Towards Modern Card Games with Large-Scale Action Spaces Through Action Representation

Zhiyuan Yao*, Tianyu Shi†, Site Li‡, Yiting Xie‡, Yuanyuan Qin‡, Xiongjie Xie‡, Huan Lu‡ and Yan Zhang‡
*School of Business, Stevens Institute of Technology, Hoboken NJ USA
† Intelligent Transportation Systems Centre, University of Toronto, Ontario Canada
‡Deterrence, rct AI, Burbank CA USA
Email: *zyao9@stevens.edu, †ty.shi@mail.utoronto.ca, ‡lisite, xieyiting, qinyuanyuan, eric, hiker, yan@rct.ai,

***Abstract*—Axie infinity is a complicated card game with a huge-scale action space. This makes it difficult to solve this challenge using generic *Reinforcement Learning* (RL) algorithms. We propose a hybrid RL framework to learn action representations and game strategies. To avoid evaluating every action in the large feasible action set, our method evaluates actions in a fixed-size set which is determined using action representations. We compare the performance of our method with two baseline methods in terms of their sample efficiency and the winning rates of the trained models. We empirically show that our method achieves an overall best winning rate and the best sample efficiency among the three methods.**

***Index Terms*—Game AI, Reinforcement Learning, Large-Scale Action Space, Action Representation, Axie Infinity**

I. Introduction

Games have facilitated the rapid development of RL algorithms in recent years. Card games, as a classical type of games, also pose many challenges to RL algorithms. The direct applications of generic algorithms [1]–[4] in card games are problematic in many aspects because of large-scale discrete action spaces [5] with millions of actions. Prior works have proposed RL methods to approach a number of traditional card games, like Texas Hold'em [6]–[8], Mahjong [9], DouDizhu [5], [10], etc. However, the issues brought by the large action space still remain, especially for modern card games such as *Axie Infinity*[1] which has a huge discrete action space.

Axie Infinity is an one-versus-one online card game which has millions of players globally. *Axies* are virtual pets that have different attributes such as species, health, speed, etc. Each axie has its own card deck consisting of 2 copies of 4 cards. The player needs to form a team of 3 axies at the beginning and play their cards (24 cards in total) to beat the opponent player's team.

This game is different from the traditional card games in the following aspects:

1) For a fixed team, the player needs to choose one sequence of cards from 23 million different card sequences.
2) The effect of a card is usually influenced by many factors such as its position in the card sequence, other cards, and the status of the axies.
3) There are thousands of teams for players to choose. The optimal strategies for different teams are various.

These difficulties are all connected to the large-scale action space, and they are shared by a lot of other modern card games such as Hearthstone[2].

Some existing works have investigated the large action space issue in card games. Zha et al. [5] propose an action encoding scheme for DouDizhu. However, this encoding scheme cannot properly encode the action in our problem as the complexity of the action space in our problem is much higher than that in DouDizhu (27472 possible moves). Dulac-Arnold et al. [11] propose to choose actions in a small subset of the action space to speed up the action search process. This set is chosen based on a proper action encoding method which usually relies on prior knowledge. However, the prior human knowledge for our problem is hard to obtain due to the diversity of the teams and the strategies. Chandak et al. [12] propose an algorithm to learn action representations from the consequences of corresponding actions. This method can avoid using prior human knowledge, but the policy-based method produces optimal actions which are not feasible in the discrete action set. We also mention several general techniques in [13] to reduce the size of the action space to improve the performance. However, they are not applicable in our case as it still requires prior human knowledge of this game.

In this paper, we consider a hybrid RL method to deal with the large discrete action space. This method chooses the optimal action in a small subset of the large-scale feasible action set. It can quickly train models on different teams with minimal prior knowledge. We test our method with other baseline methods using Axie Infinity task. We have the following main contributions:

1) A Markov Decision Process (MDP) formulation for Axie Infinity problem including a novel action encoding scheme.
2) An efficient RL method to solve card game problems with large-scale discrete action space.
3) A supervised learning method to learn action representations.
4) Empirical results demonstrating the superiority than other baseline methods

