Master Thesis

Using reward shaping and cooperative asynchronous method to accelerate the training process of reinforcement learning

Xiaohui Zhu

A Thesis Submitted to the Department of Computer Science and Communications Engineering, the Graduate School of Fundamental Science and Engineering of Waseda University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Student ID	5118F116-6
Date of Submission	July 18th, 2020
Advisor	Prof. Toshiharu Sugawara
Research Guidance	Research on intelligent software

Contents

Abstract

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 demand for massive amounts of training data. Thus, it always needs a long time to train a DRL agent, especially when the reward of the environment is not easy to learn (sparse or original reward uncertainty). In this thesis, we proposed two methods to speed up the process of DRL. First, the reward shaping method, we present a reward model, trajectory action meta-reward model, to shape the training rewards in real-time for deep reinforcement learning of motions with discrete action space. Without using the expert data and having to hand-craft the reward model, our meta-reward model and reward shaping method use the combination of the agent's self-demonstrations and the potential-based reward shaping (PBRS) method, to make the neural networks converge faster in every task. Second, based on the asynchronous reinforcement learning (A3C), we introduced a learning method with using the cooperative multi-agent deep reinforcement learning settings. Through sharing the meaningful experience, not only the fail but also the success, we make the asynchronous agents in RL more cooperative so that reduce the training time of DRL. We experimentally evaluated our proposed methods and compared the performances of classic motion control problems in long-horizon environments, and in the video games. The result showed that our proposed methods could speed up the DRL in the classic control problems of an agent in various environments.

1 Introduction

In recent years, *deep reinforcement learning* (DRL(Mnih et al., 2015)) has achieved many impressive results, such as playing video game and robot domains. However, the learning process requires a huge number of trials, so it is necessary to provide massive amounts of training data for the agent 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 (Vanschoren, 2018; Santoro et al., 2016; Vinyals et al., 2016). For meta-learning in *reinforcement learning* (RL), one of the most popular algorithms is *model-agnostic meta-learning* (MAML) (Finn et al., 2017), 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 (Ng et al., 1999; Eck et al., 2013; Badnava and Mozayani, 2019; Grześ, 2017), 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 (Saunders et al., 2018; Knox and Stone, 2012; Abel et al., 2017; Christiano et al., 2017). Furthermore, Ibarz *et al.* (Ibarz et al., 2018) 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 (Xiaohui and Toshiharu, 2020)) based on *k-nearest neighbors* (k-NN) and PBRS for DRL to learn how to execute the given tasks efficiently by adjust the training reward. 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 (Zou et al., 2019). This model is versatile in the sense that it can be used in both the *actor-critic method* (AC method) (Sutton and Barto, 2011) and the *deep Q-learning* using the *deep Q-network* (DQN) (Mnih et al., 2015). 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.

On the other hand, *asynchronous reinforcement learning*(Mnih et al., 2016) is also a useful approach to accelerate the learning process by using the multi-core structure of the CPU. Unlike DQN, where a single agent represented by a single neural network interacts with a single environment, *asynchronous reinforcement learning* utilizes multiple incarnations in all the threads in order to learn more efficiently. In the *Multi-Agent Reinforcement Learning* (MARL), we know that if cooperation is done intelligently, each agent can benefit from other agents' training information, episodic experience, or learned knowledge. (Tan, 1993) Therefore, we proposed a method to make multiple incarnations solve problems cooperatively by sharing the experience. We also experimented this cooperative asynchronous learning method in classic motion control problems in long-horizon environments to accelerate to it. Furthermore, we show that combining the TAMRM and proposed cooperative asynchronous learning method can speed up the DRL, through experiment whose results were shown in section 5.

2 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 (Mataric, 1994; Laud, 2004). PBRS modifies the original reward function by using potential-based shaping functions $\Phi(s)$ to make RL methods (e.g., Q-learning) converge faster (Ng et al., 1999; Asmuth et al., 2008; Harutyunyan et al., 2014). PBRS usually depends on state s, but it has also been extended to *potential-based advice* (PBA) (Wiewiora et al., 2003) to include *action* a in potential function $\Phi(s, a)$. It has also been extended to *dynamic PBRS* (Devlin and Kudenko, 2012) by introducing *time* t into potential function $\Phi(s, t)$. Marom and Rosman (Marom and Rosman, 2018) 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 (Brys et al., 2015). Zou *et al.* (Zou et al., 2019) used a method, which is similar to MAML, to pre-train the *dueling deep Q-network* (dueling-DQN (Wang et al., 2015)) 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 (Ng et al., 2000; Ziebart et al., 2008; Abbeel and Ng, 2004; Suay et al., 2016). By combining the imitation learning and the DRL, Wu (Wu and Tian, 2017) created a hand-coded RL reward function designed by experienced human experts for a learning agent. Hester *et al.* (Hester et al., 2018) proposed a framework of the *deep Q-learning from demonstrations* (DQfD) and Ibraz *et al.* (Ibarz et al., 2018) 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.

In multi-agent reinforcement learning (MARL) framework, cooperative agents can accelerate the learning process with the teacher-student communication among the agents(Zhu et al., 2019)(Torrey and Taylor, 2013). A more experienced agent (teacher) can advise another agent (student) which action to take in a state. To make all the agents of RL play a role as teacher-student and cooperative, experience sharing is one of the useful

ways.(Tan, 1993) (Da Silva et al., 2017).

Moreover, not only in the multi-agent system, recently, experience sharing (ES) techniques had been used to homogeneous agents and non-dynamic environments(Souza et al., 2019) . In ES, each training agent will submit their problem to requests board in the episode stage, and then, in the sharing stage, each agent will check the problem on the requests board and share the successful experience with the corresponding agent, which made the RL agents play the role of learner or teacher to speed up the learning process. As the asynchronous reinforcement learning would create multiple homogeneous agents in all the threads, thus, in this thesis, we proposed a ES setting to make the agents in asynchronous reinforcement learning cooperative and further acceleration for DRL.

