Learning to Play Othello using Reinforcement Deep Learning Networks

Xuecong Fu
Department of Biological Sciences
Fudan University
No. 220 Handan Rd.

Luyang Huang
Department of Physics
Fudan University
No. 220 Handan Rd.

Abstract

The deep neural networks are proved to be efficient and effective when solving problems which cannot be easily tackled with usual methods. And in some artificial intelligence task, the reinforcement learning is widely used to learn a best policy. In this project, we aim to combine deep neural networks with the reinforcement learning to tackle a popular game, Othello, which is an adversarial and zero-sum game. We used deep q learning methods in general, adjusting the networks and adding some convolutional network structure as well to figure out whether this algorithm works on the Othello instead of the traditional empirical methods. We also tried different opponents during training to see whether the opponents also influence the result.

1 Introduction

There are a lot of different games which we usually play in our leisure time. Some with a determined reward of penalty after each specific and limited steps while others have numerous possibilities. And usually we also receive a final reward or penalty after each game. The difficulty of the games lies in what we need to find out is the overall best policy which receive a largest rewardoverall but not the policy which follows the greedy algorithm. For the past years, there were plenty of research on this kind of games, including the one with most reputation and also the complexity, AlphaGo, as well as simpleand interesting games like Five-in-a-row, Othello.

Withthe arising of the artificial intelligence, especially the reinforcement learning, researchers started to study on the ways to solve simple games by reinforcement learning, from the value iteration, policy iteration to Q-learning, SARSA and TD-learning methods. But for complex board games, the traditional reinforcement learning doesn't work. Due to the large searching space they need, these board games are quite impossible to achieve the best methods through search algorithm. Therefore, the deep learning was introduced to the area, since people found that the neural networks could approximate any formula. Another advantage of the deep learning is that it has no need to know the underlying mechanism but put the reward and the input into the network and train it.

During the training of reinforcement learning, an agent needs to play a large number of training games. In this report, we want to figure out whether different changes on the algorithm work. We also want to know whether the different opponents will influence the final result. Therefore, we tried a plenty of improvement on the reinforcement learning algorithm and changed the opponent of the training to prove our assumptions.

The reinforcement learning is the core of our research. However, the deep network we use can vary because of different games we are solving and the different characteristics they perform in different situations. Apart from the multi-layer perceptron, we also tried the convolutional

neural networks to be connected in order to extract features of the situations on the board.

2 Othello

Othello is a perfect information, zero-sum, two-player strategy game played on a board of 8 by 8. And the rules are simple. The white and black player place a position on the board alternatively. And the beginning status is two white and two black in the center, each color occupies a cross line. A move is valid if the newly placed disc causes some opponents discs to become enclosed. The enclosed discs are then changing their color to the player's. if a player cannot make any of the opponents' disc enclosed, he/she passes. When the board is full of the discs, the game is finished. The victory belongs to the player with larger number of discs. A draw is declared if the number of discs of each color are equal.

Despite a simple and easy game, the game of Othello is far from trivial because the number of legal position is 1028, so it's impossible to use search methods to solve this problem. Hence in this project, we wish to improve the way of learning to win the Othello game by using reinforcement learning methods. Due to the simple q-learning,TD-learning and SARSA have the similar results according to the articles, we tried the q-learning methods to tackle the task. In order to detect the collaborative information within the adjacent square in this game, we also introduced the convolutional neural networks to extract the local features.

3 Reinforcement Learning

Reinforcement learning is a subset of machine learning, with the characteristics of learning from interaction. An agent learns from the environment and gain information through the reward and punishment. After each step of action which the agent makes, the learner receives a reward or penalty for its action and learns to act better according to the environment. The aim of the reinforcement learning is to learn a best policy so as to achieve the highest total rewardor other objectives through adjusting the policy each time according to experience. The policy is a sequence of actions or a set of probabilities to do certain actions given the current states. The most important problem of the reinforcement learning is the contribution allocation problem because the useful information can be only obtained after a set of actions through the whole model. Therefore, it contains some delay.

