Meta-Reward Model Based on Trajectory Data with k-Nearest Neighbors Method

Xiaohui Zhu
Department of Computer Science and Eng.
Waseda University
Tokyo, Japan
ricardo-zhu@toki.waseda.jp

Toshiharu Sugawara
Department of Computer Science and Eng.
Waseda University
Tokyo, Japan
sugawara@waseda.jp

***Abstract*—Reward shaping is a crucial method to speed up the process of reinforcement learning (RL). However, designing reward shaping functions usually requires many expert demonstrations and much hand-engineering. Moreover, by using the potential function to shape the training rewards, an RL agent can perform Q-learning well to converge the associated Q-table faster without using the expert data, but in deep reinforcement learning (DRL), which is RL using neural networks, Q-learning is sometimes slow to learn the parameters of networks, especially in a long horizon and sparse reward environment. In this paper, we propose a reward model to shape the training rewards for DRL in real time to learn the agent's motions with a discrete action space. This model and reward shaping method use a combination of agent self-demonstrations and a potential-based reward shaping method to make the neural networks converge faster in every task and can be used in both deep Q-learning and actor-critic methods. We experimentally showed that our proposed method could speed up the DRL in the classic control problems of an agent in various environments.**

***Index Terms*—reinforcement learning, reward shaping, machine learning.**

I. Introduction

In recent years, systems with *deep reinforcement learning* (DRL) have attained excellent performance in a number of challenging tasks in robot and game AI domains. However, a major limitation of such applications is the need for massive amounts of training data. Thus, it always takes a long time to train a DRL agent, especially when the reward of the environment is sparse or the original reward model contains uncertainty in the long horizon environment that makes it difficult for the agent to learn.

Recently, meta-reinforcement learning has been found to be a useful way to improve learning efficiency and the ability to accelerate the learning process for task adaptation [1]–[3]. For meta-learning in *reinforcement learning* (RL), one of the most popular algorithms is *model-agnostic meta-learning* (MAML) [4], which learns a versatile initialization of model parameters θ through reusing the data that the agent used to successfully solve the task in the environment. However, it is not easy to use MAML due to its slow running because it has to compute a great number of gradients backpropagated in neural networks.

To accelerate the DRL process, apart from using meta-learning, reward shaping is one of the useful and powerful solutions to reduce the amount of training data, and its goal is to shape the original rewards into a better reward structure that makes it easier for the DRL agent to learn to solve various problems. *Potential-based reward shaping* (PBRS) through the state of the agent and *potential reward function* is an effective method for directly shaping the rewards from the training environment [5]–[8], but it had poor performance and thus slowed the learning convergence when using it in DRL. To shape the reward for a DRL agent, the human teacher approach is useful to train the agents to navigate to earn more training rewards in practice [9]–[12]. Furthermore, Ibarz *et al.* [13] proposed a reward shaping method that used the expert demonstrations to pre-train the agent and the annotator to label the data. Obviously, the limitation of this method is the difficulty in getting expert demonstrations in various environments; thus, we often have to hand-craft the reward model while pre-training.

This paper focuses on the learning of motion of an agent, which is a control program for a mobile robot, vehicle, drone, or computer game player, to accelerate learning its movement or operations in various tasks. We assume that the activities of the agent can be expressed by the trajectory that is usually a long sequence of actions, so the agent's tasks are long horizon problems. Because these trajectory data describe the successful experiences of solving tasks, we attempt to make more use of these trajectory data so that the agent can identify how to execute the tasks efficiently by shaping the training reward. By using the shaped reward proposed here, the distribution of rewards in the environment is modified to make the agent learn faster to successfully control the movement so that we can reduce the training time.

Then, we propose a *trajectory action meta-reward model* (TAMRM) based on *k-nearest neighbors* (k-NN) and PBRS for DRL to learn how to execute the given tasks efficiently. Our contributions are: (1) this meta-reward model does not need the expert demonstration data by using the agent's self-demonstrations to shape the reward for training, and (2) this meta-reward model requires only a reasonable amount of memory and can predict the rewards in real time with fewer calculations without extra gradient backpropagation [14]. This model is versatile in the sense that it can be used in both the *actor-critic method* (AC method) [15] and the *deep Q-*

learning using the *deep Q-network* (DQN) [16]. This model also makes the agent learn faster and makes the PBRS more suitable for the DRL. We experimentally evaluated our reward shaping method by solving classic motion control problems in long horizon environments with a simple deep neural network for the AC method and the Q-learning.