3 Preliminaries

3.1 k-Nearest Neighbors Algorithm

k-NN (Cover and Hart, 1967) 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.

3.2 Potential-based Reward Shaping (PBRS)

Reward-shaping function $F : S \times A \times S \rightarrow \mathbb{R}$ 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), $MDP(S, A, T, \gamma, R)$, 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 (Ng et al., 1999) 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.

3.3 Deep Reinforcement Learning

Deep-reinforcement learning (DRL) (Sutton and Barto, 2011) 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).

3.3.1 Deep Q-learning

DQN (Mnih et al., 2015) 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.

3.3.2 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) (Mnih et al., 2016) 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.

4 Proposed Method

4.1 trajectory action meta-reward model

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.

4.1.1 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

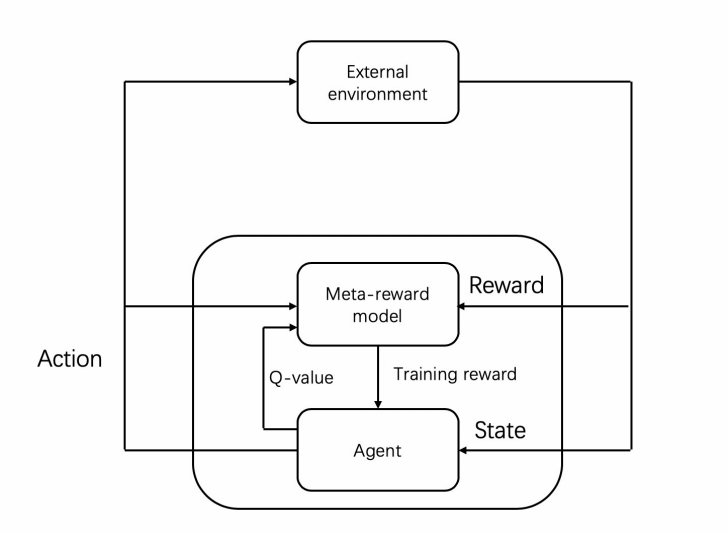

Figure 1: Trajectory action meta-reward model.

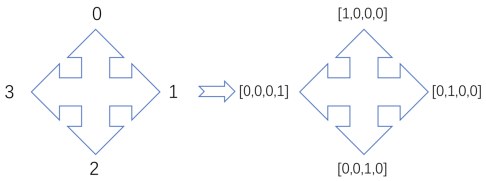

Figure 2: Example of encoding the up, right, down, and left actions.

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.

4.1.2 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.

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.

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 ε-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 ε-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

4.2 Full experience exchange

In asynchronous reinforcement learning, each agent has independent MDPs with similar goals. In our proposed method, experience is defined as a tuple $\{s_t, a_t, R, s_{t+1}\}$, representing a transition taken by an agent from one state to another and the response received

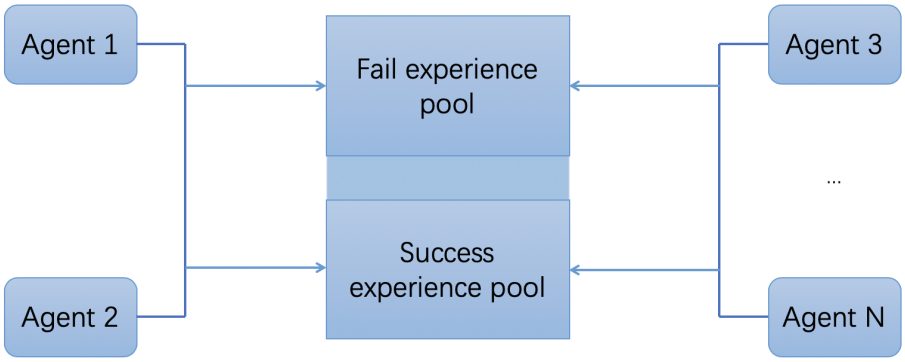

Figure 3: Full Experience Exchange-collection stage.

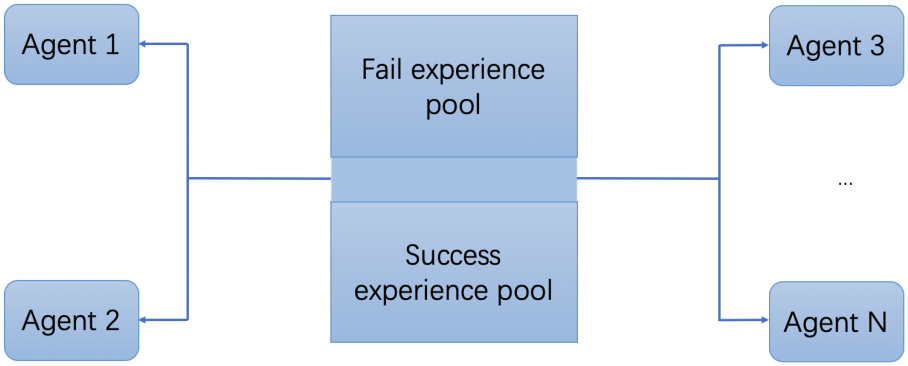

Figure 4: Full Experience Exchange-learning stage.

from the training environment. We try to make the agents in asynchronous reinforcement learning to solve a problem cooperatively by sharing the experience.

In this thesis, the baseline asynchronous reinforcement learning algorithm is Asynchronous Advantage Actor-Critic (A3C)(Mnih et al., 2016). Based on the global A2C network, in multi-threads, training agents will be created to learn to solve the problem in environment. Since each agent is created by the parameters of the global network. Therefore, the problems and successes that agents encounter in the training environment are common to all. Thus, sharing experiences from the training process, we believe, can effectively accelerate the training process.