Although reinforcement learning is a subset of machine learning, it is different from supervised learning on that reinforcement learning aims at maximize the expected reward but not a correct policy. And the reinforcement learning do not need knowledge about the Markov Decision process.

3.1 State-Value and State-Action Value Function

The value of a polic $\pi, V^{\pi}(s)$, is defined as

$$V^{\pi}(s) = E(\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, \pi)$$

which is the expectation of total reward given a initial state and a policy. We define a coefficient of discount γ to reduce the weight of long term reward. Because in some situations, the more action the policy contains, the larger rewards it will gain, which may lead to an unintelligent policy with countless actions.

To better optimize the policy, we introduce the state-action value function, $Q^{\pi}(s, a)$, which is the expectation of total reward given an initial state, a policy and an action we take.

$$Q^{\pi}(s, a) = E(\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi)$$

The state value function indicates how good it is for agent to stay at the state s, while the state action value function indicates how good it is for agent to perform the action a if staying at

the state s.

There are many ways of reinforcement learning including Policy Iteration, Value Iteration. These methods are model related and the state transition probabilities and reward function of the Markov Decision Process need to be available. Another problem is the large complexity, which lead to low efficiency of the algorithm. For example, in Go, there are 3^{361} types of states. Therefore, a function (usually a neural network) is introduced to approximate the value function to reduce the complexity, the same as what we introduce in the following parts. Another category of the reinforcement learning is called model free reinforcement learning, which includes Monte Carlo Sampling and –Temporal Difference Learning, SARSA, TD learning and so on. In the following part, I will introduce the Q-learning algorithm which we used in the task and how we combined it with deep neural network.

3.2 Q-learning

Q-learning is a model free reinforcement learning technique. The core is to learn an action-value function which is denoted by $Q(s,a)$. It is a kind of Temporal Difference Learning method which combines the dynamic programming with the Monte Carlo Sampling Method. In order to be more efficient, we use the Bellman Equation to estimate the current total reward.

$$Q_T^{\pi}(s,a) = (1-\alpha)Q_{T-1}^{\pi}(s,a) + \alpha(r_T + \gamma Q_{T-1}^{\pi}(s',a'))$$

In which the a' is got from the $\pi(s') = argmax_{\alpha \in |A|} Q(s,a)$

Therefore, in the Q-learning algorithm, the strategy of renewing the parameter is presented by

$$Q(s,a) = Q(s,a) + \alpha(r_T + \gamma Q(s',a') - Q(s,a))$$

3.3 Deep Q-learning Networks

In order to solve the problem of large complexity of the state space in board games like Go, or the continuity of the states and actions in automatic driving, we usually approximate the value function or strategy with a deep neural network to reduce the complexity and improve the ability of generalization.

In deep Q-learning networks, we use a deep neural network to approximate the value function. $Q_\theta(s,a)$ is a deep neural network.

$$Q_\theta(s,a) = Q^{\pi}(s,a)$$

We use the loss function of $L(s,a,s';\theta)$ to minimize the loss.

$$L(s,a,s';\theta) = (r + \gamma max_{a'} Q_\theta(s',a') - Q_\theta(s,a))^2$$

In order to solve the problem of unstable objectives or the relativity among the samples, we used the Deep Q Networks which is created by Mnih. We adopted the method of freezing the target networks, which created two networks and renewed one of them within particular steps. Another improvement was creating an Experience Replay to reduce the relativity of the samples. Each step we randomly get one sample from the experience pool to renew the network.

