ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230581

Cardsformer: Grounding Language to Learn a Generalizable Policy in Hearthstone

Wannian Xia[a,b], **Yiming Yang**[b], **Jingqing Ruan**[b,c], **Dengpeng Xing**[a,b;*] **and Bo Xu**[a,b;**]

[a]School of Artificial Intelligence, University of Chinese Academy of Sciences
[b]Institute of Automation, Chinese Academy of Sciences
[c]School of Future Technology, University of Chinese Academy of Sciences

Abstract. Hearthstone is a widely played collectible card game that challenges players to strategize using cards with various effects described in natural language. While human players can easily comprehend card descriptions and make informed decisions, artificial agents struggle to understand the game's inherent rules and are unable to generalize their policies through natural language. To address this issue, we propose Cardsformer, a method capable of acquiring linguistic knowledge and learning a generalizable policy in Hearthstone. Cardsformer consists of a Prediction Model trained with offline trajectories to predict state transitions based on card descriptions and a Policy Model capable of generalizing its policy on unseen cards. To our knowledge, this is the first work to consider language knowledge in a card game. Experiments show that our approach significantly improves data efficiency and outperforms the state-of-the-art in Hearthstone even when there are untrained cards in the deck, inspiring a new perspective of tackling problems as such with knowledge representation from large language models. As the game constantly releases new cards along with new descriptions and new effects, the challenge in Hearthstone remains. To encourage further research, we make our code publicly available and publish PyStone, the code base of Hearthstone on which we conducted our experiments, as an open benchmark.

1 Introduction

Reinforcement learning (RL) has shown human-level performance in many traditional board games [17, 18] and poker games [3, 24]. Despite their large search spaces, the dynamics of these environments are rather simple. For instance, the state transition of Go is simply the placement of Go pieces. In recent years, many video games with complex dynamics have been introduced as testbeds to challenge reinforcement learning algorithms, such as StarCraftII [22] and Dota2 [2]. However, learning from scratch without any prior knowledge of the environment requires enormous training time, and the learned policy may easily fail when the environment dynamics change. In this work, with the inspiration of how human players learn and transfer to new environment dynamics with linguistically described prior knowledge, we investigate representing natural language to learn a generalizable policy in the popular collectible card game Hearthstone.

* Corresponding Author. Email: dengpeng.xing@ia.ac.cn
** Corresponding Author. Email: xubo@ia.ac.cn

Many previous works have designed specific environments for language grounding tasks like Messenger [12], RTFM [25], and HazardWorld [23]. These environments pose 2D maps with a single agent to perform a demanding goal, and the action space is limited to four movement directions. Unlike them, Hearthstone is a zero-sum game with two players competing against each other. With more than 3,000 collectible cards, the policy space is relatively large. The environment dynamics of Hearthstone are determined by the card effects, which are expressed in natural language and intricately integrated within the game mechanics.

Up to date, tree-search-based methods have been generally explored to develop Hearthstone AI systems and even achieved performance approaching that of human experts [20]. These methods usually require a delicate simulation program to generate outcoming results of performing different actions. However, developing such a simulation is not as straightforward in Hearthstone. The game dynamics of Hearthstone are complex, and the card effects may mutually affect. When new cards are released, the simulation program requires to be updated to provide proper results. Moreover, Hearthstone features imperfect information and stochasticity in card effects, tree-search methods need to be further modified to mitigate these difficulties.

In this work, we propose Cardsformer to learn a generalizable policy in Hearthstone. Instead of using a real-time simulation of the game states to construct a search tree, we train a neural network model capable of grounding natural language to predict state transitions and perform reasonable actions according to the predictions. The inference of Cardsformer is divided into two steps - predicting future states given current state and action representations with the Prediction Model and leveraging the results to approximate the state-action values with the Policy Model. To incorporate language descriptions in state and action representations, we use a pre-trained language model MPNet [19] to get sentence embedding. We train the Prediction Model with supervised learning on the collected offline trajectories of Hearthstone. Actively prompted by the fixed Prediction Model, the Policy Model is trained with Deep Monte-Carlo (DMC) Q-learning [24] in a self-play manner. Experiments demonstrate that our method outperforms previous methods in Hearthstone on both training cards and untrained cards. Besides absolute performance, we also show that Cardsformer takes advantage of the language grounding manner to boost data efficiency and generalization.

The contributions of this work can be summarized as follows:

- Cardsformer is the first end-to-end Hearthstone AI system that

Figure 1. Two sample cards, a snapshot of the game board, and a list of game entities in Hearthstone. As described down below on each card, taking advantage of the special effects is crucial to gameplay. In our assessment of Hearthstone as a testbed for RL algorithms, we focus on the agent's ability to generalize the usage of various cards based on their descriptions.

does not rely on a real-time simulated search tree. Our method achieves state-of-the-art (SOTA) performance on the training decks and can be transferred to other decks with unseen cards and descriptions with minor degradation.

- We are the first to propose a method that introduces a natural language prompted reinforcement learning agent in a card game. Experiments demonstrate that the Prediction Model, which takes advantage of language representations of a pre-trained language model, improves data efficiency and generalization capabilities of Cardsformer.
- The challenge of Hearthstone remains in many aspects. We release our experiment testbed, PyStone, as an open benchmark to inspire further research[1].