II. Related work

There are many studies on improving learning efficiency to reduce the amount of training data or training time. For example, reward shaping is a very effective way to speed up training [17], [18]. PBRS modifies the original reward function by using potential-based shaping functions $\Phi(s)$ to make RL methods (e.g., Q-learning) converge faster [5], [19], [20]. PBRS usually depends on state s, but it has also been extended to *potential-based advice* (PBA) [21] to include *action* a in potential function $\Phi(s, a)$. It has also been extended to *dynamic PBRS* [22] by introducing *time* t into potential function $\Phi(s, t)$. Marom and Rosman [23] proposed a reward shaping framework based on the Bayesian method in Q-learning for executing complex tasks. Besides, by combining it with expert demonstrations for the complex tasks, the PBRS in RL became more powerful [24]. Zou *et al.* [14] used a method, which is similar to MAML, to pre-train the *dueling deep Q-network* (dueling-DQN [25]) and used the PBRS for fast convergence. On the other hand, our work differs from these efforts in that we shape the reward for the agent in the DRL without any pre-training stage.

Inverse reinforcement learning is also an effective way to find the reward function by using the expert demonstrations [26]–[29]. By combining the imitation learning and the DRL, Wu [30] created a hand-coded RL reward function designed by experienced human experts for a learning agent. Hester *et al.* [31] proposed a framework of the *deep Q-learning from demonstrations* (DQfD) and Ibraz *et al.* [13] trained a deep neural network to model the reward function by using expert demonstrations and real-time expert feedback and then used its predicted rewards to train a DQN. In our work, however, we do not need any professional demonstrations and feedback from experienced people; we just use the agent's self-demonstrations to shape the rewards in a real-time manner instead.

III. Preliminaries

A. *k-Nearest Neighbors Algorithm*

k-NN [32] is a simple algorithm that stores all the available data that are already classified and then classifies the new data into one of the classes using a similarity measure (e.g., distance functions, such as the Euclidean distance and Manhattan distance) based on the class labels of k neighbors. k-NN is also used for regression by calculating its average value. Parameter 'k' in k-NN refers to the number of nearest neighbors to include in the majority of the voting processes. The advantage of the k-nearest neighbors algorithm is that it is simple to implement and has high accuracy, but its accuracy is greatly affected by the selection of parameter k. It also needs to perform a great number of calculations when there is a large amount of sample data in memory.

B. *Potential-based Reward Shaping (PBRS)*

Reward-shaping function $F : S \times A \times S \to \mathbb{R}$ is used to modify original reward function R to make the RL methods (e.g., Q-learning) converge faster using more "instructive" rewards. It usually resides in the same functional space as the reward function and transforms the original *Markov decision process* (MDP), *MDP*(S, A, T, γ, R), into another shaped *MDP*$(S, A, T, \gamma, R^{'})$, where $R^{'} = R + F$. In the MDP, S is the state space, A is the action space, T is the state transition probability, γ is the discount factor, and R is the original reward function, i.e., the reward from the environment. Of all possible shaping functions, the potential-based shaping functions [5] lead to the optimal policy, as summarized below.

$$R^{'} = R + F(S, A, S^{'}), \tag{1}$$

$$F(S, A, S^{'}) = \gamma\Phi(s) - \Phi(s^{'}). \tag{2}$$

$\Phi(s)$ is the potential function that theoretically can be an arbitrary function that represents the state reward correctly. Generally, it is suggested to use the optimized value of $s \in S$, $V^{*}(s)$, where

$$V^{*}(s) = max_{a \in A} Q^{*}(s, a). \tag{3}$$

Therefore, the real-time reward is updated by: $R^{'} = R + \gamma V^{*}(s) - V^{*}(s^{'})$.

It is possible that an agent will adjust the Q-value effectively and adapt the training environment quickly by making use of the shaped reward. The PBRS is a simple method to accelerate the process of the RL but is not used in the DRL; in contrast to the Q-table based method, the DRL uses the neural network with randomly initialized weight and bias to approximate the Q-values. Therefore, the shaped reward may be inappropriate before adapting it sufficiently because the output from the network may be incorrect; thus, if it is used to update the real-time reward, this may result in a failure of learning or slow learning, especially with a long horizon or sparse reward, which needs a long time to adjust the parameter values of the network. In this paper, we attempt to make the PBRS method more effective in the DRL by using our proposed method.

C. *Deep Reinforcement Learning*

Deep-reinforcement learning (DRL) [15] applies the deep learning technique for RL to decide tasks to do next in a complex environment. The most popular and powerful RL methods are Q-learning and actor-critic (AC).

1) Actor-Critic: AC is an efficient RL method that combines value-based methods and policy-based methods. In the AC method based on neural networks, the policy network θ plays the role of an actor that selects actions (policy-based), and the value network ϖ plays the role of a critic that measures how good each action taken is (value-based). In the *advantage* *actor-critic* (A2C) [33] method in the DRL, the policy and the values are updated as follows:

$$\Delta\theta = \alpha * (Q_{\varpi}(s,a) - V_{\varpi}) * \bigtriangledown_{\theta} log\pi_{\theta}(s,a),$$

$$\Delta\varpi = \beta * (R(s,a) + \gamma V_{\varpi}(s^{'}) - V_{\varpi}(s)) * \bigtriangledown_{\varpi} V_{\varpi}(s),$$

where α and β are the learning rates, and γ is the discount rate. $V_{\varpi}(s)$ is the output from the value network and $Q_{\varpi}(s,a) = R(s,a) + \gamma V^{*}(s^{'})$, where $s^{'}$ is the next state. $\pi_{\theta}(s,a)$ is a value function, measuring how good action a is in situation s.