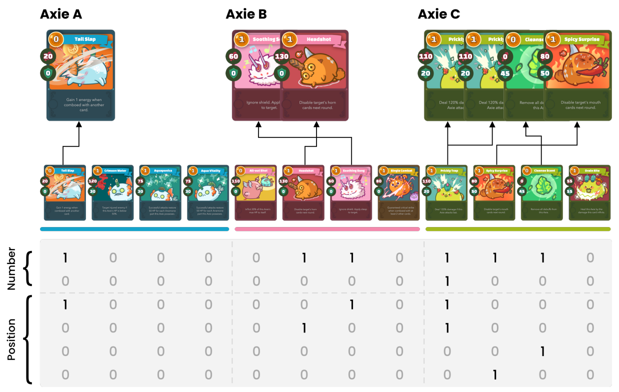
[1]A detailed description of Axie Infinity and its rules can be found in https://whitepaper.axieinfinity.com/

[2]https://playhearthstone.com

II. Problem Formulation

We assume our agent uses a fixed team to play against a rule-based player who randomly uses multiple popular teams. As the opponent is fixed, we formulate this environment as a single-agent Markov Decision Process (MDP). This MDP has a finite time horizon, each time step is one game round. The process is terminated when one player is defeated. The one-step transition probability measure is denoted as P_t. Every state in the state space $\mathcal{S}$ consists of all the available information to our player. This includes all six axies' status, energy, available cards, card history, etc.

In Axie Infinity, each team has three axies, each of which has two copies of 4 distinct cards. Thus, this forms a 24-card deck. In each round, the player places a sequence of cards for each axie, thus three sequences of cards are placed. Each sequence can have at most 4 cards by the rule of the game. Considering all these rules, we propose a novel action encoding scheme to vectorize an action as a 6×12 matrix. One example is given in Figure 1. All legit actions form the discrete action space $\mathcal{A}$ whose size equals 23,149,125. In contrast, DouDizhu in [5] only has 27,472 actions.



Number	1	0	0	0	0	1	1	0	1	1	1	0
	0	0	0	0	0	0	0	0	1	0	0	0
Position	1	0	0	0	0	0	1	0	1	0	0	0
	0	0	0	0	0	1	0	0	1	0	0	0
	0	0	0	0	0	0	0	0	0	0	1	0
	0	0	0	0	0	0	0	0	0	1	0	0

Fig. 1. A demonstration of an encoded action consisting of 3 card sequences. Axie A/B/C respectively places 1/2/4 cards in this round. The status of each card is encoded in a 6-digit vector where the first two digits represent the number of this card and the rest 4 digits encode its positional information. The matrix is formed by 12 such vectors for 12 distinct cards.

We define the reward as the result of the game, with a penalty on the activity of discarding cards. Mathematically, we denote the terminal state set $\mathcal{T}$ as a set of states at which the game is over. For a transition tuple (s_t, a_t, s_{t+1}), $0 \leq t < T-1$, we define the reward function as

$$r_t(s_t, a_t, s_{t+1}) = \begin{cases} I - c \cdot n_d, & \text{if } s_{t+1} \in \mathcal{T}, \\ 0, & o.w., \end{cases}$$

where I and n_d are components in s_{t+1}. The game result indicator I equals to 1/0/-1 if the agent wins/ties/loses the game, and n_d is the number of discarded cards in the whole game. The positive constant c adjusts the importance of the penalty term.

III. Methodology

Due to the challenges brought by the large-scale action space, we consider to only evaluate a small group of actions which are filtered out from all feasible actions. We form this small set of actions with those actions which have similar effects with a target action. This target action is generated by a policy function. A distance function is needed to measure the "similarity" between two actions. Inspired by Word2Vec [14], we learn an action embedding function which maps one-hot-like action vectors into a continuous space which is defined as the latent action space. The Euclidean distance in this latent space reflects the difference between two actions in terms of their effects. We notice that the method in [11] incorporates a similar idea. The difference between their work with ours is that [11] uses the Euclidean distance in the original action space instead of the latent space.

A. *A Decision Procedure*

We illustrate the decision procedure in Figure 2. The algorithm generates a point in the latent action space based on the state. Those feasible actions which are close to this point in the latent space form a candidate set of actions. Then, we apply the Q-function to evaluate each action in this set to find the optimal one.