Different from the approach of ES(Souza et al., 2019), we extending the vision of each agent that sharing all the meaningful experience with others, both success, and failure, called Full Experience Exchange (FEE) as shown in Algorithm 2. We divide this method

into two stages: collection and learning stage. In the collection stage, we will collect all the problems and success experiences among all the asynchronous agents to the global experiences pool. In the learning stage, each agent searches its experiences pool and submits the solved experiences to the global memory pool, and then the agent begins learning and update the networks. Each agent learns a policy independently from others, and interacts with them only for the purpose of sharing experiences. Meanwhile, for avoiding a great amount of communication cost, we limit the communication between the agents to once each episode. Through sharing the experiences, we make the asynchronous agents learn to solve the problem cooperatively.

Algorithm 2 Full Experience Exchange

```
Initialize agent and environment
Initialize fail experience memory pool DF and success experience memory pool DS
while in train process do
  for each agent in the environment do
    agent plays episode
    push the K batch size fail and success experiences to the DF, DS
  end for
  for each agent in the environment do
    agent check the fail experiences to recognize the problems in this episode
    if agent has success experiences corresponding the problem then
      push the experiences to the DS
    end if
    agent learns the experiences from DF and DS
  end for
end while
```

$\mathcal{K}$ is the most important parameter in our FEE method. To avoid a great amount of calculation, the k value can not be set too large. As the experience is defined as a tuple $\{s_t, a_t, R, s_{t+1}\}$, we share the k batch size experience $\{(s_{t-k}, a_{t-k}, R, s_{t-k+1}), ..., (s_t, a_t, R, s_{t+1})\}$ around all the agents in our FEE method.

5 Experiments

In this section, we divided our experiment section into three parts: the TAMRM part, the FEE part, the TAMRM+FEE part to evaluate our proposed methods, and used the OpenAI's *Gym* framework and *Tkinter* to build up our test environments.

5.1 Experiment for TAMRM

In this part, 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.

5.1.1 Cart pole problem

The cart pole problem (Fig. 5) 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. 6. 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,

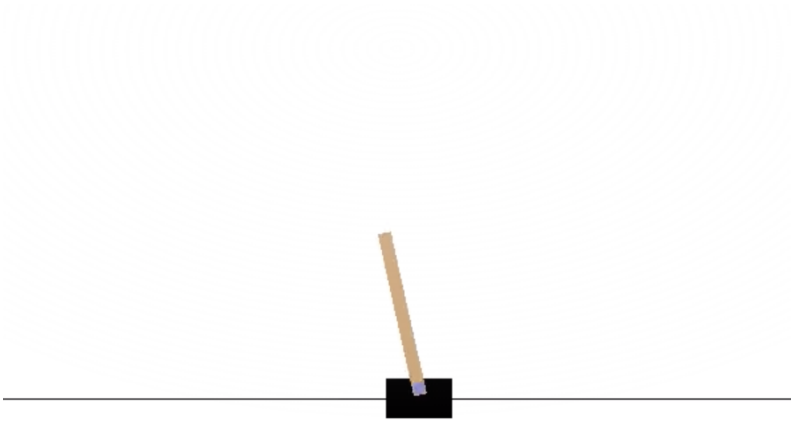

Figure 5: Cart pole problem.

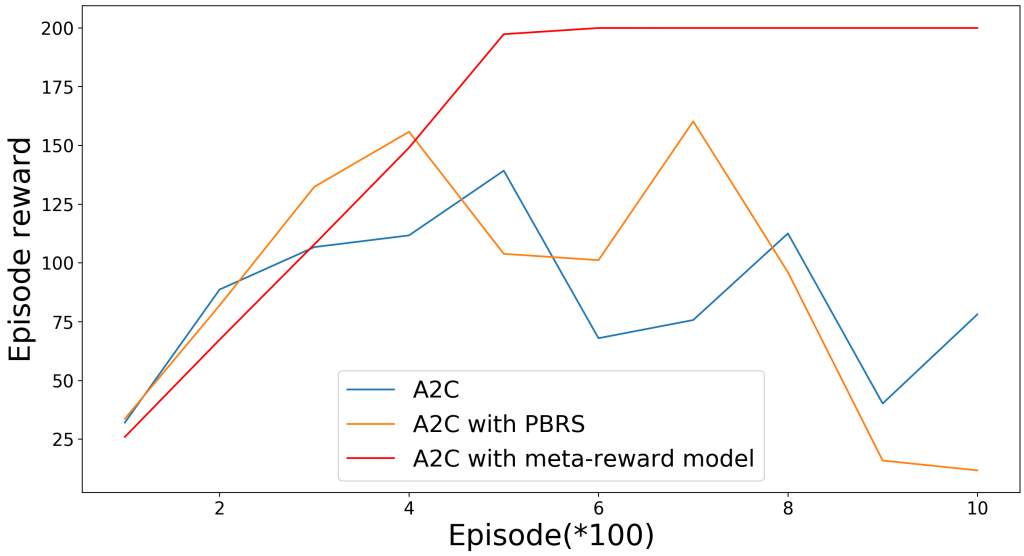

Figure 6: Results of cart pole problem.

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.

5.1.2 Mountain car problem

Similar to the cart pole problem, the mountain car problem (Fig. 7) 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.

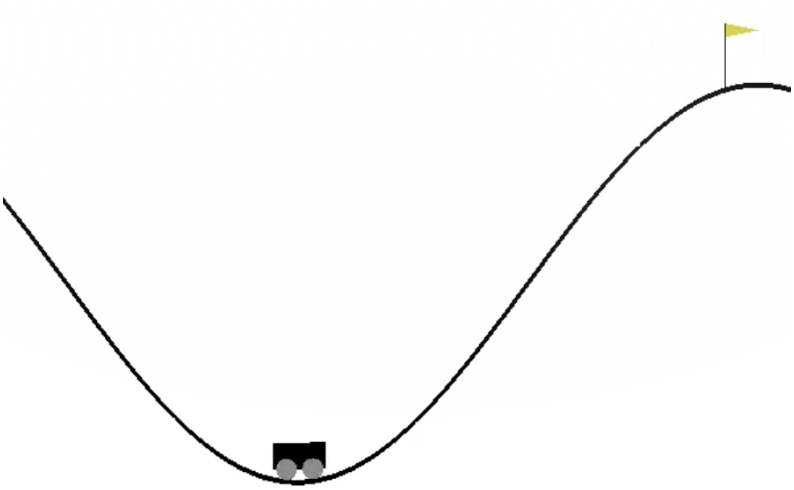

Figure 7: Mountain car problem.

