






# Actor-Critic for Multi-agent System with Variable Quantity of Agents

Guihong Wang  and Jinglun Shi  

South China University of Technology, Guangzhou 510641, China  
eew.guihong@mail.scut.edu.cn, shijl@scut.edu.cn

**Abstract.** Reinforcement learning (RL) has been applied to many cooperative multi-agent systems recently. However, most of research have been carried on the systems with fixed quantity of agents. In reality, the quantity of agents in the system is often changed over time, and the majority of multi-agent reinforcement learning (MARL) models can't work robustly on these systems. In this paper, we propose a model extended from actor-critic framework to process the systems with variable quantity of agents. To deal with the variable quantity issue, we design a feature extractor to embed variable length states. By employing bidirectional long short term memory (BLSTM) in actor network, which is capable of process variable length sequences, any number of agents can communicate and coordinate with each other. However, it is noted that the BLSTM is generally used to process sequences, so we use the critic network as an importance estimator for all agents and organize them into a sequence. Experiments show that our model works well in the variable quantity situation and outperform other models. Although our model may perform poorly when the quantity is too large, without changing hyper-parameters, it can be fine-tuned and achieve acceptable performance in a short time.

**Keywords:** Multi-agent · Reinforcement learning · Variable quantity of agents · Communication · Fine-tune

## 1 Introduction

In recent years, owing to the great progress in deep learning, reinforcement learning (RL) has attracted a lot of attention from researchers [1]. By combining with deep neural network, it has been applied to a variety of fields and solved many problems, such as game playing including Atari video games and Go game high-dimensional robot control and etc.

Previous works have extended reinforcement learning to multi-agent domain. In cooperative systems, where all the agents share the goal of maximizing the discounted sum of global rewards, most researchers fix the quantity of agents in the systems and pay more attention to the communication between them. The CommNet [2] uses a single network to control agents. Each agent sends its hidden state as communication message to the embedded communication channels. The averaged message from other agents then is sent to the next layer of a specific agent. Unlike CommNet, developed from DQN, ACCNet [3] and MADDPG [4] are both extended from actor-critic policy

gradient method. They collect actions from all agents, and put the concatenation of them into the critic network, using the critic as a communication medium. However, these methods should fix the quantity of agents before training, and when the quantity is changed, both of them should be retrained. In lots of practical applications, we cannot know the quantity of agents in the environment in advance. Additionally, the number may change over time. Take the urban traffic control as an example, the cars in the road are always moving, so it's impractical to fix the quantity of agents in the learning model.

Tampuu [5] etc. simply use independent Deep Q-learning Network (IDQN) to control agents. This approach avoids the scalability problem, but because of the experience replay, a thorny problem appears that the environment may become non-stationary from the view of each agent. To solve this problem, Leibo et al. [6] have limited the size of experience replay buffer to keep track of the most recent data, while Foerster [7] uses a multi-agent variant of importance sampling and fingerprint to naturally decay obsolete data in the experience replay memory. Similarly, DIAL [8] also uses a single network for each individual agent, but in their model, each network has an extra output stream for communication actions. When communication is to be performed, the source agent outputs a communication signal and puts it into the target network for the next timestep. However, the environment may still become non-stationary since the message needs to be delayed for one timestep.

Recurrent neural network (RNN) has also been an effective method for coordinating variable quantity agents in some research [9]. The actor network in our proposed method is similar to the BiCNet [10] which uses BLSTM [11, 12] unit as a communication medium between agents. With BLSTM, it shares all parameters so that the number of parameters is independent of the number of agents and allows it to train using only a smaller number of agents, while freely scaling up to any larger number of agents during the test. However, RNN should be used in the sequence situation while in most natural systems, it can't directly regard the agents as a sequence.