Specifically, this decision procedure of our method consists of 3 parametrized components:

1) A raw policy function $u_{\theta_1} : \mathcal{S} \to \mathbb{R}^n$ where n is the dimensions of the action space, $\mathcal{A} \subset \mathbb{R}^n$.
2) An embedding function $f_{\theta_2} : \mathbb{R}^n \to \mathcal{E}$ where $\mathcal{E}$ is the latent action space, and $\mathcal{E} \subset \mathbb{R}^m$, $m < n$. For an arbitrary action a, $e = f_{\theta_2}(a)$ is called the latent action representation of a.
3) A state-action value function, i.e., a Q-function $q_{\theta_3} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The Q-value $q_{\theta_3}(s, a)$ evaluates the expected return when executing action a at state s.

For a given input s, the overall policy function μ selects the action using the following procedure. First, we obtain a point $\bar{a} = u_{\theta_1}(s)$ as a "raw action", note it is possible that $\bar{a} \notin \mathcal{A}$. Second, we denote the set of all available actions for s as $U(s)$. We calculate the distance between feasible actions with the raw action in the latent space by

$$d(a, \bar{a}; f_{\theta_2}) = \|f_{\theta_2}(a) - f_{\theta_2}(\bar{a})\|^2,$$

for all $a \in U(s)$. We form a k-element subset of $U(s)$ with the top k closest actions to the raw action in the latent space, denote it as $\mathcal{U}_k(s; \theta_1, \theta_2)$. Mathematically, this is done by

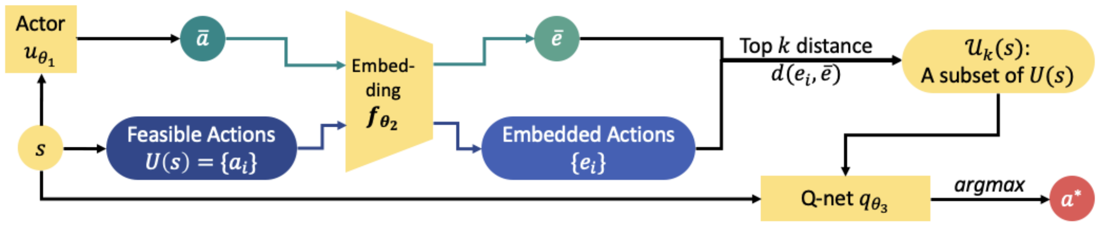
$$\mathcal{U}_k(s; \theta_1, \theta_2) = \mathop{\arg\min}_{\mathcal{U} \subset U(s), |\mathcal{U}| = k} \sum_{a \in \mathcal{U}} d(a, u_{\theta_1}(s); f_{\theta_2}). \tag{1}$$

In the last, we select the action which has the highest Q-value in this subset. If we denote an overall policy function as μ for a given state s, the action is selected by

$$a^* = \mu(s; \theta_1, \theta_2, \theta_3) = \mathop{\arg\max}_{a \in \mathcal{U}_k(s; \theta_1, \theta_2)} q_{\theta_3}(s, a). \tag{2}$$

B. *Training procedures*

Our method consists of three sets of parameters θ_1, θ_2, and θ_3. We design a two-stage algorithm to train these parameters.



Actor u_{θ_1}
$\bar{a}$
s
Feasible Actions $U(s) = \{a_i\}$
Embed-ding f_{θ_2}
$\bar{e}$
Embedded Actions $\{e_i\}$
Top k distance $d(e_i, \bar{e})$
$\mathcal{U}_k(s)$: A subset of $U(s)$
Q-net q_{θ_3}
argmax
a^*

Fig. 2. An illustration on the decision procedure.

In the first stage of training, we design a supervised learning method to learn the embedding function f_{θ_2}. We learn the action representations based on the effects of the actions on the system states. For instance, assume an arbitrary state $s \in \mathcal{S}$ and two actions a_1 and a_2, if the probability measure $p(s, a_1)$ is similar to $p(s, a_2)$ where $p(s, a) = P(S_{t+1} \mid S_t = s, A_t = a)$, we say a_1 and a_2 have similar effects. Following this idea, we define a deterministic transition function $m_{\theta_4} : \mathcal{S} \times \mathcal{E} \to \mathcal{S}$, $m_{\theta_4}(s, f_{\theta_2}(a))$ should estimate the next state after executing a at s. We define the following objective function for the first stage training

$$J_1(\theta_2, \theta_4) = \mathbb{E}_{P_t, \cdot}[(m_{\theta_4}(s, f_{\theta_2}(a)) - s')^2]. \tag{3}$$

We collect the transition tuples (s, a, s') in a dataset D by randomly selecting actions. Then, we apply a gradient-based optimization method to optimize θ_2 and θ_4 by minimizing J_1 on D. We only need θ_2 in the next stage.