2 Related Work

2.1 Hearthstone AI

Building AI agents to perform game combat has been mostly investigated within Hearthstone. Many competitions have been held to promote related research [7, 14]. One main approach is to use Monte-Carlo Tree Search (MCTS) [4]. Świechowski et al. [20] modify MCTS to deal with imperfect information in Hearthstone and further enhance it with heuristic rules to solve the combinatorial explosion of game simulations. Choe et al. [5] utilize state abstraction to present the search space of Hearthstone as a directed acyclic graph and apply sparse sampling on their modified Upper Confidence Bound algorithm. Another approach is to use an Evolutionary Algorithm (EA). García-Sánchez et al. [8] propose a method to automatically calculate the weights of a hand-coded Hearthstone agent, and use an EA to optimize the function which scores all possible actions in a specific state. Besides game combat, there are also other AI-related research disciplines in Hearthstone, including balanced card/game design and automatic deckbuilding. De Mesentier Silva et al. [6] investigate methods for balancing meta-game to search for a combination of changes to the card attributes. García-Sánchez et al. [9] use a genetic algorithm to evolve automatic deckbuilding in Hearthstone. In our work, we propose Cardsformer which, unlike previous methods using MCST, does not require any real-time simulation of branching states. As the game states and card descriptions are represented as input vector features and sentence embedding respectively, Cardsformer is an end-to-end system where any card, trained or untrained, are potentially implemented within.

2.2 Reinforcement Learning with Natural Language

In recent years, a series of works have been proposed to learn generalizable policies with knowledge from natural language. One major domain is Instruction Following, where the agents are presented with language instructions. These instructions can describe navigation targets [13], object manipulation tasks [1], or enable multi-agent interactions [21]. Goyal et al. [10, 11] investigate the use of language for reward shaping in RL and achieve a faster learning process. In [16, 12, 25], the agents are informed with environment manuals to specify the goals or dynamics of the environment. However, these works consider rather small action space or naive environment dynamics (e.g., a 2D world with only four actions for an agent to do) and do not require explicit knowledge representation from language descriptions. In this work, we introduce Hearthstone as a testbed for such a language grounding task. With the intuition that language descriptions can tell the inner transition pattern between states, we train a supervised model to capture such patterns and a reinforcement learning agent with access to future state predictions. This framework differs from prior work in that it incorporates natural language representation to explicitly explain environment dynamics by predicting future states and taking advantage of the predictions to enhance performance.

[1] https://github.com/WannianXia/Cardsformer

3 Introduction of Hearthstone

The testbed for our method, Hearthstone, is a popular video game with millions of monthly active players worldwide. Players use their self-constructed decks and a unique hero power to compete in 1v1 duels. During the game, the players take turns performing actions of using cards or ordering minions to attack, until the opponent's hero is eliminated by reducing its health to 0. Unlike other board games, players can use their limited amount of mana crystals, which is 1 at the first turn and increases by 1 at every new turn, up to a maximum of 10, to perform multiple actions in a single turn. Thus, players must use the cards in an appropriate sequence to take full advantage of the card effects. For example, to destroy an opponent's minion with 5 *Health*, you can first enhance a minion with 2 *Attack* using the card *Blessing of Might: Give a minion +3 Attack*, and then order this minion (now 5 *Attack*) to attack the opponent's minion and destroy it. Each player can draw a new card from their remaining deck cards at the beginning of each turn. A snapshot of the game board and 2 sample cards are shown in Fig. 1.

In Hearthstone, the vast card pool is a central feature that allows players to construct their decks. Since the game was published in 2014, the card pool has been continuously enriched by game designers, and a total of more than 3,000 unique collectible cards have been released. The cards can be of the type spell, minion, or weapon. Spell cards are discarded after use. Instead, using minion cards will summon minions to the board on your behalf, and they can attack the opponent's minions or hero. Using weapon cards will equip your hero with a weapon and enable it to attack as a minion. The special effects of all types of cards are described in natural language, which can be dealing damage, restoring health, drawing cards, etc. Human players can ground the meaning of card descriptions, infer the results of using them, and then take reasonable actions. For artificial agents, Hearthstone provides a testbed for the agents to learn the natural language knowledge described by the cards and make decisions accordingly.

4 Preliminaries

Language Conditioned Partially-Observable Markov Decision Process (LC-POMDP) In our setting, we formulate the Hearthstone environment $\mathcal{G}$ as a Language Conditioned Partially-Observable Markov Decision Process, defined by the tuple $\mathcal{G} =< S, A, T, r, O, Z, E >$. S is the set of states, A is the set of actions. $T(s_{t+1}|s_t, a_t)$ is the conditional probability of the transition from state s_t to s_{t+1} by taking action a_t. $r(s_t, a_t)$ is the reward function that determines how the agent should be rewarded by taking action a_t in state s_t. O is the set of observations, and Z is the conditional probability $Z(o_t|s_t)$ of the observation o_t given the state s_t. In Hearthstone, a state s_t consists of hand cards, deck cards, and minions of both sides. An observation o_t consists of observable information about a player, where the hand cards and deck cards of the opponent are excluded. Additionally, we define E as the set of game entities. To model a game state s_t, we arrange game entities of hand cards, minions, heroes, and secrets as a sequence, i.e., $s = \{e_i\}_{i=1,2,\ldots,n}$. Each entity e_i has a feature vector c_i representing the entity's attributes and a pre-trained language embedding of its description l_i. e_i is then represented as a concatenation of c_i and l_i. In our method, we use MPNet [19] to get l_i. The actions are represented by referring entities, and the actual effect of taking an action is represented in the corresponding entity description l_i. More details are explained in Section 5.