3.4 Application to Othello

The important thing we need to notice is that, in the application to Othello, our state which our value function relies on, is the previous state before we take moves. Therefore we implement such steps:

1) Initialize the experience pool, the Q network θ and the target Q network θ^-
2) Observe the current state s_t
3) For all possible actions a_t' in s_t use Q target neural network to compute $Q(s_t, a_t')$
4) Select an action a_t' using the policy $\pi(s') = argmax_{\alpha \in |A|} Q(s,a)$, get the simultaneous

reward r

5) Put the tuple (s_t, a_t' ,r) into an experience pool D with length d

6) Sample *(s,a,r)* from D

7) Compute target value of previous state-action pair $Q^{new}(s_{t-1}, a_{t-1})$ according to the formula

$$Q^{new}(s_t, a_t) = r_t + \gamma max_a Q(s_{t+1}, a)$$

8) Use Q neural networks to compute the current estimate of the value of previous state-action pair $Q(s_{t-1}, a_{t-1})$

9) Adjust the neural network by back propagating the error (loss function)

10) Execute action a_t

11) After c steps, $\theta^- = \theta$

4 Experiments and results

In training our learning agents, we try different networks. First, we use feedforward multi-layer perceptrons with 2 hidden layers as function approximations. The first layer contains 128 hidden nodes and the second layer contains 64 hidden nodes. The input is a vector is a [1, 257] dimension vector. States are represented by a 3*8*8 matrix , each 8*8 matrix corresponding to a square on the Othello board. The first is representing the black states, with 1 on the place where the black discs are placed. The second 8*8 matrix represents the white discs' position and the third represent the blanks positions. We reshape the state of the board matrix of 3*8*8 and to a [1,192] vector. We also reshape the action from the valid moves to one hot vector and concatenate these two vectors so that the input becomes 257-dimension vector. The output is a one dimension vector, which is the estimated Q value function. We use the rectified linear units (ReLU) as the activation function because it can avoid the vanishing gradient problems. The final outcome is the estimated Q value.

The reward associated with a terminal state is 1 for a win and -1 for a loss. The discount factor is set to be 0.8. the probability of exploration is 0.3. It didn't decrease so that it can allow more possibilities of other situations. The learning rate is set to 0.001 for Q-learning. Due to the port restrictions, we cannot use TD-learning because we need environment to be an input of the placement policy in the game.

We use the squared difference of the estimation of Q value and the target value to be the loss function. It's better than softmax function because it is not a classifying problem.

In our training, we train the model with different opponents, random player, heuristic player and itself. The random player randomly chooses an action from the valid moves. The heuristic player has a table attributing values to all squares and it chooses moves which maximize the positional values (Figure 1) of the states after the move is executed. The values is calculated by the following function:

$$V = \sum_{i=1}^{64} c_i v_i$$

c_i is 1 if the square i is occupied by player's own disc and where c_i is -1 if square i is occupied by opponent's disc.

100	-25	10	5	5	10	-25	-100
-25	-25	2	2	2	2	-25	-25
10	2	5	1	1	5	2	10
5	2	1	2	2	1	2	5
5	2	1	2	2	1	2	5
10	2	5	1	1	5	2	10
-25	-25	2	2	2	2	-25	-25
100	-25	10	5	5	10	-25	100

(a)

Figure 1. the positional values used by player Heuristics trained using co-evolution[3]

The self-play mode is simply defined as the principle that opponent share the same policy making strategy and model parameters of Q-value with the trainer. But the opponent's moves are not learned. Only during the trainer's turn will the weights and bias be renewed.

In order to improve the multi layer perception network, we add the experience pool to sample the different state-action-reward pairs in order to reduce the connection among the samples. We also build two Deep Q networks so that one is the network which we train and renew weight simultaneously. The other one is the target network which copies the parameter of the training network after certain steps.

To extract the features of the board information and details of the situations, we try to add the convolutional neural networks instead of the fully connected layer. We build a 4 layer convolution network to extract the information on board. The first hidden layer has 3*8*8*64 feature maps. The input is a three-channel 8*8 matrix with each channel of the matrix representing current position information of the black discs, the current position information of the white discs and the places of the valid action afterwards. Each unit in a feature map receives data only from a 3*3 receptive field. And the kernel is what we are going to learn. We also use the ReLU activation function

Our basic structure of the convolutional deep Q network is network composed of 4 convolutional layers shown below.