In the second stage of training, similar to Deep Deterministic Policy Gradient in [2], we apply an iterative training procedure to alternatively update the deterministic raw policy function (the actor) u_{θ_1} and the Q-function (the critic) q_{θ_3}. We use the raw action $\bar{a}$ instead of the final action a^* in the policy improvement part, details can be found in [11]. The Q-function training uses Monte-Carlo estimate described in [15].

IV. Experiment

We compare our method with two baseline methods:

1) Douzero. We adapt the Douzero method in [5] to our problem. We design a similar action encoding scheme as the one mentioned in this paper, where each action is encoded as a 2-by-12 matrix.
2) Douzero+pooling. We reduce the size of the action space by shrinking its dimensions. We design this baseline method by adding an 1D pooling layer in [16] on the flattened actions from Douzero.

Indeed, other techniques to reduce the scale of action spaces are mentioned in [13]. They inevitably introduce prior human knowledge, which conflicts with our intention, and this makes the comparison unfair. We try to answer the following questions through experiments:

1) Does our method produces overall better performance than other baseline methods?
2) Does our method achieve better sample efficiency?

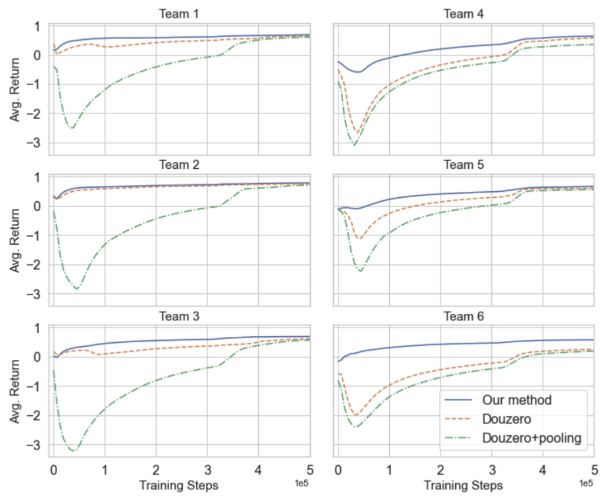


Fig. 3. **Comparison on the learning curves.** Six line-charts shows the learning curves of 18 models in the first 5×10^5 steps. It can be seen that our method shows a faster convergence with the same amount of samples on most of the 6 teams.

A. Sample Efficiency

We select six teams that are popular at different levels in the global rank[3]. We train a model for each team using 3 methods: our method, the Douzero method, and the Douzero+pooling method. Thus, we trained 18 models in total. To make a fair comparison, we stop the training after 1×10^7 steps for all these models.

Figure 3 shows the evolution of average returns during training in the first 1×10^5 steps. It can be seen that, on most of the teams, the models from our method have the highest average return among the three by trained with the same amount of samples. This indicates that our method can quickly produce high-quality models with limited amount of samples.

B. Battle

We evaluate the performance of each model by letting it play against other models and rule-based players. This guarantees that the models trained for the same team have the same set of opponents. Each battle consists of 1000 games. Each model plays 29000 games against various opponents to obtain a

[3]A detailed description of these 6 teams can be found in our full paper: https://arxiv.org/pdf/2206.12700.pdf

TABLE I
THE AVERAGE WINING RATE OF 3 METHODS ON 6 TEAMS
(174000 GAMES FOR EACH METHOD)

