Reinforcement learning - TicTacToe

Saurabh Burewar (B18CSE050)
Deepak Arjariya (B18ME021)

Why RL here?

Reinforcement learning does not presume a model of the opponent. Without making any model of the environment or conducting any possible future searches for actions or states, the agent is able to plan ahead. This saves a lot of computation and exploration of large search space.

In reinforcement learning, the opponent is a part of the environment for the agent which it can interact with. So, instead of a model or neural network or a search like minimax, the agent can achieve the skill by simple trial and learning.

This becomes useful in adversarial games, allowing us to train an AI player with less computational power as compared to Neural networks or search algorithms.

Implementation

Implementation is done in three classes: State, ComputerPlayer and HumanPlayer class.

State

The state class acts as the board as well as the judge. The three major components that make up this algorithm are state, action and reward. The state is the state of the board which is essentially a 2D matrix, the action is the moves made by the players and reward is given at the end of each game.

- This is responsible for recording the board status and updating the board after every action of the players.
- We need a way to store the board in a state value dictionary which we can use to store our policy. For that, it hashes the board state.
- We update the cell of the board with 1 if player 1 takes the action and with -1 if player 2 takes the action. The rewards are given at the end of the game.
- After every move, it checks if either player has won. If yes, it ends the game and declares the winner. If the board is out of empty spaces, the game ends in a draw.

- It has a reward function which gives reward +1 to whoever player has won. And if the game is drawn we give less reward for player1 and more reward for player 2 something like [0.1, 0.6].This distribution has little problem that we will talk about in the next sections.
- The training is also handled by the state class. The training is performed between two computer players playing against each other. We have trained our agent for 200000 iterations.
- It also initiates the game when playing human vs computer, where the computer is the trained RL agent and human is the user.

Computer Player

The Player represents our agent and has everything responsible for choosing actions, recording the state of the board, updating the state-value estimations and save and load policy.

- Uses policy found after training for deciding the next best action.
- For choosing the best move to play, it loops through all the available positions on the board and checks the hash of it, if that was the next move. Then, it takes the max of them and selects it as the best action.
- While choosing, we need a balance of exploring and greediness. Here, we define the agent taking greedy action 70% of the time and taking random actions 30% of the time.
- When the agent reaches the end of the game, the estimates are updated.
- We store our estimations to form our policy which we load when playing the game after training.

Human Player

The human player is the user, so we take the input for the row and column for the user's next move, check whether that position is empty and update the board accordingly.

Training

The state handles the board training and the player handles our agent and learning, so we can now start our training process. We put two players against each other for training.

- The game is allowed to run for a certain number of times so that the agent can learn the policy after every iteration.
- During training, we use the epsilon greedy method , in which the player is allowed to explore with certain probability and rest of the time it uses the best available action from the policy.
- Less reward is given to the agent if the match is drawn so that agent avoids that path.

- To update value estimation of states, we will apply the following value iteration
  $$R(s_t) = R(s_t) + \alpha(\gamma * R(S_{t+1}) - R(S_t))$$
- The rewards are updated in reversed fashion when the agent reaches the end state
- The training can be performed using two RL agents.
- We can also use a RL agent against a minimax player but this will cause the RL agent to always lose matches during training making it impossible for it to learn.
- One improvement upon this is using a minimax player that chooses a random move or a minimax move with 50% probability.
- Another improvement to be done is using a combination of easy opponents (random opponents or another untrained RL agent) and hard opponents (minimax, models).

Play

Now that the training is done, we can move to actually testing the agent by playing the game.
The agent which is trained for 200000 iterations against another agent performs moderately. One noticeable difference here is that the agent focuses more on maximizing its profit rather than minimizing the opponent's profit. The reason for this could be the distribution of rewards in draw position, since agent get little reward in draw position in comparison to winning one so agent sometimes avoids defending itself and only work for maximizing its profit. When trained against a minimax player, the RL agent performs worse since the learning is less.

The game can also be played “human vs human”.

Future Improvements

- We can train the agent against an opponent that takes random as well as optimized actions based on probability.
- We can train the agent on a mix of opponents where some are easy while some are hard opponents. This will allow the agent to learn better.
- Working with different reward values to optimize results.

References

1. https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/
2. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
3. https://medium.com/analytics-vidhya/tic-tac-toe-in-the-age-of-reinforcement-learning-part-1-f85fc19229b1

4. https://towardsdatascience.com/reinforcement-learning-and-deep-reinforcement-learning-with-tic-tac-toe-588d09c41dda

Code

https://colab.research.google.com/drive/11ceW5x3RwRqLQGbjMgMaqFbzZCWGS5ru?usp=sharing