2) Deep Q-learning: DQN [16] uses a neural network to approximate the Q-values, and the agent chooses the action whose Q-value is the highest (value-based). With *experience replay* and a *fixed target Q-network*, the DQN exhibited good performances in various domains, and it may even outperform human behaviors. In the DQN, the neural network θ_t at time t is updated to minimize the loss function:

$$\mathcal{L}(s,a|\theta_t) = (r + \gamma max_{a\in A} Q(s^{'},a|\theta_t) - Q(s,a|\theta_t))^2,$$

where r is a reward from the environment. Then, network θ_{t+1} is updated for the next time:

$$\theta_{t+1} = \theta_t + \alpha \bigtriangledown_{\theta} \mathcal{L}(\theta_t),$$

where α is the learning rate, and γ is the discount rate.

IV. Proposed Method

People often use their experiences or the contents of their memory to compare the current situation with similar situations in their memory to learn/decide what action is beneficial and thus will do it next. In our studies, the agent retains the sequence of actions that led to a desirable result in an episode as trajectory data $\{(a_0, a_1 .... a_L)\}$, where L is the length of the sequence and is also the time to finish the task or the length of the episode. When $L \geq L_{long}$, where L_{long} is a positive integer, we define it as a long horizon task (we set it to 100 in our experiments below). Then, these desirable data are stored in memory pool $\mathcal{D}$ in the agent.

We attempt to know the situation of the agent from the current trajectory data, so, inspired by k-NN, we propose a method to find similar situations by comparing the distance between the current trajectory data of the agent and the data in memory pool $\mathcal{D}$. By imitating a human decision, if the agent's current trajectory is close to one of good data in memory pool $\mathcal{D}$, we attempt to give a more positive prediction reward for the agent; otherwise, a smaller negative prediction reward will be given. By combining our prediction method with the PBRS to fix the output of the neural network, we propose a meta-reward model to predict the training reward in a real-time manner.

Figure 1 shows the view of the trajectory action meta-reward model (TAMRM) to update the rewards of the current action for the agent. In our reward model, we make use of the Q-value of the agent's networks to calculate $F(S, A, S^{'})$ with Formulae (2) and (3) and take advantage of the action chosen by the agent to calculate the value of the k-NN-based reward prediction method below. Finally, we use these two values to shape the original reward to make the agent's learning faster.

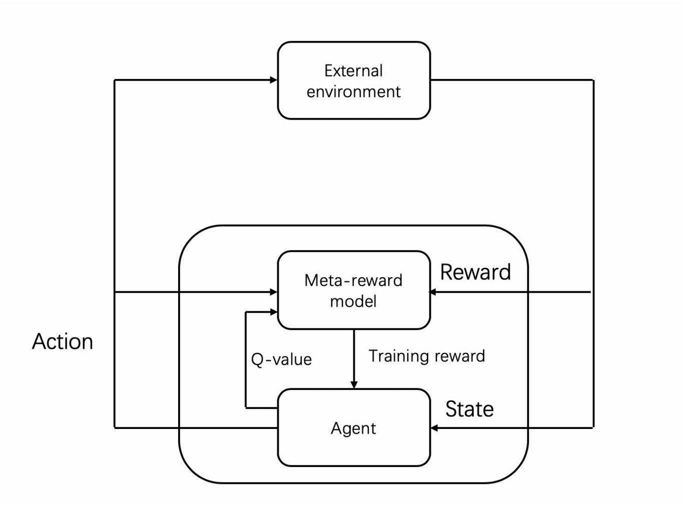

Fig. 1. Trajectory action meta-reward model.

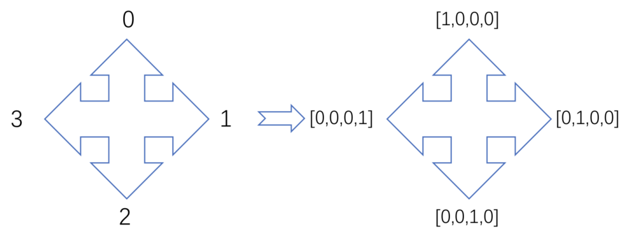

Fig. 2. Example of encoding the up, right, down, and left actions.