Method	Winning rate
Our method	**0.4986**
Douzero	0.4477
Douzero+pooling	0.4283

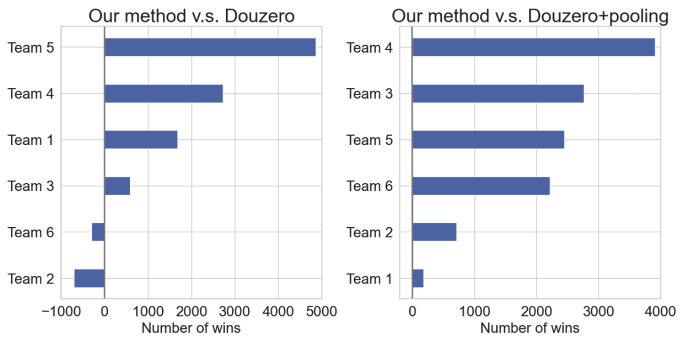


Fig. 4. **Comparison on battle performance between our method with two baseline methods on 6 teams.** We train 18 models using these three methods on 6 teams. We evaluate each model by letting it play against a same set of opponent players consisting of the other trained agents and random players (29000 battles in total). Note the qualities in the left/right bar-plot respectively show (the number of wins of our method - the number of wins of the Douzero/Douzero+pooling method).

comprehensive result. This makes the variance of estimators of the winning rates small enough to show statistical significance.

We aggregate the winning rate of the models trained by each method in Table I. It can be seen that the overall winning rate of our method is 5% and 7% higher than the Douzero and Douzero+pooling method respectively. This also confirms our method has a better generalization ability on different teams.

We also compare the models which use the same team. For each team, we calculate the difference between the number of wins by our method with that of Douzero or Douzero+pooling. A positive difference indicates our method plays this team better than the baseline method. We visualize these differences in the number of wins in Figure 4. We can find that our method can achieve a higher number of wins across most teams than the baseline methods. This indicates that our method has good generalizability to most team types. We notice that our method cannot outperform Douzero on teams 2 and 6. The reason is that strategies for these two teams are more diversified than those for the other teams. This makes high-reward actions far away from each other in the latent space. In such cases, the Douzero method may make better decisions than our method because it evaluates every feasible actions, though it is more computationally expensive.

V. CONCLUSIONS

In this study, we try to use an RL method to solve the challenges in a card game problem with a large action space. We give an MDP formulation for the game Axie Infinity. We propose a general RL algorithm to learn the strategies of different teams. We design a training procedure to learn the action embedding function without prior information. We empirically demonstrate our method outperforms the baseline methods in terms of the battle performance and the sample efficiency.

Our work can be improved by incorporating the self-play technique [17] in training to enhance the opponents. Moreover, many modern card games provide rich textual data to explain the rules and the cards. Using advanced language models to learn prior information from such textual data can enhance the action embedding component in our method. Future works may focus on these directions for further improvement.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.
[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[5] D. Zha, J. Xie, W. Ma, S. Zhang, X. Lian, X. Hu, and J. Liu, "Douzero: Mastering doudizhu with self-play deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 333–12 344.
[6] M. Bowling, N. Burch, M. Johanson, and O. Tammelin, "Heads-up limit hold'em poker is solved," *Science*, vol. 347, no. 6218, pp. 145–149, 2015.
[7] N. Brown and T. Sandholm, "Superhuman ai for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2018.
[8] ——, "Superhuman ai for multiplayer poker," *Science*, vol. 365, no. 6456, pp. 885–890, 2019.
[9] J. Li, S. Koyamada, Q. Ye, G. Liu, C. Wang, R. Yang, L. Zhao, T. Qin, T.-Y. Liu, and H.-W. Hon, "Suphx: Mastering mahjong with deep reinforcement learning," *arXiv preprint arXiv:2003.13590*, 2020.
[10] Y. Guan, M. Liu, W. Hong, W. Zhang, F. Fang, G. Zeng, and Y. Lin, "Perfectdou: Dominating doudizhu with perfect information distillation," *arXiv preprint arXiv:2203.16406*, 2022.
[11] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.
[12] Y. Chandak, G. Theocharous, J. Kostas, S. Jordan, and P. Thomas, "Learning action representations for reinforcement learning," in *International conference on machine learning*. PMLR, 2019, pp. 941–950.
[13] A. Kanervisto, C. Scheller, and V. Hautamäki, "Action space shaping in deep reinforcement learning," in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 479–486.
[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[16] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International conference on rough sets and knowledge technology*. Springer, 2014, pp. 364–375.
[17] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.