2017 16th IEEE International Conference on Machine Learning and Applications

Applying reinforcement learning and supervised learning techniques to play Hearthstone

Ilya Kachalsky*, Ilya Zakirzyanov*†, Vladimir Ulyantsev*
* ITMO University, Saint Petersburg, Russia
{kachalsky, zakirzyanov, ulyantsev}@rain.ifmo.ru
† JetBrains Research, Saint Petersburg, Russia

***Abstract*—Hearthstone is the most popular digital trading card game nowadays. This paper addresses the problem of applying machine learning methods for the development of an agent, which plays Hearthstone. We show how to separate a large action space of Hearthstone in two parts, one of which is quite small, in order to apply reinforcement learning. Also we develop a new tree-walking algorithm which allows us to consider only the most promising game states of Hearthstone in order to determine the best action sequence on a given turn. Besides that, we propose a new way to train and build a game state metric using supervised learning techniques. We develop four agents based on these proposals. Two of them show better results compared to previous works. Lastly, we show that the best developed agent can compete with game experts.**

***Index Terms*—Machine learning, Supervised learning, Back-propagation, Artificial intelligence, Multi-layer neural network**

I. Introduction

Digital *trading card games* (TCG) are a fast growing video game genre, mainly because of the success of Heathstone, developed by Blizzard Entertainment [1]. Hearthstone is a multiplayer video game where one can play against a randomly chosen opponent or a friend. While Hearthstone might look simple on the outside, on the inside it is a very complex game with many strategies and decisions. Players who can successfully find the best strategy become professionals and compete for millions of dollars each year [2]. While keeping game fun enough for casual players, Blizzard Entertainment developed a new e-sport discipline.

Hearthstone is a combinatorially difficult, discrete, imperfect information game, featuring many complex interactions. Much game data is available due to the popularity of Hearthstone. In 2014 Blizzard published an infographic stating that two billion hours were played [3]. This amount of replay data (a *replay* is a recorded game) makes reasonable to use machine learning to solve such a complex task as playing Hearthstone.

Machine learning is a field of computer science that gives us opportunity to find and recognize patterns in an environment which even experts sometimes cannot notice. This brings us to the research questions of this work:

- Can we apply techniques of machine learning to the task of developing an efficient *agent* (a computer program which autonomously interacts with a game environment)?
- Can we improve results of previously suggested agents?
- How would such an agent perform against the experts?

II. Background and related work

A. Hearthstone

Hearthstone is a digital turn-based card game, in which one assembles a deck containing 30 cards and then plays against an opponent and their deck. Each moment of the game is a game state. An example of the game state is given in Fig. 1.

Each player is represented by one of nine predetermined *heroes* with an unique *hero power*. A hero has 30 *health points* (HP) at the beginning of the game. The goal of the game is to reduce your opponent's hero HP to zero. There are two types of cards in the game: *minions* and *spells*. Each player has a so-called *hand* – a set of cards which they can play on their own turn by paying resource called *mana*, and a *board* – a place where minions are placed. Minions have *attack value*, own HP and optional special properties. If a player plays a minion from the hand, it is summoned onto their board. A *spell* has effect, which can affect one's board, hand, or hero, if it is played. At the beginning of each turn a player gets a random card from their deck and adds it to their hand. After that, the player can play cards from their hand or use a hero power. Each turn

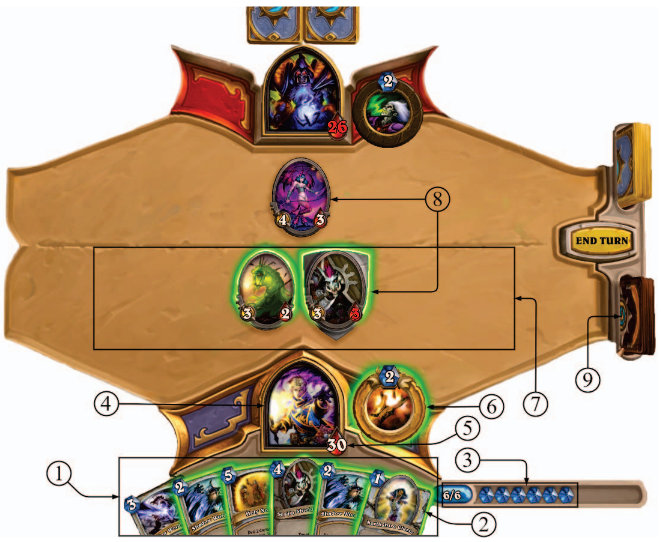

Fig. 1. An example of a Hearthstone game state. 1 – a hand with six cards, 2 – a card, 3 – available/maximum mana, 4 – our hero, 5 – our hero HP, 6 – a hero power, 7 – our board, 8 – minions, 9 – our deck. This picture is under trademarks and copyrights of Blizzard and its licensors, all

DOI 10.1109/ICMLA.2017.00016

one can use minions on their board to attack enemy's minions or hero. Essentially, playing Hearthstone comprises managing hero's HP and resources of the board and of the hand.