A. k-NN-based reward prediction

We store the pair of the trajectory data and associated total reward $d_i = \{(a_0^i, a_1^i .... a_{L_i}^i, R_i)\}$ of the agent in each episode (*episode reward*) in memory pool $\mathcal{D}$ to predict the rewards for future tasks, where R_i is the total reward calculated using only the original rewards from the environment. Note that we choose the best data, which has the highest episode reward, from memory pool $\mathcal{D}$. When the memory pool is full and the episode reward of the new trajectory data is greater than the minimum episode reward in the memory pool, the trajectory data with the minimum episode reward is replaced with the new data so that the data in the pool can be guaranteed to be the best historical trajectory data.

In this prediction method, first, we will compute the distance between trajectory data $d_i \in \mathcal{D}$ and current trajectory data d at each time t in the training stage as follows:

$$dis_i = \|d_i - d\| = \sum_{j=0}^{t} |a_j - a_j^{'}|^2, \quad N \leq t \leq L_{min}$$

where a_j and $a_j^{'}$ are the j-th actions in d_i and d, t is the current time, L_{min} is the minimum length of trajectory data in memory pool $\mathcal{D}$, and $N > 0$ is the start threshold.

To compute the distance between actions $|a_j - a_j^{'}|$, we use *one-hot encoding* to encode the action space, as shown in Fig. 2. Note that the length of the trajectory data in the memory pool may be different, so in this case, we set the minimum length (L_{min}) of the trajectory data in the memory pool to the end threshold. If current time t is greater than L_{min} and the episode does not terminate, the reward prediction will be stopped, and the agent will use the original reward from the environment; then, it continues training. Note that to avoid over imitating the historical trajectory data, by setting an integer $N > 0$ in the training process, the original environment reward will be used to train the agent before the N timestep, and after N timesteps, we start to predict the reward for the agent. If the episode stops quickly within several timesteps or the length of the episode is short, we have to set N to a small value.

Second, based on distance dis_i, we calculate probabilities P_i with all trajectory data $\forall d_i \in \mathcal{D}$.

$$Rate_i = \frac{\sum_{d_i \in \mathcal{D}} dis_i}{dis_i}$$

$$P_i = \frac{Rate_i}{\sum_{d_i \in \mathcal{D}} Rate_i}$$

These probabilities indicate how similar the current agent's situation (expressed by the current trajectory) is to the trajectory data in the memory pool.

Then, we calculate real-time rewards $r(t)$ for the current action of the agent:

$$x_i = \frac{R_i - min_{i \in D} R_i}{max_{i \in D} R_i - min_{i \in D} R_i}$$

$$f(t) = \sum_{i \in \mathcal{D}}^{k} x_i P_i \tag{4}$$

$$r(t) = e^{f(t)} - e^{\frac{1}{2}}, \tag{5}$$

where x_i is the normalized episode reward of trajectory data d_i in $\mathcal{D}$, so $x_i \in [0, 1]$. Note that in Formula (4), similar to k-NN, we set the k best values to choose the k largest probabilities $P_1, P_2, ...P_k$ and normalize episode rewards x_i to calculate $f(t)$. k cannot be greater than the size of $\mathcal{D}$, so $1 \leq k \leq |\mathcal{D}|$. By using Formula (5), we can calculate the predicted reward based on the agent's self-demonstration data in a real-time manner and use it in our reward model in the next part.

B. Meta-reward model

By combining the k-NN-based reward prediction and the PBRS, we propose the meta-reward model, TAMRM, to tune the training rewards R_{meta} for the agent calculated by:

$$R_{meta} = R + \alpha * r(t) + \beta * F(s, a, s^{'}),$$

where R is the original reward from the environment, $\alpha and \beta$ are the reference rates, and $F(s, a, s^{'})$ is defined by taking into account Formulae (2) and (3) as follows:

$$F(s, a, s^{'}) = \gamma V^*(s^{'}) - V^*(s).$$

With the proposed k-NN-based reward prediction method, we can obtain the predicted rewards from successful experience data to make the reward easier and the learning faster. By using the PBRS, we can adjust the output values from the neural network effectively. The learning procedure of reinforcement learning based on this model is shown in Algorithm 1. Unlike with off-policy DRL (e.g., DQN) in the on-policy DRL (e.g., AC and A2C), we can use the obtained shaped reward directly to train the agent using the experience data from the replay buffer, not samples.

