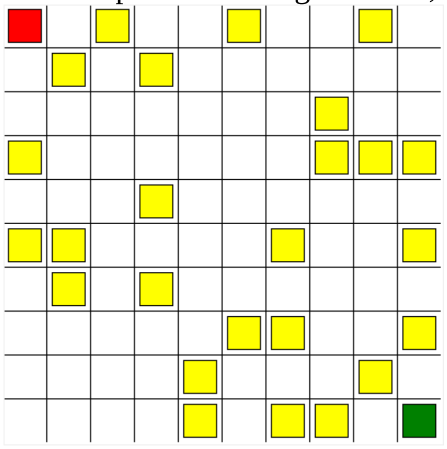

Figure 3. A 10*10 maze experiment. The goal is shown as a green block, traps are shown as yellow block, and agent is shown as a red block.

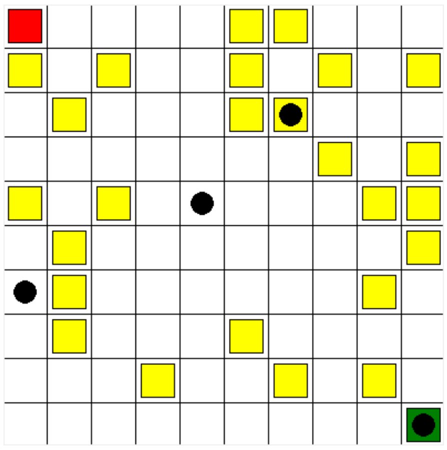

Figure 4. A 10*10 maze with beacons. Beacons are shown as black circle.

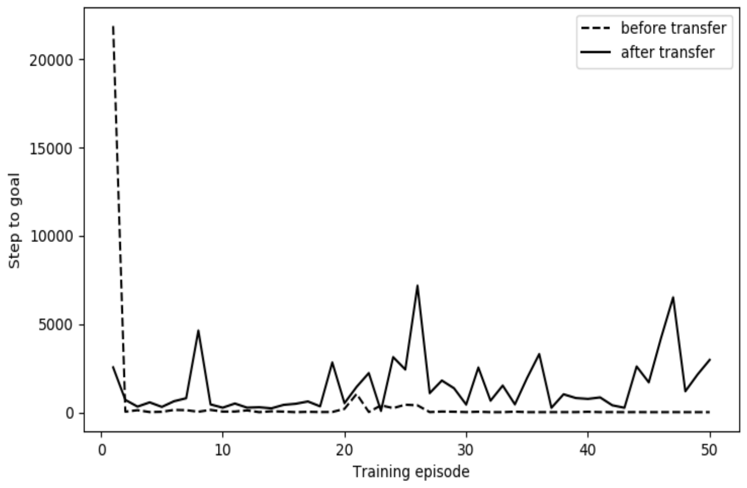

Figure 5. Steps to goal against episodes in the transfer task for agent that have completed 5 training tasks.

Agents perform reinforcement learning using DQN algorithm ($\alpha = 0.01$, $\gamma = 0.9$, $\varepsilon = 0.9$, memory size = 5000, butch size = 500, replace target iterator = 500). Q-network and Shared q-network have the same structure. The neural network has two layers with ten neurons in each layer. The coefficient is initialized to a random number within (0,1) and the constant is initialized to 0. The active function used in the first layer is relu6, the optimization algorithm is RMSprop, and the loss is the mean of the square deviation of the q value.

3.3.2 Following a Homing Beacon

The four beacons are numbered from 0 to 3. During the entire experiment, the source numbered 0 is always on the target, and the other three beacons are randomly distributed in the maze. The closer to the beacon numbered 0, the higher reward should be, and the signals from the other 3 beacons should be ignored. Fig. 4 shows a 10*10 maze with beacons.