B. Machine learning

Reinforcement learning (RL) is a technique of machine learning in which an agent interacts with an environment by taking actions and getting back a new state of the environment and some instant reward. *Q-learning* is a method of RL aimed at finding ways to maximize the cumulative reward by learning an action-value function throughout the simulation (see, e.g., [4]). *Supervised learning* is a machine learning task, in which a model tries to develop dependences between training data and correct answers on some question asked about this data.

C. Previous work

In 2015 David Taralla developed an agent called Nora [5]. Nora was able to detect strategic moves and won 93% of games against a random agent (all actions are chosen randomly). However, against a heuristic agent Nora won only around 10% of the games. Our research utilizes the theory from this paper and the proof that the percentage of wins in Hearthstone converges at ten thousand games.

In [6], the authors suggested a new approach for agent building. They were able to successfully the heuristic agent by separating their agent into two models – an action tree and a metric. An action tree is a model where nodes are game states and transitions are actions. The goal is to find the best action sequence using the metric. Because each turn in Hearthstone lasts a maximum of 75 seconds the authors used a heuristic algorithm to consider only potentially profitable sequences. To develop the metric, they used supervised learning. Their agent managed to win 67.35% of games. This value was determined by conducting ten thousand games between their agent and the heuristic one.

III. Proposed approaches

Hearthstone is a turn-based game. Each player performs actions only within their turn. Thus each player's turn is an *action sequence* where the last action is to pass the turn to an opponent. In this paper the goal of an agent is to get an optimal action sequence. We propose four different agents consisting of two parts: a feature extractor and a sequence generator, which uses features obtained by the feature extractor. The feature extractor is common for all agents but the generator differs in each case.

A. Feature extraction

To solve this problem as a machine learning one, we need to extract features which would describe a game state in the best way but would not be too complex. The proposed features are summarized in Table I. A vector for each minion includes the following information: the minion's HP, its attack value, whether it can attack this turn and markers of three special properties (we take into account only the most common properties for simplicity). A player's vector includes the sum of their minions' attack values, the number of minions on their board, the number of cards in their hand and deck. Another feature is a turn number which allows us to identify a game stage. It is useful since some strategies heavily depend on it. The last feature is the current total available attack value – the sum of attack values of all player's minions which can attack at the current moment.

TABLE I
Features which describe a game state

Feature number	Feature description
0	Player's hero HP
1	Opponent's hero HP
[2,85]	14 six-dimensional vectors for minions
[86,93]	2 four-dimensional vectors for players
94	Turn number
95	The current total available attack value

B. Agents based on Q-learning

The Hearthstone action space can be divided into two parts: hand actions (playing a card, using a hero power) and board actions (issuing an attack). The attack action in Hearthstone is a two-phase action. First, one chooses a minion from their side of the board to attack with, and second, one chooses an opponent's hero or a minion from an opponent's board to attack with the previously chosen minion. Most of the time the board actions do not affect the hand actions, which enables us to divide the game process into two parts, one with a large action space and one with a small one. Thus the sequence generator can be split into the hand model and the board model.

The first agent, called Iana, is based on a board model with 57 actions. At any given time there could be up to seven minions on both players' boards. Thus, the maximum number of possible attack actions is $7 \cdot 8 = 56$. The 57th action is the opportunity for the model to do nothing. Which in our case means to pass the turn to the opponent. To learn which action to perform we use Q-learning. We use Q-function with multilayer perceptron as an approximation and the algorithm taken from the paper [7] for training it.

This board model is trained by playing against the heuristic agent from [6]. For the second agent, called Kulu, we separated the board model into two parts. The first part selects an attacking minion or passes the turn to the opponent. This part is trained in the same way as the Iana's board model. The second part chooses a target for the attack by the selected minion. It is trained on heuristic agent's actions using a supervised learning method. This way of designing the board model reduces the time used for its training and shows better results.

The hand model is the same for Iana and Kulu. In the beginning of the turn the hand model builds all possible hand action sequences. Then for each sequence the hand model asks the board model for the reward of this sequence execution. This reward is based on Q-values of possible board actions.