In this work, we propose a model extended from actor-critic framework to process the multi-agent systems with variable quantity agents. In our model, we add a feature extractor to embed variable length states. The actor networks play the role in making decision for agents, and similar to BiCNet [10], by employing BLSTM, the agents can communicate and coordinate with each other. However, it is noted that the BLSTM is generally used to process sequences, so we use the critic network as an importance estimator for all agents and organize them into a sequence, sorting by their importance. Besides, because of partial observability, we embed a long short term memory (LSTM) layer in the critic network for single agent to maintain its historical states. Our experiments show that our model can still work when the quantity of agents is changed, and if the model cannot perform well in the systems with too many agents, it can be fine-tuned in the new system and get acceptable performance in a short time.

## 2 Proposed Method

### 2.1 Preliminaries: Multi-agent Markov Games

In this paper, we consider a fully cooperative multi-agent setting in which the system is composed of a set of states  $\mathcal{S}$ , a set of actions  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_N$  and a set of rewards  $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_L$ . Each agent  $i$  uses a stochastic policy  $\pi_{\theta_i} : \mathcal{S}_i \times \mathcal{A}_i \mapsto [0, 1]$  to choose actions, and later the next state will be produced according to the state transition function  $T : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_N \mapsto \mathcal{S}$ . Simultaneously, each agent  $i$  will obtain rewards as a function of the state and agent's action  $r_i : \mathcal{S} \times \mathcal{A}_i \mapsto \mathcal{R}$ . In this setting, all agents should cooperate with each other to maximize the global expected return:

$$R = \sum_{i=1}^N \sum_{t=0}^T \gamma^t r_{it} \quad (1)$$

where  $\gamma$  is a discount factor,  $T$  is the time horizon and  $N$  is the total quantity of agents. In addition, we use local observation setting in which each agent has its own observations. The states of an selected agent  $\mathcal{S}_i$  can be divided into  $\mathcal{S}_s, \mathcal{S}_e$  and  $\mathcal{S}_o$ .  $\mathcal{S}_s$  is the property of itself, while  $\mathcal{S}_o$  is a set of states of its observed agents and  $\mathcal{S}_e$  stands for a set of states of observed objects that can't be controlled in the system.

In reinforcement learning, there are several terminologies. State value function, denoted  $V_\pi(s)$ , is defined as the expected return when starting in  $s$  and following the policy  $\pi$  thereafter. It can be formulated as:

$$V_\pi(s) = E_\pi[R_t | \mathcal{S}_t = s] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | \mathcal{S}_t = s \right] \quad (2)$$

Similarly action value function or Q-value, denoted  $Q_\pi(s, a)$  is defined as the expected return starting from  $s$  taking the action  $a$ , and thereafter following policy  $\pi$ :

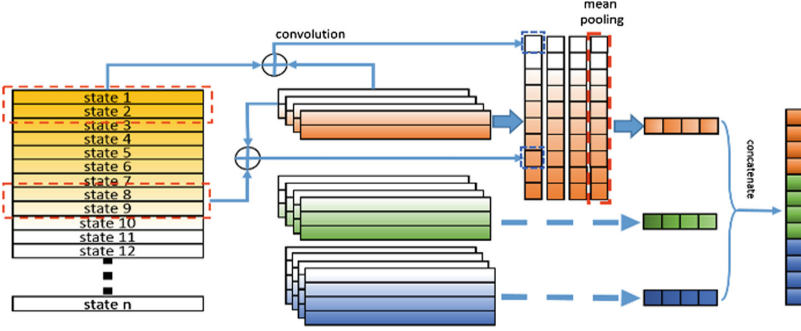
$$Q_\pi(s, a) = E_\pi[R_t | \mathcal{S}_t = s, A_t = a] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | \mathcal{S}_t = s, A_t = a \right] \quad (3)$$

### 2.2 Feature Extractor

In our setting, the states contain three parts: the selected agent's states  $S_s$ , the other observed agents' states  $S_o = \{S_{o_1}, \dots, S_{o_{N-1}}\}$  and observed objects' states in the system  $S_e = \{S_{e_1}, \dots, S_{e_M}\}$ . Because of partial observability, both the quantity of other observed agents  $N$  and observed objects  $M$  may change when the selected agent move, which means that the length of  $S_o$  and  $S_e$  are variable.