Fig. 5 shows the number of steps required to reach the goal as the agents repeat episodes in the transfer task, after completing 5 training task, compared to the number of steps required by agents without completing training task. It can be seen that the number of steps of the agent without knowledge transfer rapidly drops from 20000 to near 0 and converges. The number of steps of agent who carried out knowledge transfer started at around 3,000, but did not converge after a period of decline, and there was a small fluctuation. In general, the agent completes five training tasks, which greatly reduces the initial number of steps, but also causes the number of steps to not converge.

The agent in Fig. 6 completed 10 training tasks, the other being the same as Fig. 5. Similar to the agent trained 5 times, the number of steps of the agent without knowledge transfer rapidly drops from 50000 to around 0 and converges. The difference is that the transferred agent begins to converge after a brief fluctuation around 10,000 steps and 0 steps. By increasing the number of training tasks, the initial steps are reduced and the number of steps is converged.

3.4 Discussion and Summary

These two experiments successfully demonstrated that even an agent that only completed a small number of training tasks can significantly reduce the number of steps to reach the target after the knowledge transfer, and the number of episodes during convergence is basically the same.

The non-convergence in the first experiment alerted us: when the shared q-network training is insufficient, the transfer knowledge may have negative consequences, that is, negative transfer. In this experiment, there are two factors that may cause negative transfer.

First, although the features in the shared feature space are applicable in all tasks and have a certain relationship with the target, the q value in the DQN algorithm is determined by the two parameters of state and action. In this experiment, the shared feature space and the state space have the same actions. However, it can be found that in different maze, the same distance (meaning the same feature) is not directly related to the action that should be selected. This problem can be solved by a lot of training, and all the q values of the features close to the target should be higher.

Second, the agent fits the q-network after it reaches the state rather than the overall fit. Since the fit changes the entire neural network, it is possible to change the network that has completed the fit, resulting in a negative transfer.

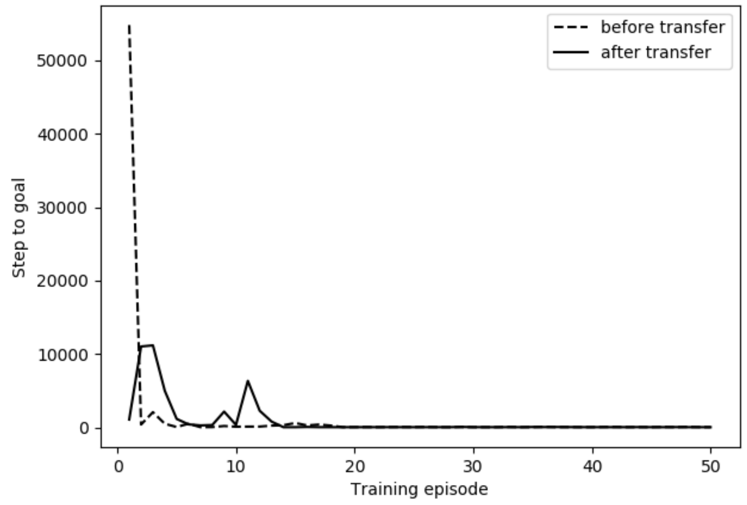

Figure 6. Steps to goal against episodes in the transfer task for agent that have completed 10 training tasks.

4. Summary

In general, training shared q-network and transfer knowledge through a series of related tasks can significantly reduce the initial number of learning. However, the selection of the shared feature space needs attention, and the shared q-network should be trained with enough training tasks. Otherwise, negative transfer will result, causing the data to not converge.

References

[1] Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction", the MIT Press, Mar. 1998.

[2] Watkins C.J.C.H., "Learning from Delayed Rewards", Cambridge University.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, et al. "Human-level control through deep reinforcement learning", nature, vol. 518, Feb. 2015, pp. 529-533, doi: 10.1038/nature14236.

[4] George Konidaris, Ilya Scheidwasser and Andrew G. Barto, "Transfer in Reinforcement Learning via Shared Features" Journal of Machine Learning Research, vol.13, May. 2012, pp. 1333-1371