Negotiation Strategy using Reinforcement Learning for OneShot Track

Takumu Shimizu

Tokyo University of Agriculture and Technology

shimizu@katfuji.lab.tuat.ac.jp

Abstract

In the SCML OneShot Track, few agents have utilized neural networks or machine learning, and none have been trained using reinforcement learning (RL). In this work, we propose two negotiation strategies based on RL. The first, RLAgent, negotiates with opponents independently and inherits the SimpleAgent. The second, RLSyncAgent, inherits the SyncAgent and is capable of receiving and responding to multiple opponent offers. To apply RL to the agents, we define the Markov Decision Process for the OneShot Track. The state consists of the current number of rounds, the current needs, and the opponent's offer. The action consists of the accept signal, and the counter offer. The reward is the profit of the day. The definition is slight different RLAgent and RLSyncAgent to adjust their actions. We evaluate the trained agents and select RLAgent as the agent to submit to the competition.

1 RLAgent

RLAgent inherits SimpleAgent and negotiates with opponents independently. In addition, RLAgent is trained by Reinforcement Learning (RL). To do this, we define the Markov Decision Process for the OneShot Track and adjust to OneShotAgent.

1.1 Markov Decision Process (MDP) for OneShotAgent

To apply RL to OneShotAgent, an MDP was formulated for birateral negotiation in the OneShot Track. A finite MDP consists of 3 factors: the state space, the action space, and the rewards. Negotiations in the OneShot Track is defined using a finite MDP as follows:

State

State consists of the following factors:

- The current number of rounds $r \in \{0, 1, ...R\}$.
  R is the negotiation deadline.
- The current needs q_r^{need}.
  It is the same as the quantity of the exogenous contracts on the day.
- The opponent's offer $\omega_r'^a$.
  $\omega_r'^a$ consists of the quantity q', the negotiation time t', and the unit price p'.

Action

Action consists of the following factors:

- The accept signals η_r^a.
  η_r^a indicates whether to accept or reject an opponent's offer.
- The counter offer ω_r^a
  ω_r^a consists of the quantity q and the unit price p.

Reward

Reward is the profit of the day by using the utility function (OneShotUfun). OneShotUfun calculates the profit by using the contracts of the day and the exogenous contracts. In the last round on the day, RLAgent get the profit as reward. Otherwise, it get 0 as reward.

1.2 Negotiation Strategy

In the first round of the negotiation, there are no offers of opponent's. We make supposed offer, enter into the model as state, and get an action. RLAgent send the first offers included in the action to the opponents. The quantity of the supposed offer is 5, median of the negotiation issue. The unit price of the supposed offer is the best price of opponent. In other words, the maximum price when the opponent is seller, the minimum price when the opponent is buyer.

In other round, RLAgent receive the opponent's offer, enter into the model as state and get action. For the corresponding action for each opponent, RLAgent send an acceptance response when the accept signal is true or send a counter offer when the accept signal is false.

After RLAgent get the action from the trained model, it make the response and counter offer. However, the response is END_NEGOTIATION if the needs $q^{\text{need}} \leq 0$. In addition, the offer with excessive quantity is rejected forcibly.

1.3 Policy Network and Reinforcement Learning

We use the Actor-Critic approach, which is a type of reinforcement learning algorithm. Specifically, We used the Policy Gradient Algorithm [2], which involves optimizing the policy of the agent through gradient ascent. Within the Actor-Critic framework, I utilized the Proximal Policy Optimization (PPO) algorithm [1], which has been shown to be effective in improving sample efficiency and reducing variance in the training process. To optimize the neural network, we used Mean Squared Error (MSE) as the loss function. An overview of the model is shown in the following figure.