Algorithm 1 Off-policy DRL with meta-reward model

```
Initialize neural network and replay buffer $\mathcal{B}$
Initialize meta-reward model and pool $\mathcal{D}$
for each task do
  if memory pool is not full then
    run the agent under policy $\varepsilon$-greedy and store experienced data $\{s_t, a_t, R, s_{t+1}\}$ in replay buffer $\mathcal{B}$
    update network with experience samples from $\mathcal{B}$ by gradient descent
  else
    run the agent under policy $\varepsilon$-greedy
    if $timesteps < N$ then
      store experience data $\{s_t, a_t, R, s_{t+1}\}$ in $\mathcal{B}$
    else
      store shaped data $\{s_t, a_t, R_{meta}, s_{t+1}\}$ in $\mathcal{B}$
    end if
    update network with experience samples from $\mathcal{B}$ by gradient descent if off-policy learning.
  end if
  store the trajectory data$\{(a_0, a_1....a_t, R_i)\}$ in $\mathcal{D}$.
end for
```

Note that in our model, one task corresponds to one memory pool $\mathcal{D}$. When the agent switches to a new task, a new pool will be built, and the agent will store the new task's trajectory data in this new pool. In the training process, when the initial state or the destination of the task is to be changed, we define that the agent has started to perform a new task.

V. Experiments and evaluation

We evaluated our method using three standard *deep reinforcement learning* environments (the cart pole, mountain car, and maze problems) and compared the experimental results using our meta-reward model with those using the conventional methods (deep Q-learning and A2C) using PBRS. Note that these neural networks have the same parameters and structures. We used the OpenAI's *Gym* framework and *Tkinter* to build up our test environments

A. Experiment 1— Cart pole problem

The cart pole problem (Fig. 3) is a classic discrete action problem in reinforcement learning. It has a dense reward: a reward of +1 is provided for every timestep that the pole remains upright, and a reward of -1 is provided when the pole is at an angle of more than 15 degrees from the vertical. It usually has a long horizon but has a short horizon if the episode ends negatively when the pole is more than 15 degrees from the vertical or the cart moves more than 2.4 units from the center. Because the original rewards of the cart pole problem do not distinguish between the desired situation (i.e., keeping the pole in an upright position) and dangerous ones (i.e. where the pole is just about to fall), it is inefficient to learn to solve this problem. By using OpenAI's environment Gym to train the agent, the observation of the cart pole problem is represented by the array that expresses the position and velocity of the cart and pole. Then, we use this array as the input to train the neural network. The output of the network is the action of *left* or *right* to control the car.

The basic setting for the experiment is as follows. An episode terminated after 200 timesteps; thus, the maximum episode reward was 200. The cart pole problem defines the "solution" as getting an average reward of 195.0 over 100 consecutive trials. The agent was trained using the A2C. We used k-NN with $k = 20$ and set the size of the memory pool to $|\mathcal{D}| = 30$ and set the reference rates to $\alpha and \beta = 0.8$. We used an MLP with two hidden layers of size 20 for the A2C.

The results of the cart pole problem are shown in Fig. 4. By using the meta-reward model, the agent could solve the problem in the 455th episode, which is faster than other methods because the TAMRM evaluated the reward for current actions using data in the memory pool that made the agent's action closer to that of the trajectory data of the high episode reward history. However, for the PBRS, before converging the neural networks, the output of the networks might not be accurate enough, and if we used these inaccurate values to shape the rewards, it might mislead the agent. Both the A2C and A2C+PBRS could not solve the task within 1000 episodes because the learning was not so easy with the original reward scheme; therefore, we need a more tailored reward scheme so that the agent can learn the appropriate actions, and our proposed method can be one of the probable methods.

B. Experiment 2 — Mountain car problem

Similar to the cart pole problem, the mountain car problem (Fig. 5) has a dense reward. However, it usually has a long horizon because an episode will terminate only when the car arrives at the goal at the top of the mountain, so the length of an episode is likely to be long in the beginning. In the training process, the environment reports the position and velocity of the car as observational data, and they are fed to the neural network. The output of the network is the action of *push left*, *no push*, or *push right* to control the car to the goal.

In this environment, we used the deep Q-learning to test our proposed method in off-policy DRL. We set the size of the memory pool in the meta-reward model to 20, α, β = 1, and $k = 20$. Note that in the mountain car problem, the fewer the steps to arrive at the goal, the higher the episode reward is. For the simple DQN, we use an MLP

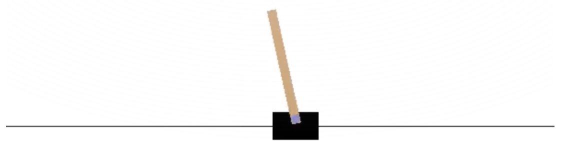
Fig. 3. Cart pole problem.

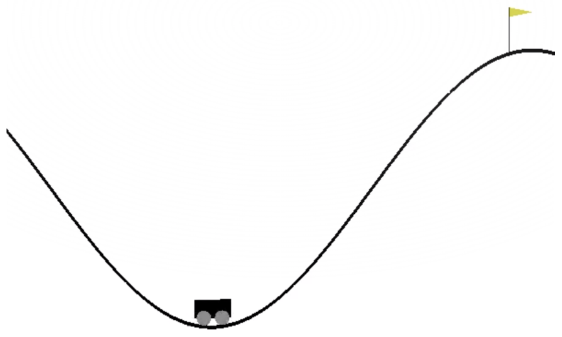
Fig. 5. Mountain car problem.