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, $\alpha, \beta = 1$, 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 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. 8, 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-

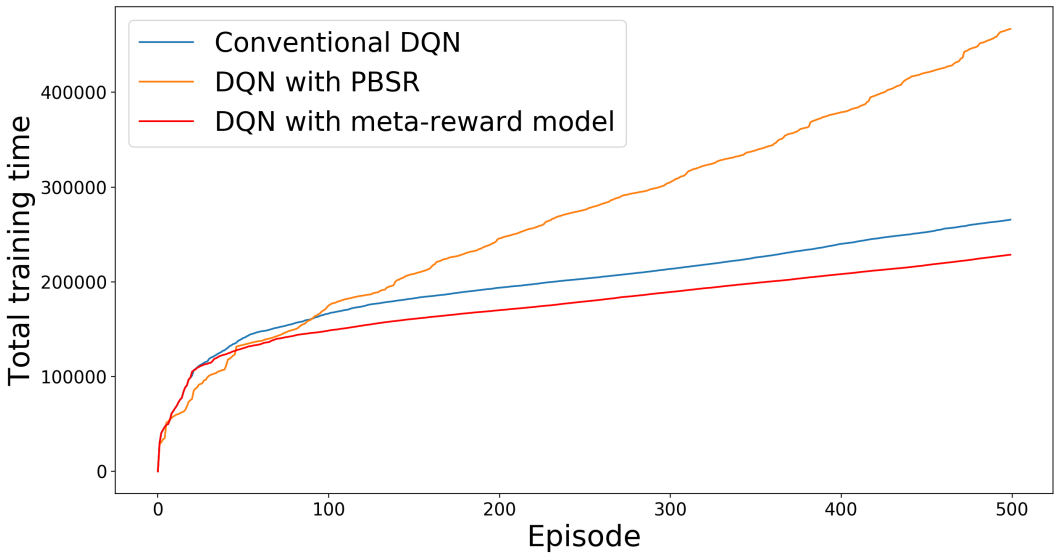

Figure 8: Results of mountain car task.

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.

5.1.3 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. 9. 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.

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

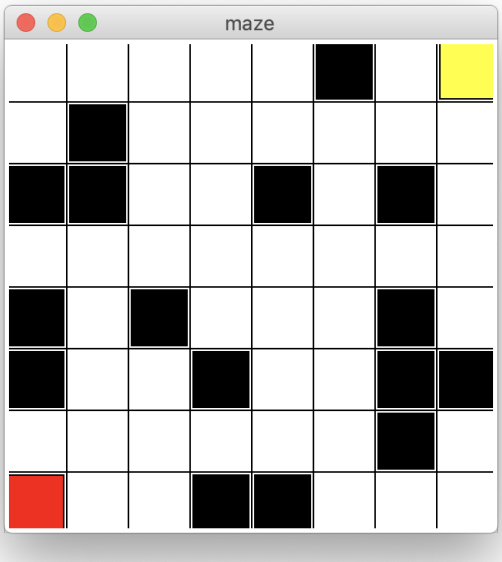

Figure 9: The maze environment.

Table 1: 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

suggested to set β and memory size to small values). The maze environment is shown in Fig. 9.

As indicated by the results of this experiment shown in Fig. 10 and listed in Table 1, 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.

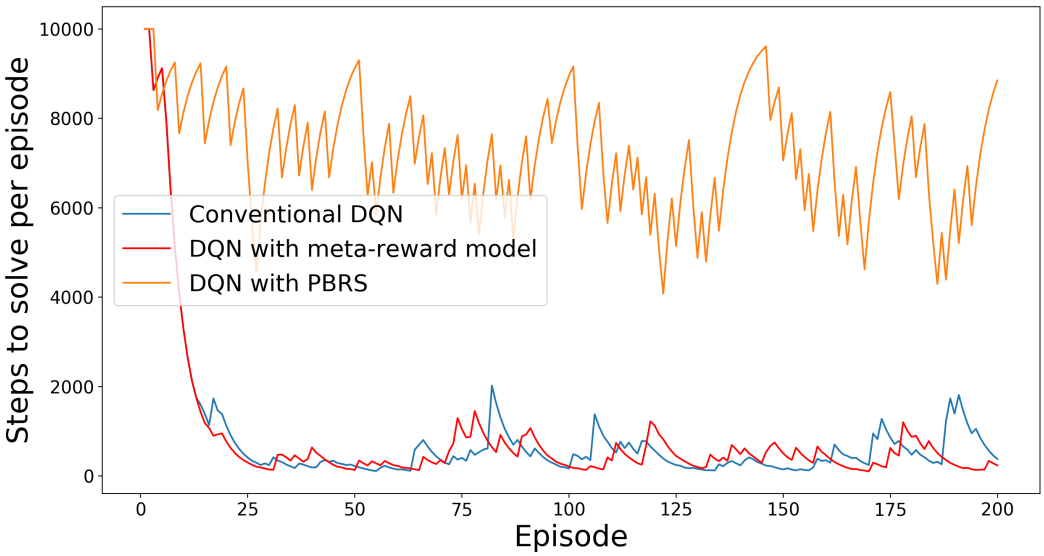

Figure 10: Results of static structure in the Maze problem.

