Beating Onitama with Reinforcement Learning: Interim Report

G118 (s1728664, s1707653, s1714441)

Abstract

Onitama is a 2 player board game played on a 5x5 board. Each player has 4 pawns and a king piece. To win the game, a player must either take the opponents king, or have their own king land on their opponents "Temple Arch Square". The moves allowed are dictated by a set of cards drawn at the beginning. The current state of the art for beating this game is an AI that uses negamax search, along with several other techniques used to limit the search space. In our project, we propose a more modern AI approach using reinforcement learning. This method allows an agent to learn what moves to make based on the board, player, and card state, using a cumulative reward over time. We use a variant of RL called Deep Q Learning (DQN) that uses convolutional layers in its model. To train the model, we will first play our agent against a simple agent that adheres to a set of simple rules declared by us, then move on to training two RL models against each other (self play), then try to train our RL model against the negamax search AI in order to beat the state of the art.

1. Introduction

With the emergence of modern machine learning techniques, many traditional AI approaches to board games (such as Stockfish for chess (Stockfish, 2020)) are becoming bested by reinforcement learning models that allow a machine to learn for itself how to beat a player, using reward based heuristics, such as DeepMinds AlphaZero.

Following the success of these models, we propose an RL approach to the strategy board game Onitama. The current state of the art (SOTA) is a negamax search that inspects the search space for the next move that will most likely lead to a win (oni, 2019). We hypothesise that training a model by maximising the cumulative reward for an agent will lead to more consistent victories.

To do this, we will use a Deep Q Network (DQN) (Mnih et al., 2015). Our inputs will be a representation of our environment, and our each of our outputs will represent an action taken.

1.1. Game Rules

Onitama is a 2 player board game played on a 5 by 5 grid. Each player has 4 pawns and a king, that are set up on their side of the board with the king in the middle and 2 pawns on each side. The middle back square that the king starts on is known as the Temple Arch square.

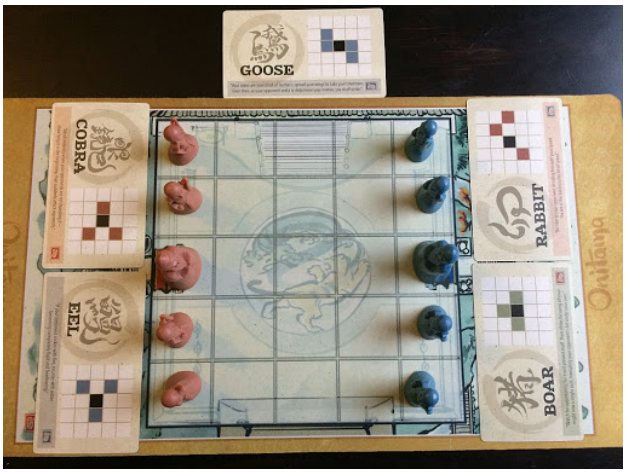

Figure 1. Photograph of Start of Game. (Quest, 2016)

5 cards are drawn from a deck of 16 cards. Each card contains a unique set of possible moves relative to a piece's position. These 5 chosen cards will be used throughout the game: 2 cards per player, and a spare card put to the side.
The colour of the spare card indicates which colour player starts the game (red or blue).
Which ever player goes first plays one of their cards and makes a move on any of their pieces that is allowed by the card they played. This played card then swaps with the spare card to become the new spare card.
To win the game, a player must either take the opponents king (by landing any piece on their king's square), or by landing their own king on the opponents Temple Arch Square.

What makes this game interesting is that the 5 cards in play throughout the game, chosen at the start, dictate completely how the game will be played, and in a way make every game unique.

We hope that this project can contribute to the Onitama community by becoming the SOTA for Onitama AI, through consistently outperforming the current negamax model.

2. Data set and task

We have coded a virtual environment for the game in Python, including an interface where the game can be

played human player vs. player, player vs. bot and bot vs. bot.

There is an official Onitama app with 3 levels of difficulty. A novice human player can beat the easy difficulty setting after a few attempts. There also exists an intermediate and hard difficulty. There is a comprehensive implementation of negmax search with alpha beta pruning which beats the highest difficulty setting of the game (Wonders, 2014). Our goal, as described, is to beat the current best AI at the game and as such achieve human level performance.

To evaluate our results we have a series of metrics. At first we have a random agent that takes a random valid move each turn. This is for sanity checks and testing.

For initial training we wrote a simple heuristic agent. The simple agent looks one move ahead and can find wins, avoid losses and does a good job of capturing pawns and avoiding pawn losses. This agent has significant drawbacks but in order to beat it an agent must learn the dynamics of capturing pieces.

We hope then to look into self play and setup an environment with the negmax Onitama AI to get the best RL possible.

Another question which is interesting for Onitama is whether an agent can play with new cards it has not seen before. Onitama comes with 16 cards but many more could be made. Whether an agent could still perform at a high level with cards never seen before is another possible threshold.

3. Methodology

Since we are using discrete action and observation spaces, we chose to use DQN (Mnih et al., 2015). Whilst it is not the state of the art technique, it is relatively reliable and well implemented and tested versions are available online. We use the Stable Baselines implementation (Hill et al., 2018).[1]

At the moment we use the standard DQN but we are looking into extensiosn to improve performance, such as double Q-learning (van Hasselt et al., 2015) and prioritised experience replay (Schaul et al., 2016).