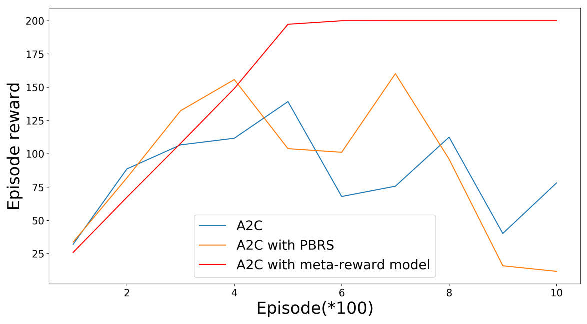

Fig. 4. Results of cart pole problem.

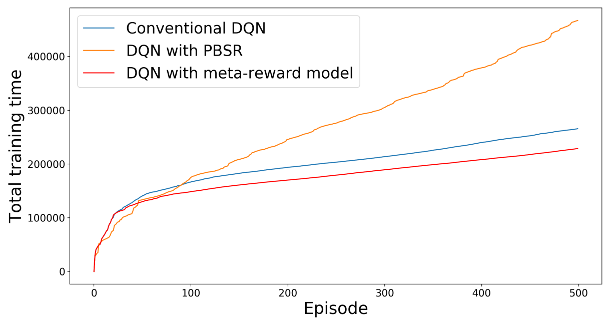

Fig. 6. Results of mountain car task.

with two hidden layers of size 20, and the initial epsilon value was set to 0.1. The decay rate was 0.9, replay buffer size was 50,000, and learning rate was 0.005.

In the long horizon environment, as indicated by the results of the experiment shown in Fig. 6, the DRL agent needed more time to fix the output of neural networks. Therefore, if we used the incorrect value of the network, the PBRS made it more difficult for the agent to converge Q-values. With the TAMRM, the agent could retrieve the history of the data in the memory pool and make the agent's behavior change direction toward successful self-demonstrations; thus, the network could converge faster than the conventional methods. These experimental results indicated that our method was also useful in the off-policy RL.

C. Experiment 3 — Maze problem

The maze problem is a standard long horizon and sparse reward environment for training a DRL agent. We build an 8×8 maze in which the red mark is the start point and the yellow mark is the end point; the black marks are the walls, as shown in Fig. 7. Before arriving at the goal, the reward from the environment is 0, and when the agent gets to the goal, it can get a reward of +1. Maze search failure is defined when an agent cannot arrive at the goal within 10,000 steps.

In this environment, the DQNs of the agents have an MLP with two hidden layers of size 10, replay buffer size was 20,000, and epsilon was 0.1. The decay rate was 0.9, and the learning rate was 0.01. We used the agent's coordinate data in this maze as the input to the networks of agents, and the network output is the action of *up*, *down*, *left*, or *right* to control the agent.

1) Static structure maze: In the first test, the size of the memory pool and k in the meta-reward model were 10 and $\alpha = 0.8, \beta = 0.3$ (for a sparse reward environment, it is suggested to set β and memory size to small values). The maze environment is shown in Fig. 7.

As indicated by the results of this experiment shown in Fig. 8 and listed in Table I, the agent that used the DQN with the PBRS found it difficult to solve the maze problem because its rewards were sparse since the rewards were 0 before arriving at the goal. When the rewards were 0, the networks were hard to converge, and the PBRS with the output value of networks estimated the training reward incorrectly, so the PBRS agent performed poorly in this task. On the other hand, using the TAMRM, the agent could reduce the 0 training reward and update the training rewards in every step by referring to the historical trajectory that successfully got to the goal. The agent with the conventional DQN could also exhibit good performance but lower performance than that of the agents using the TAMRM.

2) Modified maze: In this experiment, we set a new situation in the maze that after 100 episodes, the start point was changed, as shown in Fig. 9. We used the same settings of the DQN and the meta-reward model. Then, we compared the running results with our proposed method with those of

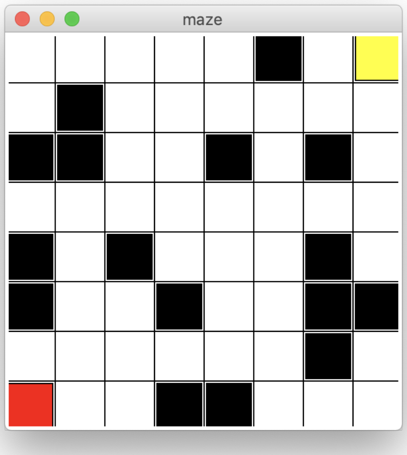

Fig. 7. The maze environment.