To achieve the goal that the network can deal with the variable length features, we add a module to preprocess the states. Let  $S_{o_i} \in R^d$  and  $S_o = \{S_{o_1}, \dots, S_{o_n}\} \in R^{n \times d}$ . As the Fig. 2 shows, similar to textCNN [13], we apply a filter  $w \in R^{h \times d \times c}$  with  $c$  channels to a window of  $h$  agents and produce a new feature map  $f_m \in R^{(n-h+1) \times c}$  after a convolution operation. Besides, we apply a mean pooling operation over all windows and then get a feature vector  $f_{vo1} \in R^c$ . This pooling scheme naturally deals with variable lengths and reduce the influence of the operation that we organize the other

agents' states into  $So$  randomly (Fig. 1). As the pooling operation may compress and lose useful information, we use a number of filters with different window length and then make a concatenation of them as  $f_{vo} = [f_{vo1}, f_{vo2}, \dots]$  to get more rich information.



**Fig. 1.** The structure of feature extractor 1-dimension filters is applied to states and after the convolution operation, new feature maps are processed by mean pooling operation, followed by a concatenation among different filters.

The operation for the  $Se = \{Se_1, \dots, Se_M\}$  is the same to that for the  $So_i \in R^d$ . Let the output of the feature extractor for  $Se$  as  $f_{ve} = [f_{ve1}, f_{ve2}, \dots]$ . At last, we concatenate  $f_{vo}, f_{ve}$  and the feature of the selected agent as the input for the subsequent deep neural network.

### 2.3 Critic Network

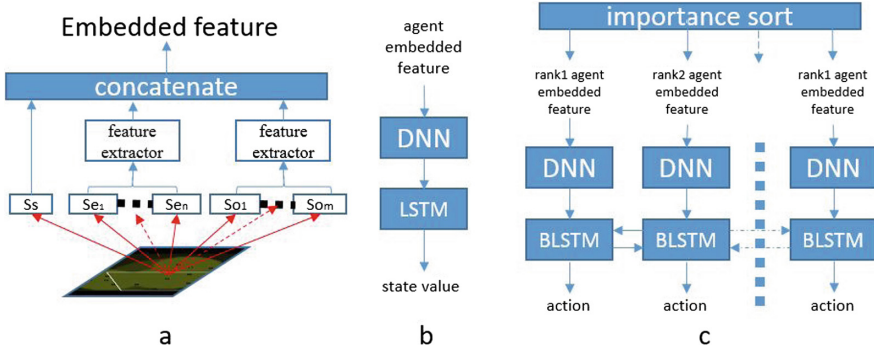
The critic network is used to estimate the current state value of a single agent. In order to make the estimations as accurate as possible, we need to collect all the useful states that the agent can obtain.

As the Fig. 2a shows, the inputs to the critic network is composed of three parts: the state of the environment, the state of the current agent, and the state of the other agents processed by the feature extractor mentioned in the previous section. The three parts are joined together and put into deep neural network. The structure of the network is shown in the Fig. 2b. Considering the local observation setting, we add a LSTM layer over time before the output layer to remember the historical states which is beneficial to estimate current state value more accurately. The output of the network is a continuous value that represents the state value of the selected agent in the current state.

The training of the network uses a supervised learning approach, where the time differential loss is shown as follows:

$$loss = E_{r,s}[(y - V(s_t))^2] \quad (4)$$

$$y = r_{it} + V(s_{t+1}) \quad (5)$$



**Fig. 2.** Our proposed model. a is the process of feature extractor dealing with variable length states. b is the structure of a critic network while c is the processes of the actor networks.

Wherein  $r_{it}$  represents the reward obtained by the agent  $i$  at time  $t$ ;  $s_t$  represents the states of the agent at time  $t$ .