Deep Reinforcement Learning for Hearthstone The optimization of LC-POMDP on Hearthstone can be solved by Deep Reinforcement Learning (DRL). DRL agents aim to learn a policy that maximizes future cumulative rewards discounted by γ: $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}(s, a)$. A typical algorithm, Q-learning, is widely used as a value-based RL algorithm. The state-action value function $Q(s, a)$ under π can be defined as $Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}(s, a) | s_t = s, a_t = a\right]$, where $a_t \sim \pi(a|s), s_{t+1} \sim T(s'|s, a)$. Cardsformer uses deep neural networks as function approximation of Q, and calculates the optimal state-action value function using the Bellman equation as:

$$Q^*(s, a) = \mathbb{E}_\pi[r(s, a) + \gamma \sum_{s'} T(s'|s, a) \max_{a'} Q^*(s', a')]. \quad (1)$$

The optimal policy can be derived as: $\pi^*(a|s) = \arg\max_a Q^*(s, a)$.

5 Methodology

Our model, Cardsformer, aims to learn a policy that can generalize to new cards. We describe the state/action representation, the model design, and the training methods here.

5.1 State/Action Representation

As shown in Fig. 2(a), the entity's feature vector c_i contains numerical information about its attack, health, and other attributes such as whether the entity has a *Divine Shield* or *Stealth*[2], represented in 0/1 flags. We designed four different types of entities: hand card, minion, hero, and secret. Hero powers and weapons are not considered separate entity types, instead, we treat hero powers as hand cards, and a weapon entity is represented within a hero entity. Unlike in other environments, the actions in Hearthstone correspond tightly with the entities - an action must originate from a source entity and may choose a target entity if necessary, and its special effect is described by the source entity. For example, in the case of using the hand card *Arcane Intellect* described as *Draw two cards*, the source entity is the hand card and it requires no target entity, while the action of using the card *Blessing of Might*, which has the description *Give a minion +3 attack*, requires the player to specify a target minion entity. Thus, to represent all possible actions in the game, each entity is attached with a *Source* flag and a *Target* flag. An action of using card entity e_i towards its target e_j is to tag the Source flag of e_i and the Target flag of e_j to 1, while these flags of other entities remain 0. In our practice, the Source and Target flags are included in c_i, and the model tells its description from corresponding l_i. Although actions are not separated from states as they are represented within c_i which is a component of the state, this is still reasonable since Cardsformer is a Q-value function approximator.

5.2 Model Design

As shown in Fig. 2(b), the Policy Model takes in the current state and action concatenated with predictions from the Prediction Model to compute the Q-values for the agent to make decisions upon. The predictions are scalar features of minion and hero entities in the next

[2] *Divine Shield* and *Stealth* are terminologies in Hearthstone, enabling the agent to negate the next damage once or not to be targeted respectively.

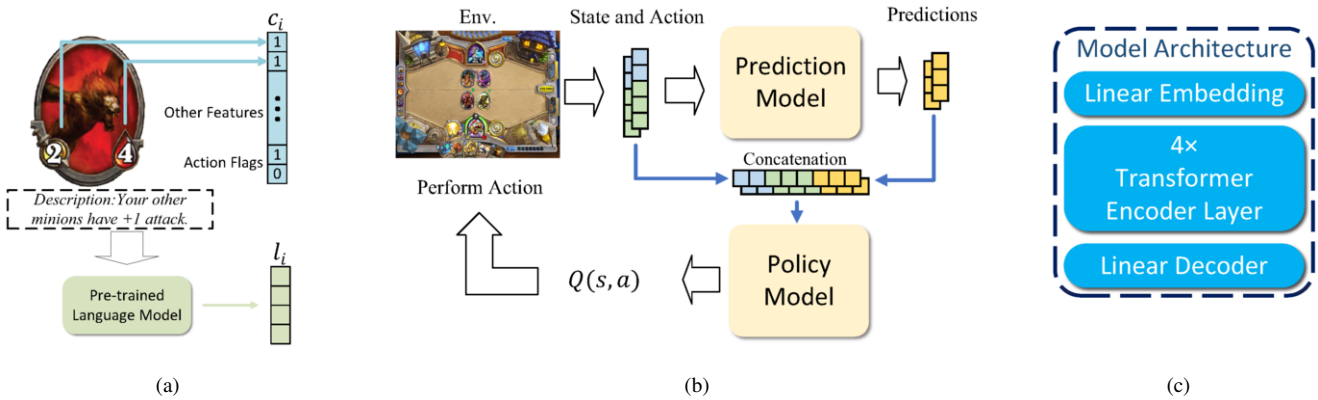


Figure 2. Architecture of Cardsformer: **(a)** State and action representation of a minion entity; **(b)** The overall framework of Cardsformer; **(c)** Model details for the Prediction Model and the Policy Model.

time step that can tell the effect of a given action to the most extent. Theoretically, concatenating the predictions as input is redundant for vanilla Q-learning. However, since we aim to learn a generalizable policy, we assume that a highly generalizable Prediction Model which can tell the Policy Model how acting on unseen cards will result in future states can be beneficial. Our experiments in subsections 6.3 and 6.4 demonstrate this. The Prediction Model and the Policy Model are trained separately and have similar architectures as shown in Fig. 2(c). The Linear Embedding layers embed c_i and l_i of different entity types into a uniform vector space, which is 320 dimensions – 64 from c_i and 256 from l_i. The embeddings of all entities are then fed into four layers of transformer encoder as sequential data. The feedforward dimension of the transformer encoder layer is 512, and each layer has 8 attention heads. The Linear Decoder decodes the output of the transformer into the state of a relevant entity at the next time step for the Prediction Model, or the Q-value for the Policy Model.