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

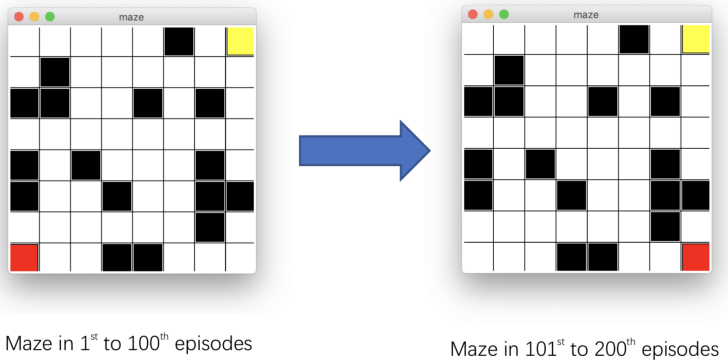

Figure 11: Modified structure in the maze.

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. 12, 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,

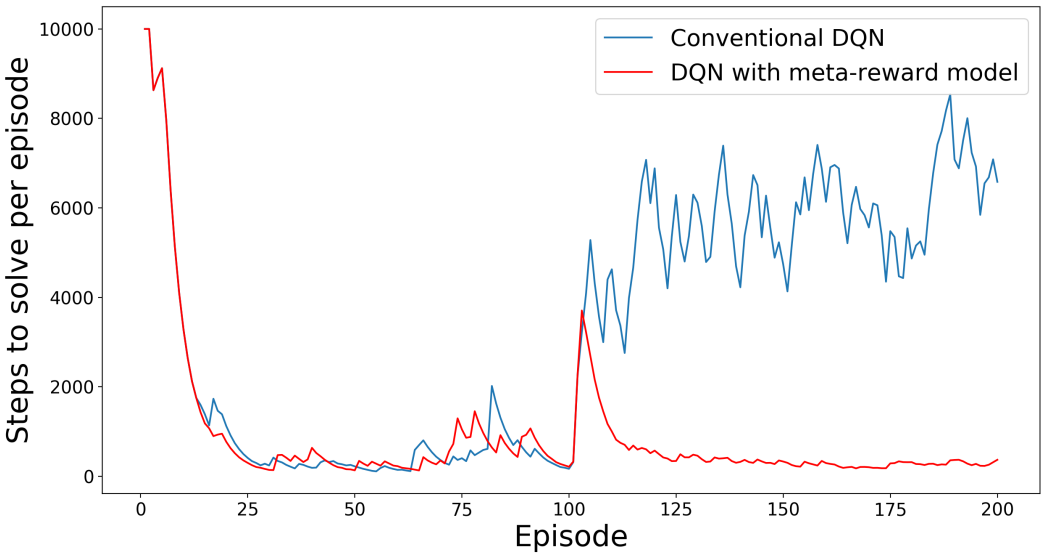

Figure 12: Results of modified structure in the Maze problem.

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.

5.1.4 Discussion for our reward model

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.

5.2 Experiment for FEE

In this part, we evaluated our FEE method using *Asynchronous Advantage Actor-Critic* (A3C)(Mnih et al., 2016) algorithm at classical control problem and compared the experiment result with ES. We train our asynchronous reinforcement agent with 12 threads, which means that 12 agents learn to solve the problem concurrently.

5.2.1 Cart pole problem

In the basic setting of FEE, we set the share memory size $\mathcal{K} = 5$ and we used an MLP with two hidden layers of size 100 for the global network structure for the *A3C*. In general, when getting a negative reward is failure experiences and getting a positive reward is success experiences. However, the CartPole problem had a dense structure that agents can get +1 positive reward at each timestep before the pole falling down. Therefore, in this experiment, we defined the negative reward as the fail experience and the successful experience corresponds to fail experience. For example, if there is the fail experience $(s_t, a_t, r_t, s_t + 1), r_t < 0$, the successful experience is $(s_t, a_t^{'}, r_t^{'}, s_t^{'} + 1), r_t^{'} >= 0$.

The experiment results are shown in Fig. 13. Our memory sharing method (red line) had a better performance in controlling the cart to prevent the pole from falling over. All the cart agents shared their successful and fail experience, and learn together so that all agents can realize and learn all the problems and solved ways as quickly as possible. For the ES method, the agent only learns its own problem and the solved experiences from other agents. Thus, the FEE method can solve this problem within 200 episodes that it is learning faster than the ES method.

5.2.2 Mountain car problem

In the basic setting of our FEE, we set the share memory size $\mathcal{K} = 15$ that we shared the 15 batch experiences about failure and success. In this problem, the car arrived at the

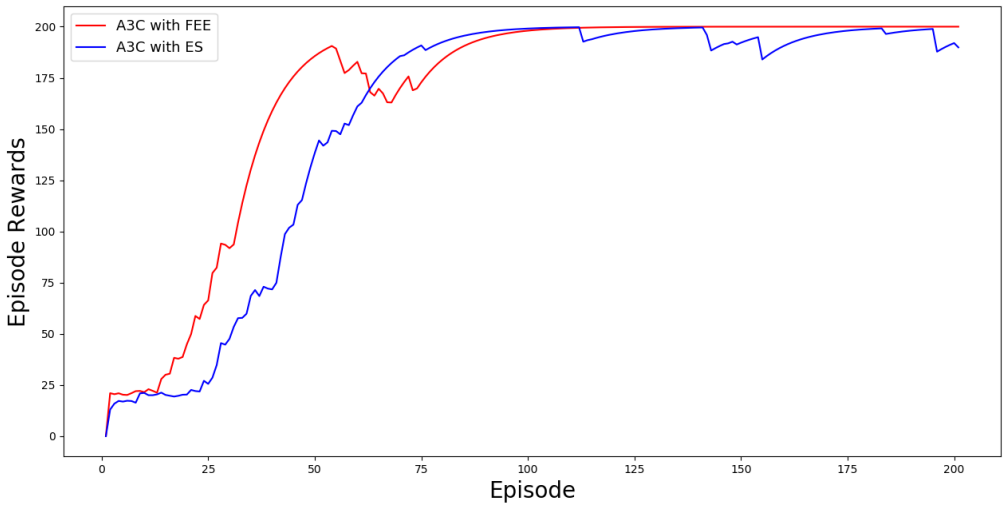

Figure 13: Result of CartPole problem with FEE and ES

mountain is a successful experience and fails to arrive at the mountain within 10000 steps is fail experience and the successful experience is the car get to the mountain that gets a positive reward.