## 2.4 Actor Network

Actor networks are used to map agents' states into actions. In our framework, the actor networks are still decentralized, indicating that each agent makes decisions based on its local information.

The structure of the actor network is shown in the Fig. 2c. The input module is the same to the critic network, where feature extractors are applied. Furthermore, we use a BLSTM layer to help the agents communicate with each other. However, the structure of the RNN is used to process sequence, and if the rankings in the sequence is changed, the obtained result may be different. In multi-agent systems, multi-agents cannot be naturally considered as a sequence in most situations, and different ranking of the agents may output different result. So we need a criterion to depend which order is proper.

In the multi-agent decision-making process, important agents should have high priorities for decision-making. Each agent can be sorted according to its importance, and then makes decision successively by order. Maximizing global score is the ambition for every agent, so the agents that have stronger scoring ability should be given greater importance. With this idea, we use the critic network to evaluate the importance of the agents in the system. We apply the critic network to all agents and get their state value. Then we sort them according to their state value and successively input them into the actor networks. It is noted that the critic network is only used in training steps. The state value describes the potential scoring ability in certain states and thus it can be regarded as the importance of the agents.

In single-agent actor-critic algorithms [14], agent update its parameters with local rewards. The gradient can be formulated as follows:

$$\nabla_{\theta} J(\theta) = E_{s \sim \rho^{\mu}, a \sim \pi} [\nabla_{\theta} \log \pi(a|s)(r + V_{\theta_v}(s') - V_{\theta_v}(s))] \quad (6)$$

Wherein  $\rho^{\pi}$  is the distribution of state,  $\theta_v$  is the parameter of value network, and  $\theta_{\pi}$  is the parameter of actor network.

However if we directly use this update in the multi-agent setting, it may encourage the agents to maximize their local return and ignore the global return leading to a local optimum. To eliminate this contradiction, we update the network with a global temporal difference loss, aiming to stimulate all agents coordinate to maximize the global return. The parameters of the actor network for agent  $i$  update as follow:

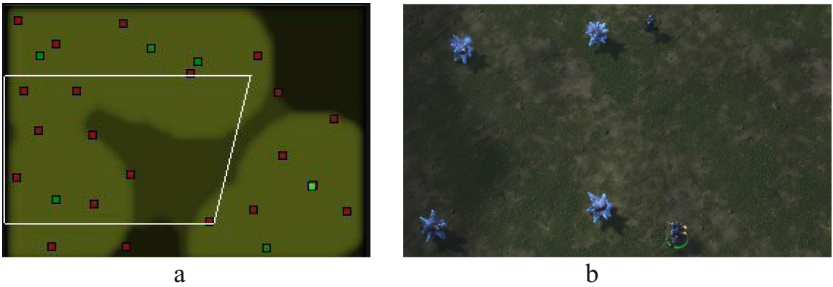
$$\nabla_{\theta_i} J(\theta_i) = E_{s \sim \rho^{\mu}, a \sim \pi_i} [\nabla_{\theta_i} \log \pi_i(a_i|s_i) \delta] \quad (7)$$

$$\delta = \delta_1 + \dots + \delta_N = \sum_1^N r_i + V_{\theta_{vi}}(s_i) - V_{\theta_{vi}}(s_i) \quad (8)$$

### 3 Experiments

#### 3.1 Experiment Setup

**Environment.** To perform our experiments, we modify the environment proposed in pycs2 [15], a challenging environment for reinforcement learning. In our task, there is a large map with some agents and 50 mineral shards. Rewards will be earned when an agent touch a mineral shard. To achieve the optimal score, the agents should split up and move independently. Whenever all mineral shards in the map have been collected, a new set of Mineral Shards are spawned at random locations. The collection time is limited to 3 min. Besides, the agents only have local vision, and can just perceive the presence of other agent and mineral shards within its scope (Fig. 3).