Then the best action sequence is executed. Finally the board model executes its own action sequence.

C. Agents based on action tree

Both agents from the previous subsection preform actions in a particular order: hand actions followed by board actions. This approach is not effective in certain cases. In this subsection we propose two other agents which do not distinguish actions by type. This allows changing the action order as needed. The sequence generator of these agents consists of two parts: an action tree and a metric. The action tree is a model where nodes are game states and transitions are actions. Using the action tree and a smart tree-walking algorithm, we produce a set of final states, which are then compared with each other using the metric in order to determine the best action sequence.

The action tree in Hearthstone is usually too big to process under the time limit of a turn. Thus an algorithm which processes only potentially profitable subtrees is needed. We suggest the following solution. In the beginning of tree walking we traverse one branch to the end and save all its states in the memory. After that, upon visiting a state, we compare it with the states from the memory. If the considering state is identical to or dominated by some saved state, we stop traversing the current branch, otherwise we continue. If the tree leaf is reached then all branch's states are saved and the traversal continues.

We suggest two different metrics to compare final states. The first one is based on the training on full chains of game states and is used in the agent called Vemi. As we play Hearthstone, we consider the resulting chain of states at the end of the game. Each state from the chain gets a certain rating based on the *rating function* $f(i, n) = \gamma^{(n-i)} \cdot R$, which is a simplified version of the function from [4]. Here γ is a discount factor, R is a final reward based on the result of the game, i is the number of state in the chain, and n is the length of the chain. Thus, the states which are closer to the end of the game get higher rating.

The other metric which we propose is used in the agent called Gela and is based on a *threat level*. We distinguish three threat levels: the green – low, the yellow – moderate, and the red – high. To decide to which threat level a state belongs we introduce a new parameter – *effective HP*. It is the difference between one's hero HP and the opponent's sum of minion's attack values. Using this parameter the game can be represented by the state machine with three colored states (green, yellow and red) and two terminal states (winning and losing). There are transitions between all colored states, between the red one and the losing one and between each colored state and the winning one. In this case we consider a chain and rate its states not at the end of the game but when the state machine moves to another state. This approach allows us to determine dependencies within the state space of Hearthstone better.

IV. Experiments and Results

To measure agent performance we use *win rate* – the percentage of won games. In all experiments we performed ten thousand games where both agents played with equal decks. The agents were trained using the deck from the work [6].

A. Agents based on Q-learning

For Iana's board model the instant reward for terminal states was set to n for winning the game and $-n$ for loosing. For other states it was set to zero. To approximate Q-function we used a multilayer perceptron. After the training process, Iana's win rate against the heuristic agent was 52.34%. After that we played a couple of games against Iana ourselves and found that it always chose to attack our hero instead of minions even when it was obviously suboptimal. In other words, the action to attack an opponent's hero always got the highest reward. This issue was caused by how we set instant rewards. In Hearthstone the most common way to win the game is to attack the opponent's hero with your minion. That is why the agent always chose these actions.

We resolved this issue by changing the way the instant rewards were set. Instead of giving a big reward only at the end of the game, we gave small rewards throughout the game and bigger ones in terminal states. They are based on the function $r(s, s') = \rho(s') - \rho(s)$, where $\rho(s)$ is the function of state from the heuristic agent, s is the previous game state, s' is the current one. After that change the actions resulting in attacking the opponent's hero got lower values compared to other actions in the most cases. The second version of Iana won 61.26% of games against the heuristic agent.

For Kulu (which is based on the board model with 8 actions) we used the same function for setting instant rewards. For the part of its board model which chooses a target for the attack we trained a neural network on data obtained from the heuristic agent games against itself. This network was used for the board model training. Kulu's win rate against the heuristic agent is 65.34%, which is slightly better than Iana's win rate. But the main advantage of this agent is the board model with fewer possible actions, which reduces the model training time almost in four times.

B. Agents based on action tree

To train Vemi's metric (which is based on full chains of game states) we used two heuristic agents which play against each other. After the game, the states are rated based on the rating function and added to the training set. To approximate the metric we used a neural network. As before, our agent played against the heuristic one and both agents used the same deck. We performed two experiments: with training deck and with ten random decks taken from the collection of Arena decks from Hearthpwn [8]. Vemi won 72.35% and 78.27% of the games using the training deck and the random ones respectively. This shows us that our agent uses cards which are not present in the training set better than the heuristic agent. It can be explained by the fact that Vemi is based on the non-greedy tree walking algorithm.