5.3 *Supervised Learning of the Prediction Model*

We train the Prediction Model with supervised data generated from the environment. We collect offline game trajectories by performing random actions at each state. Each data pair consists of a current state and an action as input, where the state of game entities at the next time step serves as the output label. Due to the partially-observable nature and stochasticity of the game, it is not applicable to predict the complete state of the game. For example, a drawing card is unknown in advance and unpredictable. Thus, we only predict selected values in vector feature c of hero and minion entities. This design aligns with how human players simulate future game states - not all values of a future state are concerned and predictable, the basic idea is to enlarge your minion army and damage the opponent's hero, which can be reflected in our design of the predictions. The training loss for the Prediction Model, noted as M with its parameters as θ, is:

$$\mathcal{L}_M(\theta) = \frac{1}{N_a}||M_\theta(o_t, a_t) - o_{t+1}||_2. \tag{2}$$

This training loss is a variation of MSE loss, in which we divide the sum of the mean squared differences between predicted and target values by N_a instead of the total number of elements, and N_a is the number of altered values in o_{t+1} compared to o_t. This is because different actions in Hearthstone can cause different elements in the observation to change, and the total number of changing elements may also vary. For example, a card that freezes a specific minion will almost only affect the *frozen* attribute. Our variation of MSE loss helps balance the loss between different actions and better suits this situation.

As the representation of a specific entity e_i, its feature vector c_i and language description l_i can be initially linked, such that if some c_i is unique, the model may neglect its l_i as the training processes. The Prediction Model is designed to learn the transition pattern represented by l_i instead of remembering c_i as a reference to card effects. Thus, during this process of training the Prediction Model, we randomly initialize the stats of each card entity at the beginning of each episode to get randomized c_i, then the model must query l_i to predict the effect of an action. This design avoids mapping feature vectors to state transitions in the model, which can enhance the model's generalization ability.

5.4 *Reinforcement Learning of the Policy Model*

We adopt Deep Monte-Carlo (DMC) [24] Q-learning to obtain the optimal policy. In general, DMC methods employ a central learner model to learn from sampled episodes that are generated by multiple actors. The parameters of the distributed actors will be synchronized with the central learner after the actors send their generated episodes. At each non-terminal state, the environment will provide a complete list of all available actions. The concatenations of these actions with the current state are fed into the Cardsformer system as an input batch to predict respective Q-values. Only the Policy Model is trained in this phase – the parameters of the Prediction Models are fixed. We do not work on heuristic rules to design rewards – the agent is only rewarded at the terminal state of the game with $+1$ for winning or -1 for losing. The discount factor for future rewards is set to 1. Thus, given episodes, the training loss for the Policy Model is to minimize the loss function $\mathcal{L}_Q$:

$$\mathcal{L}_Q(\theta) = \sum_{t=1}^{n} \mathbb{E}_{(o_t, a_t, r_t, o'_t)}[r_t + \gamma \max_{a'_t} Q_{\pi'}(o'_t, a'_i, M(o'_t, a'_t); \theta') - Q_\pi(o_t, a_t, M(o_t, a_t); \theta)] \tag{3}$$

where $M(\cdot)$ is the prediction from the Prediction Model as an additional input to the value function, θ and θ' are the parameters of the Policy Model and the corresponding target network, respectively.

6 Experiments

In this section, we first introduce our Experimental setup in Section 6.1. We show the absolute performance of Cardsformer, in terms of win rates compared to other methods, in the following Section 6.2. In Section 6.3, we perform an ablation study on the design of Cardsformer. We use Shapley values [15] to inspect how the language embedding and attributes in vector features affect the output of Cardsformer in Section 6.4.

6.1 Experimental Setup

We evaluate our method on PyStone, which is built upon the open-source project SabberStone[3]. In our setting, we skip the mulligan phase (in which the players have a chance to replace their starting hand cards) of the game since it is irrelevant to our concern in this paper. Cardsformer is trained on our predefined *training decks*. For each hero class, we build two or three decks (depending on common deck builds of the hero class) with classic and basic cards and get 20 different training decks out of nine classes. These cards and their derivatives are described in 186 different sentences. We collect a dataset of 10,000 games with approximately 800,000 pairs of labeled data by performing random actions between two random training decks. The Prediction Model is trained for 5,000 epochs on this dataset. After the Prediction Model is trained and fixed, we train the Policy Model with self-play. The agent is agnostic to its own and the opponent's decks, thus the agent must decide on its currently available hand cards and board minions instead of remembering an elaborated policy related to the deck. Training the Policy Model in a Linux server with 8 Titan XP GPUs and 64 CPU cores achieves a data throughput of approximately 2,500 frames (we denote a learning pair of $s, a \rightarrow Q(s, a)$ as a frame) per second, and we train the Policy Model of Cardsformer extensively with 100 million self-play frames, which requires approximately 11 hours of training.