As the result shows in Fig. 14, our proposed method spent less time to train the car to climb to the mountain. In this stable environment, various successful experiences (arrived at the mountain) of the asynchronous agent were shared with others and were learned repeatedly. Thus, the global network can convergence faster. In the ES method, the successful experiences between agents are not sufficiently shared, so that our method had a better performance in this problem.

5.2.3 Discussion for FEE

For the conventional A3C algorithm, each asynchronous agent individually learns the problem and updates the global network. Our FEE method makes the asynchronous agent become cooperative by sharing the meaningful(fail and success) experiences under the student-teacher role. Through the meaningful experiences flow among all the asynchronous agents, that experiences would be relearned and the agent can learn to solve the problem and avoid failure faster. For example, in CartPole problem, the various experience about the pole fell down of all the agents would be shared and learned so that the agent can realize the different types of fail experience as fast as possible and learn to avoid happening again. In Mountain car problem, all the agents shared their own experience of

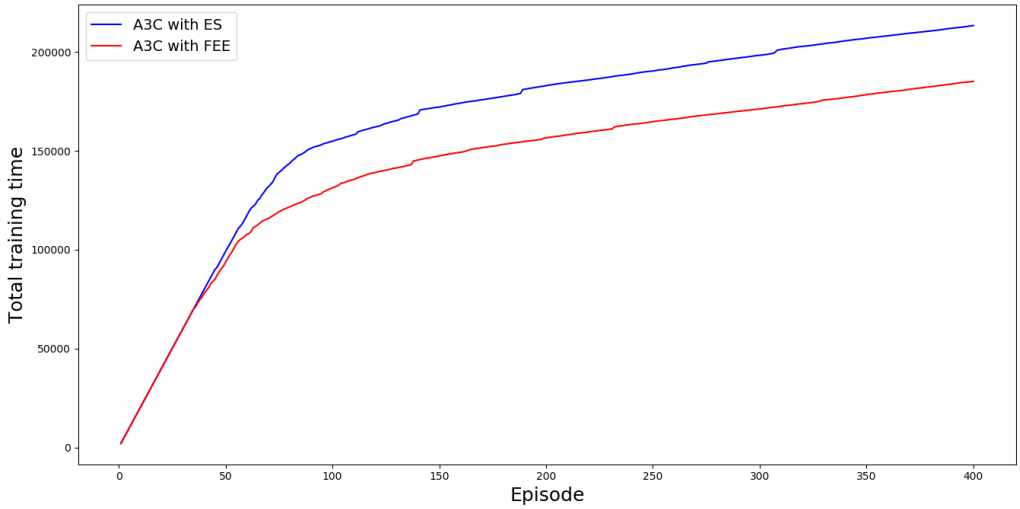

Figure 14: Result of Mountain car problem with FEE and ES

how to arrive at the top of the mountain to others. So, agents can understand and learn various successful experiences to get to the top faster. Thus, our FEE method effectively improves the learning speed by sharing the experiences which make all the agents learn cooperatively in the teacher-student role.

5.3 Experiment for TAMRM+FEE

In the section above, We demonstrate that our proposed TAMRM model and FEE methods are effective in accelerating the process of reinforcement learning. We assume that if we use the reward model and FEE method to train the agent at the same time, we can further accelerate the deep learning process of reinforcement learning. In this part, we experimented with the agent as the player to learn the motions of the bird to pass the pipe in the *Flappy Bird* video game.

Flappy Bird video game is a side-scrolling game where the agent must successfully navigate through gaps between pipes. The up arrow causes the bird to accelerate upwards. If the bird makes contact with the ground or pipes, or goes above the top of the screen, the game is over. For each pipe it passes through it gains a positive reward of +1. Each time a terminal state is reached it receives a negative reward of -1 and others times, it receives a 0 reward. Thus, it is a classical control problem with long-horizon and sparse environment.

In this experiment, the baseline asynchronous algorithm is *A3C*, and we set the

network structure as Fig. 15. Obviously, when get the negative reward is failure experience and when get the positive reward is success experience. In the FEE method, we set the $\mathcal{K} = 3$. In the meta-reward model, we set the size of the memory pool and k in the meta-reward model were 10 and $\alpha = 0.7, \beta = 0.1$. Note that, we use only one reward model, that all asynchronous agents share the same meta-reward model (Fig. 16). The red line shows the experiences sharing action among all the agents.