To train Gela's metric (which is based on the threat level) we also used two heuristic agents, but the game states were rated after each change of the threat level instead of doing it at the end of the game. For each of three threat levels we trained a neural network. Gela won 82.24% and 84.35% of games against the heuristic agent using the training deck and the random ones respectively. As it can be seen, the augmentation of the model by the threat level and using this feature in the decision making process helped the agent to perform better depending on the current game situation. For example, it allows Gela to make better decisions in the early stages of the game.

C. Comparison with previous work

To compare our agents with the agent proposed in [6] we performed ten thousand games between them using the same training deck and the random ones as in the previous subsection. The results are presented in Table II. As we can see, the agents based on Q-learning (Iana and Kulu) did not improve previous results. On the other hand our agents based on action tree (Vemi and Gela) outperformed the baseline agent. It can be explained by the new, more effective tree walking algorithm and different approaches in the development of the metric.

D. Play testing

Finally, we performed an experiment of the best performing agent – Gela – in which we performed one hundred games against game experts (each won more than three thousand games in Hearthstone). Each expert played ten games against our agent using the training deck and ten games using random decks (the agent and experts used the same deck). The agent won 50% of the games using the training deck and 42% of games using the random ones. Although these results are not entirely reliable due to the small sample size, they show that Gela performed almost perfectly with the training deck. While playing with random decks, we discovered that in some cases the agent incorrectly uses cards with special properties, which cannot be observed by our feature extraction model.

V. Discussion and conclusions

In this paper we proposed a new way of applying machine learning to play one's turn in the digital trading card game Hearthstone. First, we introduced the way to apply Q-learning. A common turn in Hearthstone has a very large action space. We solved this problem by proposing two models. The board model is responsible for attacking actions and is trained using Q-learning. The hand model is responsible for playing cards from the hand and makes decisions based on the board model. The disadvantage of this approach is the predetermined order of actions. Second, we proposed a new way of traversing an action tree, which allows us to consider branches leading to better action sequences (which did not hold for the previously suggested approaches due to the potentially big size of the tree), while keeping computational time under restrictions of Hearthstone. Also we improved the previously described algorithm of training the metric for determining the best action sequence. These proposals allowed us not only to significantly improve the agent's performance comparing to the previous works, but also play against game experts on the acceptable level.

This brings us to the answers for our research questions:

- We applied techniques of machine learning to develop an agent by proposing new ways of extracting features and structuring agents.
- We improved the results of previously suggested agents by introducing a new tree walking algorithm and a new way of training the metric.
- Our experiments show that the developed agent can perform against game experts. However, the experiments also show that in some cases our feature extraction is insufficient for cards with special properties.

In future work we would like to surpass the previously discussed problems of using cards with special properties properly. Also, we plan improve the performance of the agents by training them on replays obtained from games between real players.

TABLE II
COMPARISON WITH THE AGENT FROM [6]

Agent	**Training deck**	**Random decks**
Iana, based on board model with 57 actions	35.25%	32.21%
Kulu, based on board model with 8 actions	45.48%	44.79%
Vemi, based on full chains metric	56.97%	59.55%
Gela, based on threat level metric	67.23%	72.90%

Acknowledgments

The authors would like to thank Arip Asadulaev, Igor Buzhinsky, Andrey Filchenkov, Sergey Muravyov, Andrey Mavrin for useful comments. This work was financially supported by the Government of Russian Federation, Grant 074-U01.

References

[1] "Hearthstone official game site." https://eu.battle.net/hearthstone/en/. (Accessed on 10/07/2017).

[2] D. Gehlen, "Hearthstone: Heroes of warcraft prize pools & top players – esports profile: e-sports earnings." https://www.esportsearnings.com/games/328-hearthstone-heroes-of-warcraft. (Accessed on 20/07/2017).

[3] "Hearthside chat - mobile development update - hearthstone." https://us.battle.net/hearthstone/en/blog/16421347. (Accessed on 08/01/2017).

[4] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[5] D. Taralla, *Learning Artificial Intelligence in Large-Scale Video Games: A First Case Study with Hearthstone: Heroes of Warcraft*. PhD thesis, Université de Liège, Liège, Belgique, 2015.

[6] J. Zhu, "Will our new robot overlords play Hearthstone with us?," *CS 229 Final Report*, 2016.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[8] "Hearthstone arena decks." http://www.hearthpwn.com/decks?filter-is-forge=1&filter-deck-tag=1. (Accessed on 15.07.2017).