To evaluate Cardsformer, we define 3 *testing decks* with 10, 20, and 30 unseen cards in place of original cards. Note that each deck consists of exactly 30 cards, this experimental design tests the generalization ability of Cardsformer at different levels. The unseen cards have different descriptions from the training cards. Nevertheless, due to the limited performance of the Prediction Model, the unseen cards are restricted to those with simple effects that do not rely on unique triggers. For example, the prediction model can generalize the effect of *Drain Life: Deal 2 damage, and restore two health to your hero* exactly as it described, but fails to predict *Lorewalker Cho: Whenever a player casts a spell, put a copy into the other players hand* since its effect is triggered by a unique condition and is not generally trained on the Prediction Model. Cardsformer is also evaluated on training decks against other methods, denoted as 0 unseen cards. A more detailed list of cards used in the experiments for training and testing Cardsformer can be found in our open-source project.

6.2 Comparison to Baseline Methods

We compare Cardsformer to three publicly available baselines, which are:

Dynamic Lookahead This is the first place method[4] in Hearthstone AI Competition 2020 [7]. It takes advantage of the partially-observable simulations implemented by SabberStone to look ahead at the results of available actions and defines a heuristic rule to evaluate the states. The look-ahead depth depends dynamically on the number of available actions, ranging from 1 to 3.

Aggressive/Controlling Agent These are the basic AI methods implemented within SabberStone using MCTS. It compares hero health and board minion attributes to evaluate the states and construct a search tree. The agent will visit nodes with the highest score by executing relevant actions. The evaluation can be in an aggressive or controlling style. The aggressive agent weights higher scores on damaging the opponent's hero, while the controlling agent prefers to maintain friendly minions while damaging the opponent's minions.

As shown in Table 1, Cardsformer achieves the best performance on both training and testing decks, except on the testing deck with 30 unseen cards compared with the SOTA method, Dynamic Lookahead. On training decks, Cardsformer beats Dynamic Lookahead with an average win rate of 78.0% in 1,000 random games. Unseen cards added to the deck do have an impact on Cardsformer, but it still outperforms Dynamic Lookahead on testing decks with 10 or 20 unseen cards. In the worst case, compared on 30 unseen cards, Cardsformer can still achieve an average win rate of 47.5%. Note that Cardsformer is trained in a specific environment with a limited set of cards, and it infers the dynamics and usage of unseen cards through neural networks. In comparison, other methods rely on simulation programs where the effects of each unique card are pre-implemented. These methods become inapplicable if new cards are introduced without updating the simulation program. However, Cardsformer can theoretically incorporate any new card. Apart from its superior performance, tested on a relatively common computer device powered by Intel Core i5-11400 CPU and NVIDIA GTX 1660s GPU, the inference time for Cardsformer to make a move is 0.053 seconds, while Dynamic Lookahead takes 0.078 seconds, Aggressive Agent takes 0.118 seconds and Controlling Agent takes 0.131 seconds, averaged on 1,000 states each.

6.3 Ablation Study

6.3.1 Model Components

There are two components related to the concern of grounding language to enhance performance in Hearthstone: the presence of the Prediction Model and sentence embedding with the pre-trained language model. We perform ablation studies on these two components to investigate how Cardsformer performs with or without them. First, we remove the prediction model to get *Language Only Cardsformer*, which only represents language descriptions with a pre-trained language model but does not learn how these descriptions correspond to state transitions. Furthermore, we replace the sentence embedding in Language Only Cardsformer with one-hot embedding from card IDs to get *One-Hot Policy*, which is to compare how language representation from the pre-trained language model affects the training process. We save a checkpoint per one million frames trained on each model design, up to 100 million frames. In Fig. 3(a), we compare these 300 checkpoints in an Elo ranking system including randomly initialized models, denoted as 0 training frames. Each game is between two random checkpoints and an identical training deck. The full version

[3] SabberStone is an open-source Hearthstone simulator designed to support AI research. See https://github.com/HearthSim/SabberStone

[4] https://hearthstoneai.github.io/botdownloads.html, developed by *Sebastian Miller*.

	Win Rates and Standard Deviation of Cardsformer (%)			
Unseen cards	0	10	20	30
Dynamic Lookahead	78.0 ± 2.5	70.7 ± 2.2	64.0 ± 4.0	47.5 ± 2.6
Aggressive Agent	92.9 ± 1.6	98.4 ± 1.4	96.8 ± 1.4	97.4 ± 1.9
Controlling Agent	94.1 ± 2.2	89.6 ± 2.7	87.7 ± 3.0	89.7 ± 3.5

Table 1. Win rates of Cardsformer against other methods on training decks and testing decks with different amounts of unseen cards. The win rates are evaluated on 1,000 games, and the standard deviations are calculated by separating every 100 games.