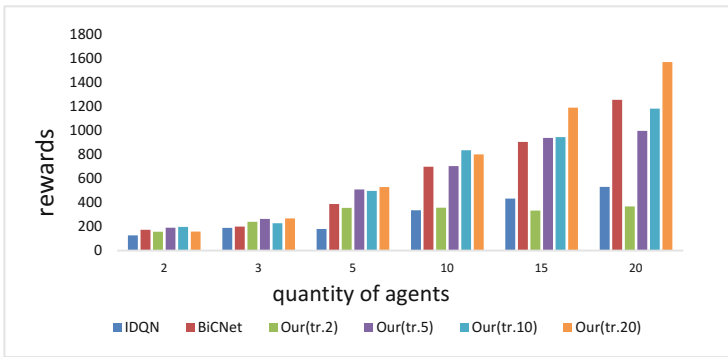
**Fig. 3.** The figure a is global view of our environment, where red objects are mineral shards and green objects are agents. The figure b is a local view. (Color figure online)

**Baselines.** We implement two baseline networks: independent DQN (IDQN) and BiCNet. This two networks are able to process the variable quantity issue, but without

local information processing units, they work poorly in the system with variable quantity of agents. We fix the length of the local state, and train and test this two models in the system with fixed quantity agents. Differently, we train the proposed model with certain quantity and test with variable quantity. For example, the IDQN and BiCNet will train on the system with 3 agents, and test on that system to evaluate its ability. While our model will train on the system with 3 agents, and test on the system with 2, 3, 5 or more agents. Our model shares all parameters so that the number of parameters is independent of the number of agents and allows it to train using only a smaller number of agents, while freely scaling up to any number of agents during the test.

### 3.2 Results

We train our model with four systems, and test them in 6 systems with different agents. As it is shown in the Fig. 4, with the increase of the quantity, the reward become larger.



**Fig. 4.** Results of the models. tr.2 means that the model is trained on the system with 2 agents.

It is because that more agents can get more mineral shards. But the growth rate is different. The tr.20 model can outperforms the other models in most scenarios, In most of the scenarios that the train quantity is larger than the test quantity, indicating that the agents can co-ordinate with each other and split well very well to increase the global reward. It can work well in different systems with different quantities of agents. The agent collect local information in the task, where the local scope can be regarded as a small system with variable quantities of agents, so it can have strong generalization ability.

It should be noted that as the quantity increase, model tr.2 and tr.3 work worse, and tr.5 and tr.10 also not work as well as tr.20. It is because that when the agents in the system increase, conflict between agents also increase. Model tr.20 is trained with 20 agents, so it can adapt for more complicated situations. While tr.2 and tr.3 only is trained with little agents, so the model haven't master the knowledge for complicated situations. That is to say, when an agent meet with ten or more agents, it may not know how to perform efficiently.

Although our model have limitation on generalization in some respects, with highly scalable structure, it needn't change any hyper parameters and the networks can be fine-tuned in the systems with more quantity of agents. As the Fig. 5 shows, we choose model tr.2, tr.3 and tr.3 to be fine-tuned in the system with 20 agents, and compare the learning curve to other models.

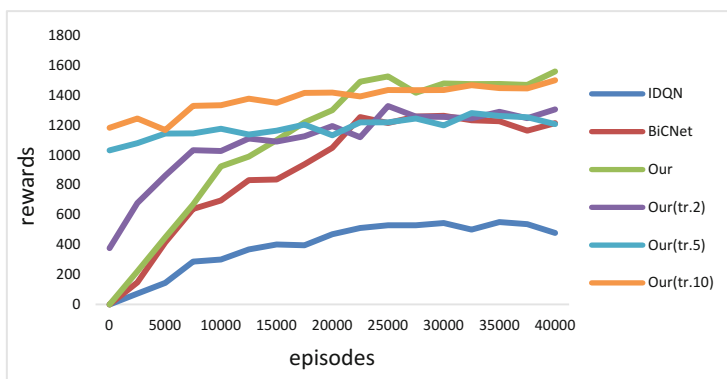


Fig. 5. Learning curve of different models trained with 20 agents.