Since the board description have a grid topology, we make use of convolutional neural networks (CNN) (LeCun et al., 1989) as the Q-function approximators in DQN, as per the original work. This allows us to gain the advantages of sparse interactions, parameter sharing and equivariant representations (Goodfellow et al., 2016).

We follow the approach of (Silver et al., 2017) in masking the output action probabilities (in the DQN case this is the Q-values) and then re-weighting before the softmax which then outputs actions. This required us to implement the masking by making significant changes to the Stable Baselines code. In particular we had to apply masking also to the ϵ-greedy exploration sampling of actions.

Our observation space thus consists of two parts: the observations and the action mask. The masking matches the actions shape.

The observations $(5 \times 5 \times 9)$ consist of a description of the game state:

- $(5 \times 5 \times 2)$ the cards in player 1's hand
- $(5 \times 5 \times 2)$ the cards in player 2's hand
- $(5 \times 5 \times 1)$ the next card
- $(5 \times 5 \times 1)$ the position of player 1's king
- $(5 \times 5 \times 1)$ the position of player 1's pawns
- $(5 \times 5 \times 1)$ the position of player 2's king
- $(5 \times 5 \times 1)$ the position of player 2's pawns

The outputs of the model are $(5 \times 5 \times 50)$. The first two dimensions represent locations in the board to pick up a piece (ie. which piece to move). Then the last dimension can be thought of as a flattened $(5 \times 5 \times 2)$, which represents where on the board to place the piece (ie. where to move to) and which card that move is chosen from (out of two possible). The masking inputs are also of this shape and can then be used to mask over invalid moves before flattening and computing a softmax. The final output action for the environment is then a single discrete number $[0, 1250]$.

4. Experiments

To assist training we added intermediate rewards. The rewards used were 1 for a win, 0.05 for a pawn capture and 0.005 for a forward move with a piece. We also tried using 1 for a win and 0.1 for a pawn capture. As this led to significantly slower training and worse results we did not train a model only rewarding wins. These rewards are negative if the opponent achieves them.

The rewards added were:

To begin with, we trained an agent on a simplified version of the game where all games have the same 5 initial cards. The rl agent played against our simple agent for 1,000,000 time steps with evaluations every 500 time steps. Training has not proved too intensive in terms of hardware and we have been able to train in a few hours on a standard personal laptop. This will likely get more intensive as we expand to all cards, train longer and include self play.

Rewards	Mean Reward	Final Result
Forward Moves	0.2158	62/100
Only Captures	-0.164	42/100

Table 1. Results from experiments run with differing intermediate rewards against the simple agent.

[1]The code for our project is available at https://github.com/Ollie-Gustav-Tim/MLP-CW2/tree/ollie/python/onitama-py

Our initial results are very promising. With the rl agent succeeding to beat the simple against in 62/100 games with the first set of rewards.

5. Interim conclusions

Our results demonstrate that our environment works, and that DQN can learn to outperform a single agent. Given a stronger machine and longer training we are confident that the rl agent will reach a higher winrate on the simple agent. The differences the intermediate rewards have on training shows they have a significant effect on rate of convergence and whether the rl agent can learn at all.

6. Plan

As described above we have made promising progress in tests with our simple heuristic agent. We are now looking into self play as a means to further improve our DQN model beyond the level of the simple agent. In order to effectively evaluate the RL, we will need a better benchmark so we are improving the heuristic model and looking at how to integrate the current best Onitama AI (oni, 2019) into our system.

More experiments into tuning the intermediate rewards are needed. Currently the simple agent makes no attempt to win by reaching the opponents Temple Arch Square (unless it is possible on the next move) and does not try to stop the opponent king from achieving this. As a result we may need to improve the simple agent such that it is a stronger opponent with regards to this win condition

One potential risk is that we do not have much experience with self-play and it can be quite a complex and intensive method of training. The backup plan would be just to focus on beating hard coded agents and trying to get the most performance we can from that method of training.

References

Onitama ai. https://github.com/maxbennedich/onitama, 2019. [Online; accessed 16-February-2020].

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Hill, Ashley, Raffin, Antonin, Ernestus, Maximilian, Gleave, Adam, Kanervisto, Anssi, Traore, Rene, Dhariwal, Prafulla, Hesse, Christopher, Klimov, Oleg, Nichol, Alex, Plappert, Matthias, Radford, Alec, Schulman, John, Sidor, Szymon, and Wu, Yuhuai. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541. URL https://doi.org/10.1162/neco.1989.1.4.541.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL http://dx.doi.org/10.1038/nature14236.

Quest, Cardboard. Onitama review. http://www.cardboardquest.co.za/review/onitama-review/, 2016. [Online; accessed 16-February-2020].

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay, 2016.

Silver, David, Hubert, Thomas, Schrittwieser, Julian, Antonoglou, Ioannis, Lai, Matthew, Guez, Arthur, Lanctot, Marc, Sifre, Laurent, Kumaran, Dharshan, Graepel, Thore, Lillicrap, Timothy, Simonyan, Karen, and Hassabis, Demis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

Stockfish. Stockfish 12. https://stockfishchess.org/, 2020. [Online; accessed 16-February-2020].

van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning, 2015.

Wonders, Arcane. Onitama. https://www.arcanewonders.com/game/onitama/, 2014. [Online; accessed 16-February-2020].