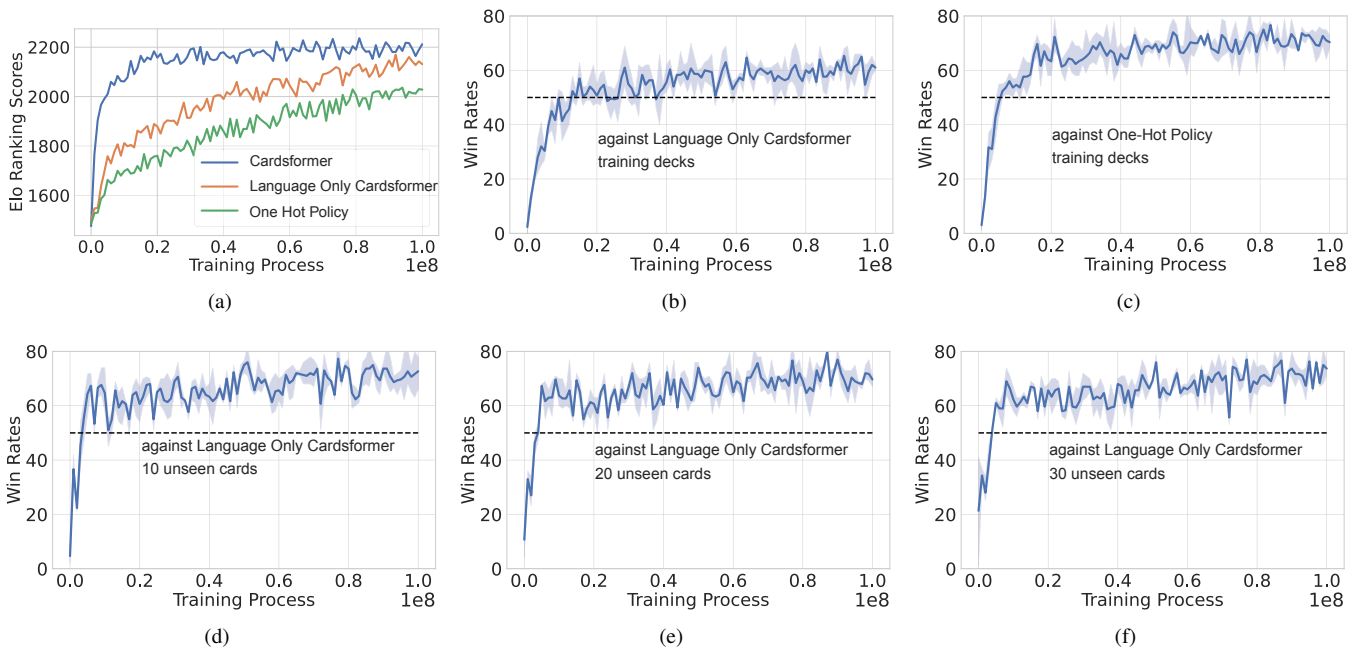
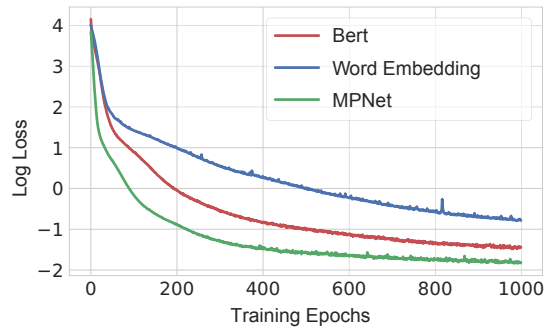


Figure 3. The comparison of ablation models throughout the training process. **(a)** shows the Elo ranking scores of all checkpoints. A total of 100,000 matches between two random checkpoints are recorded. **(b)** and **(c)** shows win rates of Cardsformer checkpoints against fully trained Language Only Cardsformer and One-Hot Policy on training decks respectively. **(d)** - **(f)** are comparisons between Cardsformer checkpoints and fully trained Language Only Cardsformer on decks with different amounts of unseen cards, which are 10, 20, and 30 respectively. Each win rate point is tested with 1,000 games and under 3 random seeds.

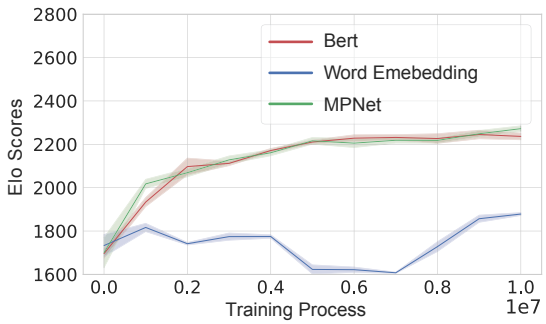
of Cardsformer significantly outperforms other versions in terms of data efficiency and final performance. After approximately 20 million training frames, the Elo score of full Cardsformer has already reached above 2100, close to Language Only Cardsformer trained with all 100 million frames and far beyond One-Hot Policy. Language Only Cardsformer also outperforms One-Hot Policy, neither of which has the Prediction Model, and we attribute this result to that the sentence embedding from pre-trained MPNet provides the agent with better representations of the cards instead of learning it from one-hot card IDs.