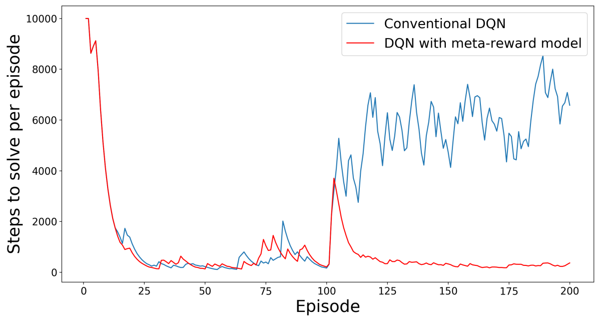

Fig. 8. Results of the Maze problem

TABLE I
TOTAL NUMBER OF STEPS COMPARISON IN MAZE

Used network	Total steps in 200 episodes
DQN with meta-reward model	127478
Conventional DQN	137599
DQN with PBRS	1433215

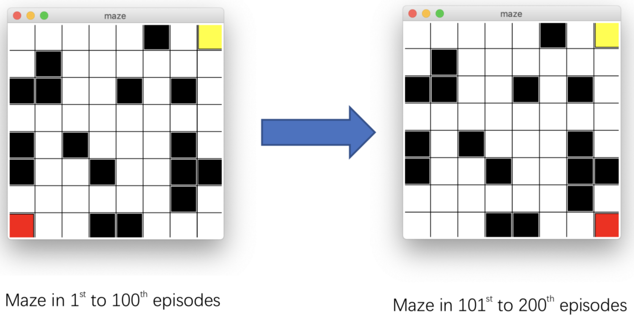
maze
maze
Maze in 1st to 100th episodes
Maze in 101st to 200th episodes

Fig. 9. Modified structure in the maze.

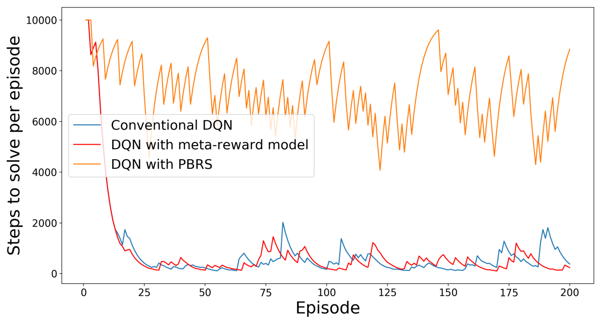

Fig. 10. Results of modified structure in the Maze problem.

the conventional DQN because the difference of performances with these methods was small.

After the 100th episode, because the place of the start point was changed, the agent had to learn to solve a new task. As indicated by the results shown in Fig. 10, the conventional DQN was affected by the previous tasks, and because the agent received 0 rewards before arriving at the goal, the agent needed more time to correct its behavior for the new start point. In our reward model, after starting to perform the new task, the agent could build a new memory pool to record the new trajectory data to shape the reward, instead of using the old pool. Through shaping the reward, we could reduce the effects of the previous task and make the agent focus more on the new task so that the neural network converged fast even for the modified maze.

D. Discussion

The original reward from the training environment may be correct, but it is often difficult for the agent to learn because the original reward structure usually cannot express the current situation correctly or cannot navigate the course of the correct actions effectively; thus, the agent can be unaware of the subsequent rewards, good or bad. In our proposed reward model, based on the original environment rewards, we used historical trajectory data of the agent to try to make the rewards for each action more sensitive to good trajectory data during the training process. For example, in the cart pole experiment, unlike the original reward that gives +1 every time and so may make the agent think that all actions are good, our reward model tried to shape the rewards to make the agent recognize the current actions that are possible to get better or worse results in the future; thereby, the use of our method could solve the cart pole problem fast and effectively. Similar to the cart pole problem, the agent in the mountain car problem could recognize how fast to arrive at the goal by giving different rewards for the different actions by referring to the agent's self-experience using our model. In the maze experiment, the original rewards from the environment were sparse, but our proposed meta-reward model changed the sparse reward structure by adding a few rewards to the zero-reward actions and thus the obtained reward structure could navigate the agent to the goal faster. Using our shaped rewards, we can also refresh the output structure of the network quickly for the new task, just like the maze experiment with the modified structure. Thus, our reward model had better performance in the experiment above by shaping the rewards.

The meta-reward model based on successful trajectory data is useful to accelerate the reinforcement learning process. We also assume that problem environments are almost stable. Therefore, if the environment varies, the agent is unable to generate successful data for shaping reward.

VI. Conclusion

In this paper, we proposed a new reward shaping method, TAMRM, which is simple to implement and uses only the historical trajectory data of itself to automatically shape the training reward in real time to make the agent learn efficiently and effectively by giving good rewards. It does not need to expand the neural network structure [34] and does not need to calculate the extra gradients backpropagated to fix the PBRS [14]; thus, it does not need a great amount of computation resources. Therefore, it has good performance in both dense and spare reward environments. In addition, it has better performance in on-policy reinforcement learning. Theoretically, it can be used in all types of reinforcement learning methods based on reward.