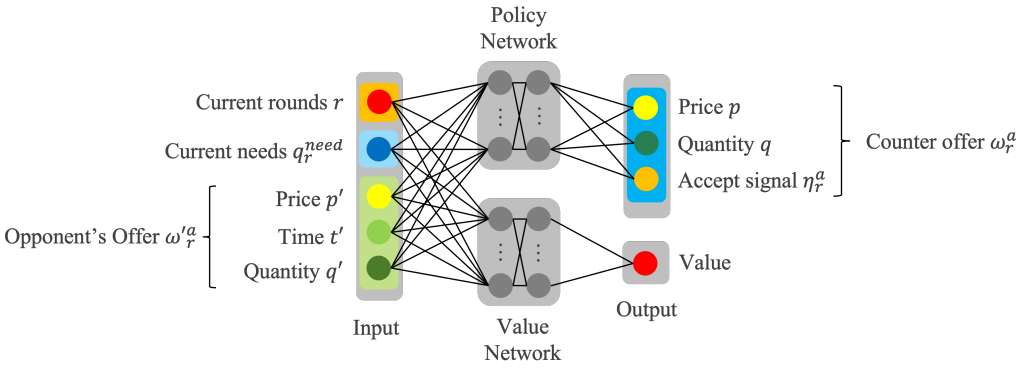

Figure 1: Model overview of RLAgent

Input and output nodes are split per each item and each item is represented by one-hot encoding (Multi Discrete). For example, the unit price of the offer is represented by two nodes ([0,0]) because the negotiation issue of the unit price is only two types.

We conducted a lot of simulations to train the policy network. The opponents in the simulation are SimpleAgent, AdaptiveAgent, And LearningAgent. We don't used the agents in the last year competition because the environment of the OneShot Track is quite different this year from last year. The best model which has the best score in the simulations is used in the evaluation.

2 RLSyncAgent

RLSyncAgent inherits SyncAgent, receiving multiple opponent's offers and making multiple responses. In addition, RLSyncAgent is trained by Reinforcement Learning (RL). To do this, we define the Markov Decision Process for the OneShot Track and adjust to SyncAgent.

2.1 Markov Decision Process (MDP) for OneShotSyncAgent

To apply RL to SyncAgent, an MDP was formulated for concurrent negotiation in the OneShot Track. A finite MDP consists of 3 factors: the state space, the action space, and the rewards. Negotiations in the OneShot Track is defined using a finite MDP as follows:

State

State consists of the following factors:

- The current number of rounds $r \in \{0, 1, ...R\}$.

  R is the negotiation deadline.

- The current needs q_r^{need}.

  It is the same as the quantity of the exogenous contracts on the day.

- The opponent's offers ω'_r.

  ω'_r is the set of opponent's offer $\omega_r'^a$, where a is the opponent negotiator. $\omega_r'^a$ consists of the quantity q', the negotiation time t', and the unit price p'.

Action

Action consists of the following factors:

- The accept signals η_r.

  η_r is the set of the accept signal η_r^a, indicating whether to accept or reject an opponent's offer.

- The counter offers ω_r

  ω_r is the set of opponent's offer ω_r^a. ω_r^a consists of the quantity q and the unit price p.

Reward

Reward is the profit of the day by using the utility function (OneShotUfun). OneShotUfun calculates the profit by using the contracts of the day and the exogenous contracts. In the last round on the day, RLSyncAgent get the profit as reward. Otherwise, it get 0 as reward.

2.2 Negotiation Strategy

In the first round of the negotiation, there are no offers of opponent's. We make supposed offers, enter into the model as state, and get an action. RLSyncAgent send the first offers included in the action to the opponents. The quantity of the supposed offer is 5, median of the negotiation issue. The unit price of the supposed offer is the best price of opponent. In other words, the maximum price when the opponent is seller, the minimum price when the opponent is buyer.

In other round, RLSyncAgent receive the opponent's offers, enter into the model as state and get action. For the corresponding action for each opponent, RLSyncAgent send an acceptance response when the accept signal is true or send a counter offer when the accept signal is false.

After RLSyncAgent gets the action from the trained model, it make the response and the counter offer. However, the response is END_NEGOTIATION if the needs $q^{\text{need}} \leq 0$.