The Elo score system shows a qualitative comparison between all checkpoints. To get quantitative results and demonstrate the efficiency of Cardsformer, we compare the win rates of checkpoints from Cardsformer throughout the whole training process against fully trained Language Only Cardsformer checkpoint and fully trained One-Hot Policy checkpoint, as shown in Fig. 3(b) and Fig. 3(c) respectively. These figures show that as the training progresses, the win rate of Cardsformer rapidly increases: Only about 20 million frames of training data are needed to match the performance of fully trained Language Only Cardsformer, and it requires even less to match the performance of a fully trained One-Hot Policy, which is approximately 10 million frames. After training for 100 million frames (the end of the curves in Fig. 3(b) and Fig. 3(c)), fully trained Cardsformer has a win rate of about 62% against fully trained Language Only Cardsformer, and about 70% against fully trained One Hot Policy. Under the same training conditions, the performance of the complete Cardsformer model is better. These results indicate that both the Prediction Model and language embedding significantly improve the training efficiency of Cardsformer.

Theoretically, the performance of both Language Only Cardsformer and One-Hot Policy on the training decks can converge to the same level as complete Cardsformer as long as the input features are well-defined. The superiority of our design of Cardsformer lies not only in its improvement of data efficiency during the training process but also in its zero-shot generalization capabilities. As shown in Figs. 3(d) - 3(f), we compare the checkpoints of complete Cardsformer against fully trained Language Only Cardsformer on testing decks with different amounts of unseen cards (One-Hot Policy is not compared on testing decks because it is not capable to get one-hot embedding from extra unseen cards). These trends are almost identical to Fig. 3(b) but with a faster growth rate and a larger gap in the final performance. This indicates that the Prediction Model helps further enhance the performance of Cardsformer on decks with unseen cards.



(a)



(b)

Figure 4. Ablation study on language embedding methods regarding **(a)** the training loss of Prediction Model and **(b)** the elo scores of the checkpoints of the Policy Model.

6.3.2 *Language Embedding*

We compare three alternative methods generating language embedding l: 1) sentence embedding from MPNet (what we use in Cardsformer), 2) mean pooling of Bert embedding, and 3) word embedding without pre-training. The training loss across 1,000 epochs is shown in In Fig. 4(a). To investigate the impact of embedding methods on the Policy Model, we train Language Only Cardsformer, which only contains the Policy Model without resorting to the Prediction Model, with different embedding methods and compare trained checkpoints in an Elo system similar to Fig. 3(a), only that the models are not fully trained with 100 million frames but 10 million instead. Under three random seeds and 10,000 games between two random opponents, the results of Elo scores are shown in Fig. 4(b). Both results demonstrate that word embedding without pre-training performs much worse. Though MPNet performs nearly identically with Bert on training the Policy Model, it performs slightly better on training the Prediction Model.

6.4 *Interpreting Cardsformer*

The extraordinary performance of Cardsformer can be attributed to many aspects, one of which is the lavishness of the model input. We use all kinds of observable game information as model input of Cardsformer, including original and predicted entity states. To interpret how each component of the model input affects the Policy Model of Cardsformer, we use SHAP [15] to inspect the importance of input features to the Policy Model. SHAP uses the classic Shapley values from game theory and their related extensions to explain the output of neural network models. A higher Shapley value indicates the sample has a positive impact on the output, which in this case is the Q-value, while lower Shapley values indicate the sample reduces the output. In addition, we can also rank the importance of all the input features according to their contributions by disturbing the input value samples and comparing the Shapley value results. The distributions of samples from the five most important input features are shown in Fig. 5. This figure demonstrates that, when the Policy Model evaluates the Q-value of a given state-action pair, it allocates most of its credits to the Prediction Model. Take the No. 1 and No. 5 features as an example - the No. 1 feature in Fig. 5 is the predicted health of the opponent's hero. When the action causes the health of the opponent's hero to be higher, the evaluated Q-value will decrease (the red area is mainly distributed on the left), but if the action causes the health of your hero to be higher, the evaluated Q-value will increase (the red area is mainly distributed on the right).

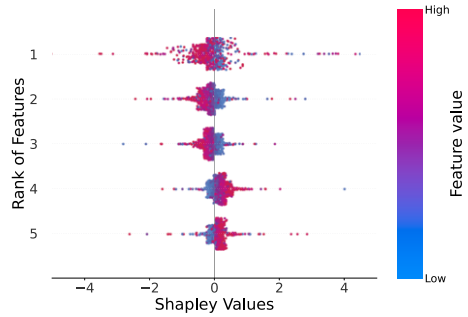


Figure 5. Shapley values of five most decisive input features, which are indexed on the vertical axis as: 1 - Opponent's Hero Health Prediction; 2 - Opponent's Max Mana; 3 - Opponent's Max Mana Prediction; 4 - Hero Max Mana; 5 - Hero Health Prediction.

7 Conclusions

In this paper, we adopt Hearthstone as our testbed for language grounding and decision-making tasks. Instead of learning from scratch, we use a large language model to get sentence embedding of card descriptions and pre-train a dynamics model to capture the inherent transition pattern of the game states. With this dynamics model, the agent is aware of the consequence of each available action, thus the RL process achieves higher data efficiency. Furthermore, the inherited generalization capability from the pre-trained language model enables the agent to perform on new descriptions even if they are unseen during training. Nevertheless, grounding language and learning a generalizable policy in Hearthstone still require further investigation. The dataset used to train the Prediction Model is generated with limited training cards and random actions, while the whole Hearthstone game features far more complex card effects. In our practice, we have noticed that Cardsformer fails to predict card effects that are triggered by unique conditions, which may be solved in the future by applying a larger and more diverse dataset. To inspire future work, we also publish our environment code as PyStone, which can be easily adopted in recent RL manners.