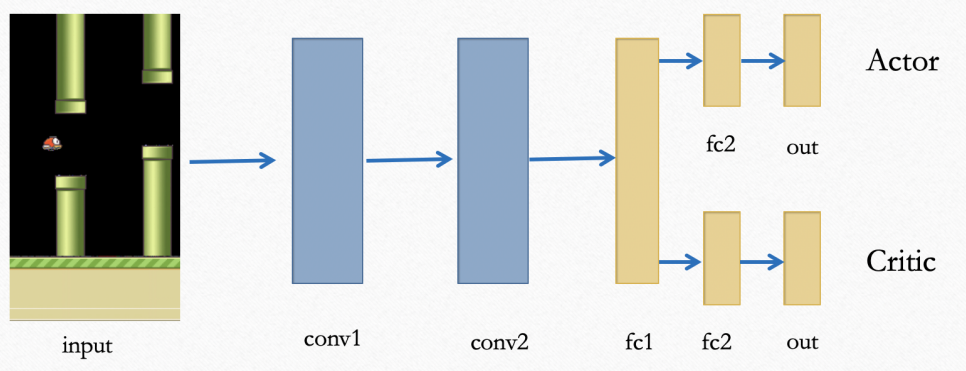

Figure 15: The network structure of Flappy Bird video game

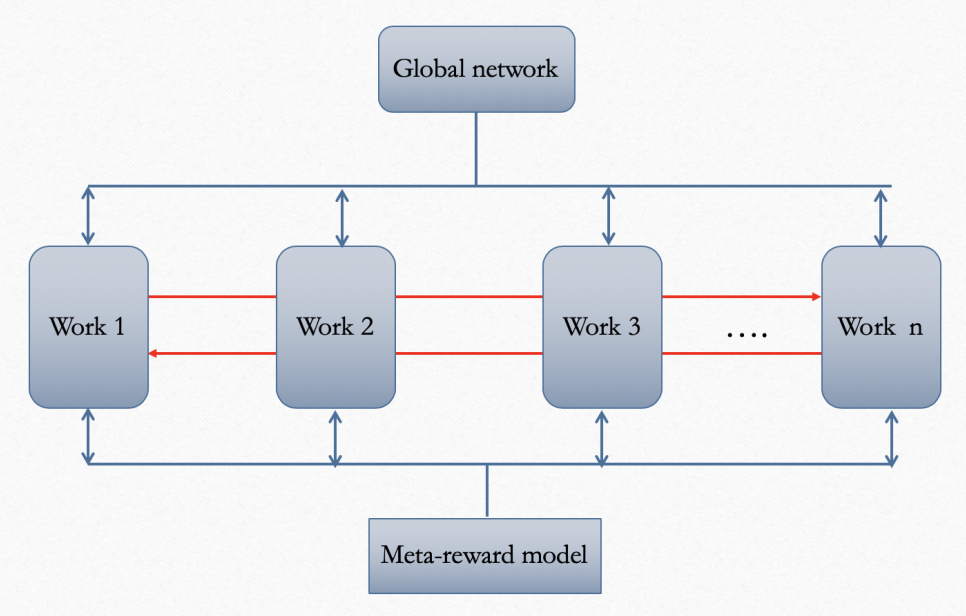

Figure 16: Training structure

As Fig. 17 shows, our accelerated method(TAMRM+FEE) can improve the learning speed of the conventional A3C method in a long-horizon, sparse environment. In this video game, the difficulty for the bird to learn is the 0 reward which made the agent could not know the situation was good for passing the pipe or was going to fail (terminal state). Through our meta-reward model, we reduced the happening of the 0 rewards and revised

the reward to try to predict whether going to pass the pipe or closing to the terminal state in the future by referring to the historical trajectory data. For the FEE, as all the asynchronous agents shared the experience to pass the pipe and the fail experience to the terminal state, all these meaningful experiences were relearned that made the agent easier to learn to pass the pipe and get the positive reward.

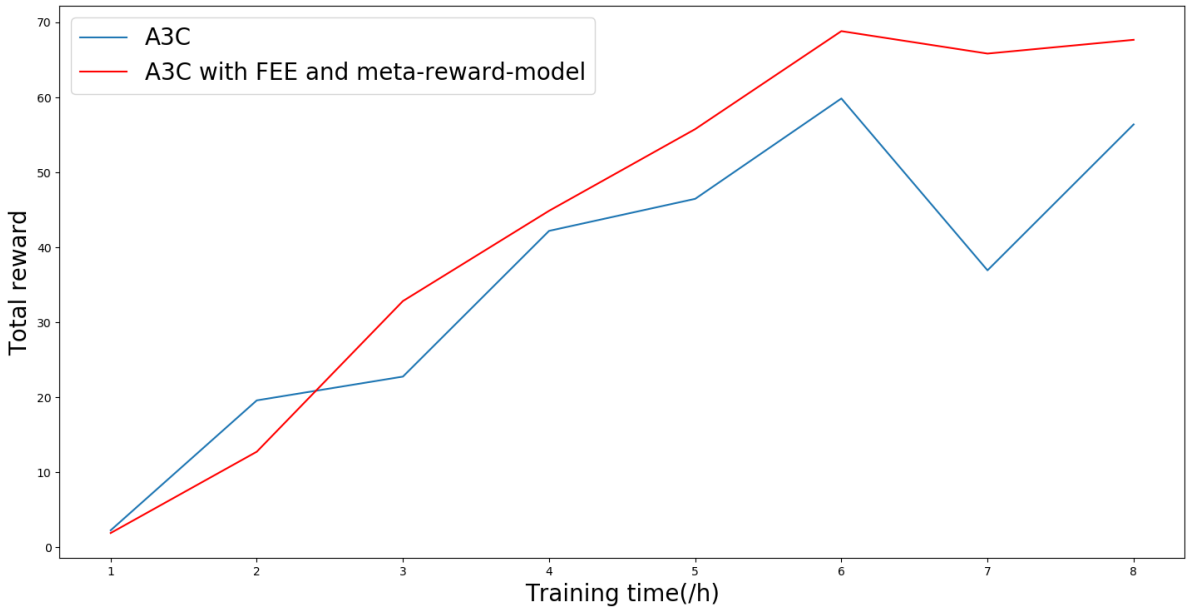

Figure 17: The result of Flappy Bird video game

5.4 Disscussion

In this section, we trained our method in discrete action space long-horizon environment with sparse or dense reward structure. We got the desired data and proved our proposed methods are effective to accelerate the deep reinforcement learning. The meta-reward model made the single-agent easier to learn with the revised reward and FEE made all asynchronous single-agents cooperatively learn to solve the problem that could save the training time. Meanwhile, we can train the agent with both TAMRMA and FEE that reward model to help the single-agent learn faster and FEE made all the asynchronous single-agent become cooperative that made the global network convergence faster in asynchronous reinforcement learning. Also, by reasonably setting the size of our reward model and shared experiences, we can effectively avoid a lot of extra calculations.