Conv: conv64→conv64→conv128→conv128→fc128→fc64→fc1 (output)

Since the board size is not as big as Go, we use zero padding on the sides and use the stride 1 parameter. We also avoid pooling layer because of the limited information. We don't want to lose any important details which usually lose in max pooling. The stack of convolutional layers are followed by two fully-connected layers one with 128 nodes and the other with 64 nodes, and then output the estimation.

We train 20000 games each and compare the performance among different methods and different opponents. We found that it's better to train the model while the opponent is random player. The Heuristic player will cause severe over-fitting. During the training, the loss start to reduce early but the accuracy isn't going down. In the end, the strategy will be stuck into one policy, leading to 0:200 or 200:0 while testing for 200 games if trained for large epochs. And the accuracy is also not good. While training with itself, it might get into local optimization or lack of the samples. Therefore, the training with itself is not obviously better. So afterwards, we still use the training with random, which will lead to larger variety of samples.

We also compare the Deep Q learning algorithm of multi layer perceptrons with one of CNN. We find that the pure Q-learning algorithm is much better, which indicates that convolutional network perform not well on the Othello combined with reinforcement learning.

Table 1: Performance of the learning algorithm with when tested versus player random

Train vs.	Q-learning	CNN + Q-learning	CNN + Q-learning + dropout
Random	0.78	0.65	0.63
Heuristics	0.54	0.48	0.44
itself	0.88	0.66	0.64

Table 2: Performance of the learning algorithm with when tested versus player ab

Train vs.	Q-learning	Q-learning + dropout	Q-learning + CNN + fc + dropout
Random	0.15	0.075	0.1
Heuristics	0.12	0.00	0.00
itself	0.20	0.20	0.075

We also try to figure out whether the improvement of Deep Q Networks, the introduction of experience pool and the freezing of the target Q network work. After training with the random player, we find the improved Q Networks is slightly better than the original version with the accuracy of 27% in the original Q network, 31% in the Q network with freezing target network and 41% with both freezing target network and the experience replay. We renew the target Q network after every 10 turns of the game. The size of experience pool is 30. The test opponent is the TA's program on the ftp (10.141.208.101) The testing game number is 70. So the experience replay and the freezing target network do work in cutting down the connections among the different state-action samples and stabilizing the objective function.

5 Conclusions

In this report, we have compared different structures of networks of Q-learning to learn to play Othello game. We have also figure out the influence the opponent will have during training on the result.

The result suggests that, the adjusted Deep Q learning Network using multi-layer perceptrons is so far the best model. And if you want to have a model with which you'd like to win an opponent you don't quite know, it is better to use random player as the opponent in training. Also, the convolutional neural network might not be a good choice while using reinforcement learning to learn to play Othello.

We planned to use the database of Othello on the Internet, but it cannot be open through softwares like Cassio, by which the database ought to be open. Therefore, further studies like using database to train efficient models are expected. Another expected improvement is to try RNN with each game because the reward is given only the game is over. So it's likely to use RNN to train a model. If there is enough database, it will also greatly improve the accuracy of current model if we can get a reward based on the current state after each step. Or we can estimate a simultaneous reward by a function or a neural network.

Acknowledgments

I'd like to thank the teaching assistant for giving me help on this task.

References

[1] Michiel van der Ree and Marco Wiering. Reinforcement Learning in the Game of Othello: Learning Against a Fixed Opponent and Learning from Self-Play, *IEEE International Symposium on Adaptive Dynamic Programming & Reinforcement Learning*. 2013: 108-115.

[2] PawelLiskowski, WojciechJaskowski, Krzysztof Krawiec. Learning to Play Othello with Deep Neural Networks*: IEEE Transactions on computational intelligence and AI in games*. 2017: arXiv:1711.06583.

[3] S. Lucas and T. Runarsson, "Temporal difference learning versus coevolution for acquiring othello position evaluation," in Computational Intelligence and Games, 2006. *IEEE Symposium* on, 2006, pp. 52–59.