2.3 Policy Network and Reinforecement Learning

We use the Actor-Critic approach, which is a type of reinforcement learning algorithm. Specifically, We used the Policy Gradient Algorithm [2], which involves optimizing the policy of the agent through gradient ascent. Within the Actor-Critic framework, I utilized the Proximal Policy Optimization (PPO) algorithm [1], which has been shown to be effective in improving sample efficiency and reducing variance in the training process. To optimize the neural network, we used Mean Squared Error (MSE) as the loss function. An overview of the model is shown in the following figure.

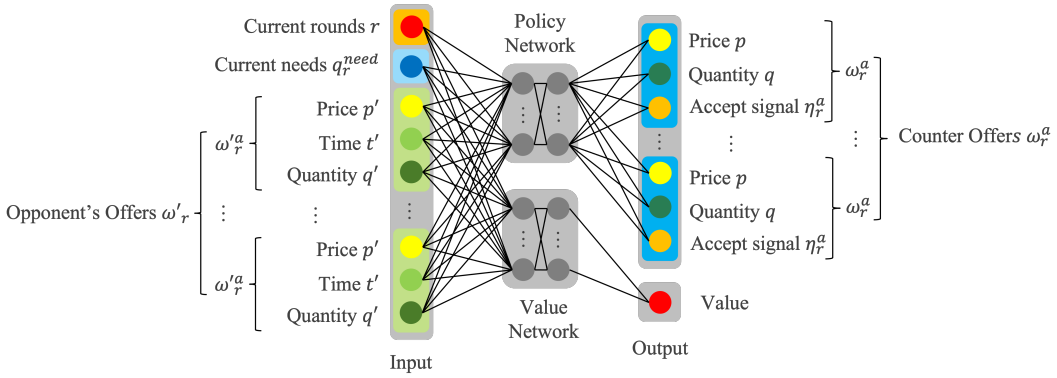

Figure 2: Model overview of RLSyncAgent

Input and output nodes are split per each item and each item is represented by one-hot encoding (Multi Discrete). For example, the unit price of the offer is represented by two nodes ([0,0]) because the negotiation issue of the unit price is only two types.

We conducted a lot of simulations to train the policy network. The opponents in the simulation are SimpleAgent, AdaptiveAgent, And LearningAgent. We don't used the agents in the last year competition because the environment of the OneShot Track is quite different this year from last year. The best model which has the best score in the simulations is used in the evaluation.

3 Evaluation

We tested RLAgent and RLSyncAgent in simulations against SimpleAgent , AdaptiveAgent, and LearningAgent. The results are shown in Table 1.

Table 1: The test results of RLAgent and RLSyncAgent

Agent	score	min	Q1	median	Q3	max
RLAgent	0.927	0.708	0.864	0.947	0.991	1.051
RLSyncAgent	0.712	0.173	0.461	0.809	0.910	1.056
SimpleAgent	1.035	0.595	1.004	1.080	1.127	1.204
AdaptiveAgent	0.978	0.620	0.883	0.989	1.083	1.206
LearningAgent	0.982	0.618	0.881	0.981	1.110	1.212

The results show that trained agents had a lower utility value than the sample agent.

Especially, RLSyncAgent get significantly lower on all scores. The reason of this is the excessive quantity of contracts that are signed. When RLSyncAgent determines which offer to accept, it can adjust the total quantity of the contracts to the proper value. However, when RLSyncAgent determines what offer to make, it is difficult to adjust the total quantity due to predictions of accepted offers. It is necessary to send the offer with small quantity, however, RLSyncAgent couldn't learn such strategy.

RLAgent get higher utility value than RLSyncAgent, however, it get lower score than sample agents. The reason of this is not considering other negotiations. When RLAgent determines what offer to accept or make, it can make more suitable action by considering other good situations negotiation. Future work includes adding information on offers exchanged in other negotiations.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[2] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12:1057–1063, 2000.