Compared to other models that are retrained from scratch, the fine-tuned models can spend less time to reach the acceptable performance. Because the ability of dealing with variable quantities of agents, the three fine-tuned have a stronger ability to score in the beginning. Besides, the learning curve of them are steep before about 15000 episodes and become gentle after that. Without adjusting hyper parameters, previous knowledge in the networks can be transferred to the new one, so they can win at the starting line. In general, the fine-tuned models outperform IDQN and BiCNet, but they are not as good as our model that is retrained from scratch. It seems that fine-tuning may cause the network get into local optimum while retraining can help the model break away from local optimum and go farther. However, in some time-critical systems, our model can save a lot of time especially when the quantity of agents is very large and achieve acceptable performance.

## 4 Conclusion

In this paper, we proposed a model extended from actor-critic framework for the systems with variable quantities of agents. We not only design a feature extractor for networks to deal with variable quantity issue and embed a BLSTM layer in the actor networks, enabling agents to co-ordinate with each other. Furthermore, we use the critic network to compute the importance of all agents and sort them into a sequence. Experiments show that our model work well in the variable quantity situation and outperform other models. Although our model may perform poorly when the quantity



is too large, without changing hyper-parameters, it can be fine-tuned and achieve acceptable performance in a short time.

## References

1. Li, Y.: Deep reinforcement learning: An overview. arXiv preprint [arXiv:1701.07274](https://arxiv.org/abs/1701.07274) (2017)
2. Sukhbaatar, S., Fergus, R.: Learning multiagent communication with backpropagation. In: *Advances in Neural Information Processing Systems* (2016)
3. Mao, H., et al.: ACCNet: Actor-Coordinator-Critic Net for Learning-to-Communicate with Deep Multi-agent Reinforcement Learning. arXiv preprint [arXiv:1706.03235](https://arxiv.org/abs/1706.03235) (2017)
4. Lowe, R., et al.: Multi-agent actor-critic for mixed cooperative-competitive environments. In: *Advances in Neural Information Processing Systems* (2017)
5. Tampuu, A., et al.: Multiagent cooperation and competition with deep reinforcement learning. *PLoS ONE* **12**(4), e0172395 (2017)
6. Leibo, J.Z., Zambaldi, V., Lanctot, M., Marecki, J., Graepel, T.: Multi-agent reinforcement learning in sequential social dilemmas. arXiv preprint [arXiv:1702.03037](https://arxiv.org/abs/1702.03037) (2017)
7. Foerster, J., et al.: Learning to communicate with deep multi-agent reinforcement learning. In: *Advances in Neural Information Processing Systems* (2016)
8. Foerster, J., Assael, Y.M., de Freitas, N., Whiteson, S.: Learning to communicate with deep multi-agent reinforcement learning. In: *Advances in Neural Information Processing Systems*, pp. 2137–2145 (2016)
9. Foerster, J.N., Assael, Y.M., de Freitas, N., et al.: Learning to communicate to solve riddles with deep distributed recurrent q-networks. arXiv preprint [arXiv:1602.02672](https://arxiv.org/abs/1602.02672) (2016)
10. Peng, P., Wen, Y., Yang, Y., et al.: Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games. arXiv preprint [arXiv:1703.10069](https://arxiv.org/abs/1703.10069) (2017)
11. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. *IEEE Trans. Sig. Process.* **45**(11), 2673–2681 (1997)
12. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
13. Kim, Y.: Convolutional neural networks for sentence classification. *Eprint Arxiv* (2014)
14. Konda, V.R., Tsitsiklis, J.N.: Actor-critic algorithms. In: *Advances in Neural Information Processing Systems* (2000)
15. Vinyals, O., Ewalds, T., Bartunov, S., et al.: Starcraft ii: A new challenge for reinforcement learning. arXiv preprint [arXiv:1708.04782](https://arxiv.org/abs/1708.04782) (2017)