Because we limited the *one-hot encoding* to encode the action space and have to calculate the distance between sequences of actions, our method is effective with a discrete action space, such as simple motions of agents, but we need to extend our method to apply it to other types of applications. In future work, we will attempt to extend the current method to make it applicable to other problems by using other encoding methods. We also intend to shape the reward for the DRL agent to solve the problems in a continuous action space and try to train the agent to play in a video game.

VII. Acknowledgment

This work is supported by JSPS KAKENHI Grant Number 17KT0044 and 20H04245.

References

[1] J. Vanschoren, "Meta-learning: A survey," *arXiv preprint arXiv:1810.03548*, 2018.

[2] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International conference on machine learning*, 2016, pp. 1842–1850.

[3] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," in *Advances in neural information processing systems*, 2016, pp. 3630–3638.

[4] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1126–1135.

[5] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[6] A. Eck, L.-K. Soh, S. Devlin, and D. Kudenko, "Potential-based reward shaping for POMDPs," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 2013, pp. 1123–1124.

[7] B. Badnava and N. Mozayani, “A new potential-based reward shaping for reinforcement learning agent,” *arXiv preprint arXiv:1902.06239*, 2019.

[8] M. Grześ, “Reward shaping in episodic reinforcement learning,” in *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 565–573.

[9] W. Saunders, G. Sastry, A. Stuhlmueller, and O. Evans, “Trial without error: Towards safe reinforcement learning via human intervention,” in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2067–2069.

[10] W. B. Knox and P. Stone, “Reinforcement learning from simultaneous human and MDP reward.” in *AAMAS*, 2012, pp. 475–482.

[11] D. Abel, J. Salvatier, A. Stuhlmüller, and O. Evans, “Agent-agnostic human-in-the-loop reinforcement learning,” *arXiv preprint arXiv:1701.04079*, 2017.

[12] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, “Deep reinforcement learning from human preferences,” in *Advances in Neural Information Processing Systems*, 2017, pp. 4299–4307.

[13] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei, “Reward learning from human preferences and demonstrations in Atari,” in *Advances in Neural Information Processing Systems*, 2018, pp. 8011–8023.

[14] H. Zou, T. Ren, D. Yan, H. Su, and J. Zhu, “Reward shaping via meta-learning,” *arXiv preprint arXiv:1901.09330*, 2019.

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 2011.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[17] M. J. Mataric, “Reward functions for accelerated learning,” in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 181–189.

[18] A. D. Laud, “Theory and Application of Reward Shaping in Reinforcement Learning,” Ph.D. dissertation, University of Illinois at Urbana-Champaign, USA, 2004, aAI3130966.

[19] J. Asmuth, M. L. Littman, and R. Zinkov, “Potential-based Shaping in Model-based Reinforcement Learning.” in *AAAI*, 2008, pp. 604–609.

[20] A. Harutyunyan, T. Brys, P. Vrancx, and A. Nowé, “Off-policy shaping ensembles in reinforcement learning,” *arXiv preprint arXiv:1405.5358*, 2014.

[21] E. Wiewiora, G. W. Cottrell, and C. Elkan, “Principled methods for advising reinforcement learning agents,” in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 792–799.

[22] S. M. Devlin and D. Kudenko, “Dynamic potential-based reward shaping,” in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 2012, pp. 433–440.

[23] O. Marom and B. Rosman, “Belief reward shaping in reinforcement learning,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[24] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, “Reinforcement learning from demonstration through shaping,” in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[25] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, “Dueling network architectures for deep reinforcement learning,” *arXiv preprint arXiv:1511.06581*, 2015.

[26] A. Y. Ng, S. J. Russell *et al.*, “Algorithms for inverse reinforcement learning.” in *Icml*, vol. 1, 2000, p. 2.

[27] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, “Maximum Entropy Inverse Reinforcement Learning,” in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, ser. AAAI’08. AAAI Press, 2008, pp. 1433–1438.

[28] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[29] H. B. Suay, T. Brys, M. E. Taylor, and S. Chernova, “Learning from demonstration for shaping through inverse reinforcement learning,” in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 2016, pp. 429–437.

[30] Y. Wu and Y. Tian, “Training Agent for First-Person Shooter Game with Actor-Critic Curriculum Learning,” in *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=Hk3mPK5gg

[31] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, “Deep q-learning from demonstrations,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[32] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[33] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International conference on machine learning*, 2016, pp. 1928–1937.

[34] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, and J. Z. Leibo, “Learning to reinforcement learn,” in *CogSci*, 2017.