In our future work, we will apply this idea of grounding language-described state transitions in other environments and further investigate other disciplines of using natural language in reinforcement learning.

Acknowledgements

This work is supported by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant (No.XDA27030300) and the Program for National Nature Science Foundation of China (62073324).

References

[1] Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette, 'Learning to understand goal specifications by modelling reward', *arXiv preprint arXiv:1806.01946*, (2018).

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al., 'Dota 2 with large scale deep reinforcement learning', *arXiv preprint arXiv:1912.06680*, (2019).

[3] Noam Brown and Tuomas Sandholm, 'Superhuman ai for heads-up no-limit poker: Libratus beats top professionals', *Science*, **359**(6374), 418–424, (2018).

[4] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton, 'A survey of monte carlo tree search methods', *IEEE Transactions on Computational Intelligence and AI in games*, **4**(1), 1–43, (2012).

[5] Jean Seong Bjorn Choe and Jong-Kook Kim, 'Enhancing monte carlo tree search for playing hearthstone', in *IEEE Conference on Games (CoG)*, pp. 1–7, (2019).

[6] Fernando de Mesentier Silva, Rodrigo Canaan, Scott Lee, Matthew C Fontaine, Julian Togelius, and Amy K Hoover, 'Evolving the hearthstone meta', in *IEEE Conference on Games*, pp. 1–8, (2019).

[7] Alexander Dockhorn and Sanaz Mostaghim, 'Introducing the hearthstone-ai competition', *arXiv preprint arXiv:1906.04238*, (2019).

[8] Pablo García-Sánchez, Alberto Tonda, Antonio J Fernández-Leiva, and Carlos Cotta, 'Optimizing hearthstone agents using an evolutionary algorithm', *Knowledge-Based Systems*, **188**, 105032, (2020).

[9] Pablo García-Sánchez, Alberto Tonda, Giovanni Squillero, Antonio Mora, and Juan J. Merelo, 'Evolutionary deckbuilding in hearthstone', in *IEEE Conference on Computational Intelligence and Games*, pp. 1–8, (2016).

[10] Prasoon Goyal, Scott Niekum, and Raymond J Mooney, 'Using natural language for reward shaping in reinforcement learning', *arXiv preprint arXiv:1903.02020*, (2019).

[11] Prasoon Goyal, Scott Niekum, and Raymond J Mooney, 'Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards', *arXiv preprint arXiv:2007.15543*, (2020).

[12] Austin W Hanjie, Victor Y Zhong, and Karthik Narasimhan, 'Grounding language to entities and dynamics for generalization in reinforcement learning', in *International Conference on Machine Learning*, pp. 4051–4062, (2021).

[13] Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, et al., 'Grounded language learning in a simulated 3d world', *arXiv preprint arXiv:1706.06551*, (2017).

[14] Andrzej Janusz, Tomasz Tajmajer, and Maciej Świechowski, 'Helping ai to play hearthstone: Aaia'17 data mining challenge', in *Federated Conference on Computer Science and Information Systems*, pp. 121–125, (2017).

[15] Scott M Lundberg and Su-In Lee, 'A unified approach to interpreting model predictions', in *Advances in Neural Information Processing Systems*, volume 30, (2017).

[16] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola, 'Grounding language for transfer in deep reinforcement learning', *Journal of Artificial Intelligence Research*, **63**, 849–874, (2018).

[17] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., 'Mastering the game of go with deep neural networks and tree search', *Nature*, **529**(7587), 484–489, (2016).

[18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al., 'A general reinforcement learning algorithm that masters chess, shogi, and go through self-play', *Science*, **362**(6419), 1140–1144, (2018).

[19] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu, 'Mpnet: Masked and permuted pre-training for language understanding', in *Advances in Neural Information Processing Systems*, volume 33, pp. 16857–16867, (2020).

[20] Maciej Świechowski, Tomasz Tajmajer, and Andrzej Janusz, 'Improving hearthstone ai by combining mcts and supervised learning algorithms', in *IEEE Conference on Computational Intelligence and Games*, pp. 1–8, (2018).

[21] DeepMind Interactive Agents Team, Josh Abramson, Arun Ahuja, Arthur Brussee, Federico Carnevale, Mary Cassin, Felix Fischer, Petko Georgiev, Alex Goldin, Tim Harley, et al., 'Creating multimodal interactive agents with imitation and self-supervised learning', *arXiv preprint arXiv:2112.03763*, (2021).

[22] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al., 'Grandmaster level in starcraft ii using multi-agent reinforcement learning', *Nature*, **575**(7782), 350–354, (2019).

[23] Tsung-Yen Yang, Michael Y Hu, Yinlam Chow, Peter J Ramadge, and Karthik Narasimhan, 'Safe reinforcement learning with natural language constraints', in *Advances in Neural Information Processing Systems*, volume 34, pp. 13794–13808, (2021).

[24] Daochen Zha, Jingru Xie, Wenye Ma, Sheng Zhang, Xiangru Lian, Xia Hu, and Ji Liu, 'Douzero: Mastering doudizhu with self-play deep reinforcement learning', in *International Conference on Machine Learning*, pp. 12333–12344, (2021).

[25] Victor Zhong, Tim Rocktäschel, and Edward Grefenstette, 'Rtfm: Generalising to novel environment dynamics via reading', *arXiv preprint arXiv:1910.08210*, (2019).