6 Conclusion

In this thesis, we attempted to accelerate the learning process of reinforcement by two ways: (1) shaping the reward to tell the agent how to learn, (2) sharing the experiences to make all asynchronous agents learn cooperatively just like a group. To shape the reward, we proposed a reward model, 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. For the sparse reward structure, our reward model, TAMRM, can revise the sparse structure by instead the 0 rewards with prediction reward based on the trajectory data. For the dense reward structure, our reward model revise the original reward and made it easier to learn. For our FEE method, it makes the asynchronous agents learn to solve the control problem cooperatively with sharing the experiences just like playing as the role of teacher that telling others what should do in order to solve the problem. The important point is that our reward model did not need the expert demonstration data or extend any structure in the neural network and can get the easier to learn reward by less calculation. Theoretically, it can be used in all types of reinforcement learning methods based on reward. To make the asynchronous reinforcement learning become cooperative, we used the experience shaping setting of multi-reinforcement learning that sharing meaningful experiences among the agents to make global network convergence faster. Our FEE method extended the horizon of each agent which made them understand all the failure and successful experience each episode, not only it's own but also other agents. Thus, each agent could more effectively learn to avoid failure and approach to success.

In our reward model, 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 with continuous action space. Also, the FEE method had a limitation about the parameter $\mathcal{K}$. To avoid a great amount of gradient backpropagation calculations, we could not set the $\mathcal{K}$ too large. However, if we set the $\mathcal{K}$ too small, the learning efficiency of this method would be reduced. So, in this future work, we will attempt to extend the current method to make it applicable to other problems by using other encoding methods to solve the limitation of TAMRM, and we will try to find a method to calculate the optimal $\mathcal{K}$ value based on the training situation to extend the FEE method.

7 Acknowledgment

I work this dissertation with my professor Sugawara. I would like to express my gratitude from the bottom of my heart for his consistent support, encouragement, and patience throughout this process. I thank my parents for believing me constantly and the help of all lab members.

REFERENCES

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM.

Abel, D., Salvatier, J., Stuhlmüller, A., and Evans, O. (2017). Agent-agnostic human-in-the-loop reinforcement learning. *arXiv preprint arXiv:1701.04079*.

Asmuth, J., Littman, M. L., and Zinkov, R. (2008). Potential-based Shaping in Model-based Reinforcement Learning. In *AAAI*, pages 604–609.

Badnava, B. and Mozayani, N. (2019). A new potential-based reward shaping for reinforcement learning agent. *arXiv preprint arXiv:1902.06239*.

Brys, T., Harutyunyan, A., Suay, H. B., Chernova, S., Taylor, M. E., and Nowé, A. (2015). Reinforcement learning from demonstration through shaping. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Da Silva, F. L., Glatt, R., and Costa, A. H. R. (2017). Simultaneously learning and advising in multiagent reinforcement learning. In *Proceedings of the 16th conference on autonomous agents and multiagent systems*, pages 1100–1108.

Devlin, S. M. and Kudenko, D. (2012). Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 433–440. IFAAMAS.

Eck, A., Soh, L.-K., Devlin, S., and Kudenko, D. (2013). Potential-based reward shaping for POMDPs. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1123–1124.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Grześ, M. (2017). Reward shaping in episodic reinforcement learning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 565–573. International Foundation for Autonomous Agents and Multiagent Systems.

Harutyunyan, A., Brys, T., Vrancx, P., and Nowé, A. (2014). Off-policy shaping ensembles in reinforcement learning. *arXiv preprint arXiv:1405.5358*.

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

Ibarz, B., Leike, J., Pohlen, T., Irving, G., Legg, S., and Amodei, D. (2018). Reward learning from human preferences and demonstrations in Atari. In *Advances in Neural Information Processing Systems*, pages 8011–8023.

Knox, W. B. and Stone, P. (2012). Reinforcement learning from simultaneous human and MDP reward. In *AAMAS*, pages 475–482.

Laud, A. D. (2004). *Theory and Application of Reward Shaping in Reinforcement Learning*. PhD thesis, University of Illinois at Urbana-Champaign, USA. AAI3130966.

Marom, O. and Rosman, B. (2018). Belief reward shaping in reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Machine Learning Proceedings 1994*, pages 181–189. Elsevier.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

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

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850.

Saunders, W., Sastry, G., Stuhlmueller, A., and Evans, O. (2018). Trial without error: Towards safe reinforcement learning via human intervention. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2067–2069. International Foundation for Autonomous Agents and Multiagent Systems.

Souza, L. O., Ramos, G. d. O., and Ralha, C. G. (2019). Experience sharing between cooperative reinforcement learning agents. *arXiv preprint arXiv:1911.02191*.

Suay, H. B., Brys, T., Taylor, M. E., and Chernova, S. (2016). Learning from demonstration for shaping through inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 429–437.

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

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337.

Torrey, L. and Taylor, M. (2013). Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 1053–1060, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Vanschoren, J. (2018). Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638.

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

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

Wu, Y. and Tian, Y. (2017). Training Agent for First-Person Shooter Game with Actor-Critic Curriculum Learning. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.

Xiaohui, Z. and Toshiharu, S. (2020). Meta-reward model based on trajectory data with k-nearest neighbors method. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages YY–ZZ. IEEE.

Zhu, C., Leung, H.-f., Hu, S., and Cai, Y. (2019). A q-values sharing framework for multiple independent q-learners. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2324–2326. International Foundation for Autonomous Agents and Multiagent Systems.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum Entropy Inverse Reinforcement Learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1433–1438. AAAI Press.

Zou, H., Ren, T., Yan, D., Su, H., and Zhu, J. (2019). Reward shaping via meta-learning. *arXiv preprint arXiv:1901.09330*.