Tic-Tac-Toe with Deep Multi-Agent Reinforcement Learning

Zhi Wen Huang
zhiwenhu@buffalo.edu

Weijin Zhu
weijinzh@buffalo.edu

Abstract

We approached a stock market problem of getting profit, by modeling this scenario as a game Tic Tac Toe using multi-agents. The game tic-tac-toe, a 3x3 board is our environment which allows agents to determine how to play their game. Using deep neural networks, we are able to teach agents to learn the game and allowing them to become experts as tic-tac-toe player. With multiple agents learning to maximize their own performance and only one can win the game, there will be hard for both agents to be at their best performances at all time. We gave two situations to overcome this problem. One is to use Minimax algorithm to maximize one agent and minimize the other. This approach will help us teach one of the agents to learn how to win the game while other will keep losing. Second we want to set the maximization to a tie where no one wins. Both agents will learn to take move optimally so that no agent will always win. Our key goal is to let agents to learn within themselves and allow them to make decision wisely by looking at each other's maximum output action. We applied a value-based approximation method – Deep Q-Network. Two agents are able to first learn how to make a move, then how to win the game, and eventually both agents can tie the game. The point we want to revel is not only the game itself, but furthermore we want to make connection between the idea behind the game with current stock market.

1 Introduction

In order to play any type of board game, there are always some sort of strategy into it to let you become a better player. Everyone at their young age had come across tic-tac-toe and sometimes lose, sometimes win, and many times it could be a tie. As we play more and more of the game, we seem to notice a certain pattern of the game which makes every game a tie if you play the right move.

Have you notice, going first will allow you make an initial move that it gave you a higher chance to win the game? Will yes, theatrically in a board game like tic-tac-toe, going first will initiate your first move and will allow you to plan your game plan before the second player that goes second. However, playing the game well, the second player can always tie the game if they know your game plan in the beginning.

Tic-tac-toe is not a big board, we can easily hand written the entire win, lose, and tie state condition for all possible outcome. In this problem, we want our agents to learn to play the game and to learn all different possible outcome instead of we supervised them to force them play into one of the three outcome. The challenge is that teaching two agents facing each other to reach the maximum of both sides starting with no knowledge of the game.

The task cooperates with the environment and updating the states each time one of the agents takes an action. Then observing the board, they will decide their next action base on their previous experience. Every new game will be saved and bring up to depending on the states of the game. Our goal is to have both agents play and learn to their maximum potential and neither of the agent will play in their minimum valued actions.

One approach for the two agent to play against each other is to let one of the agent always play to win such that it focusses on getting 3 in a row as fast as possible. This way, the agent will learn how to win the game and not lose the game. Then putting the knowledge of winning will help the other agent to tried to take better actions as more game approaches. To do this we implement Q-learning has to reward and punish system for each step the agents takes if they win the game or lose the game respectively.

The second approach is to set the learning steps base on how many tied games the two agent have. This way, we will let both agent to play the game optimally from both side, unlike the first approach where one agent play with maximum potential while the other agent always loses. The learning style of the two approach will affect how agent behave if the order is used differently.

Eventually we want to make a connection between current stock market with the game tic-tac-toe. We want to reveal of how stock market is also unpredictable just like the moves by your opponents, and at the end you either win or losing the game just you either gain the profits or lose profits from stock market.

2 Reinforcement Learning

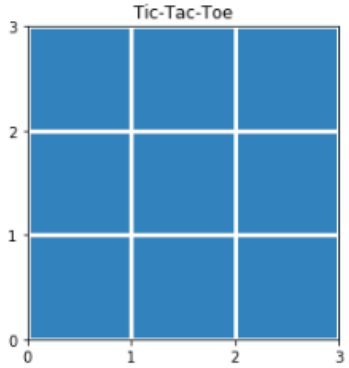
In this challenge we implement Deep-Q-Networks(DQN) to let agent observe and learn after taking each step of actions. We want to maximize the reward by giving the agent reward for each step closer to the win condition or tie condition whereas the reward function is defined as Rt = rt + γrt+1 + γ 2 rt+2…, where rt is the reward at time t and gamma as the discount factor. This function will get the reward for the steps and in order to get the optimal action value, we want take the max of the current state and action as a pair Q(s,a), s being the state and a being the action. As the agent take more steps on their way, they will have a more accurate choice of actions given from the Q-learning function. This allow us to let the agent use their previous experience to build up dataset of games that been played already to train by sampling them into batches and predict the next outcome. The complete algorithm for Deep-Q-Network is shown below: [1]

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Each agent will have their own DQN that works based on their experience. Let say we take approach one first and let agent 1 to win all the time getting all types of win situations. Then agent 2 will only lose and have a huge punishment as reward. Which then allows us to use approach two where we let tie as our largest reward allow and punishing them for. This will teach agent 2 how to win the game and at the same time rewarding if agent 2 can make a tie. Although they have their independent network, the both agent will ultimately learn to the game by losing and winning, making the game at a certain point to be all ties.

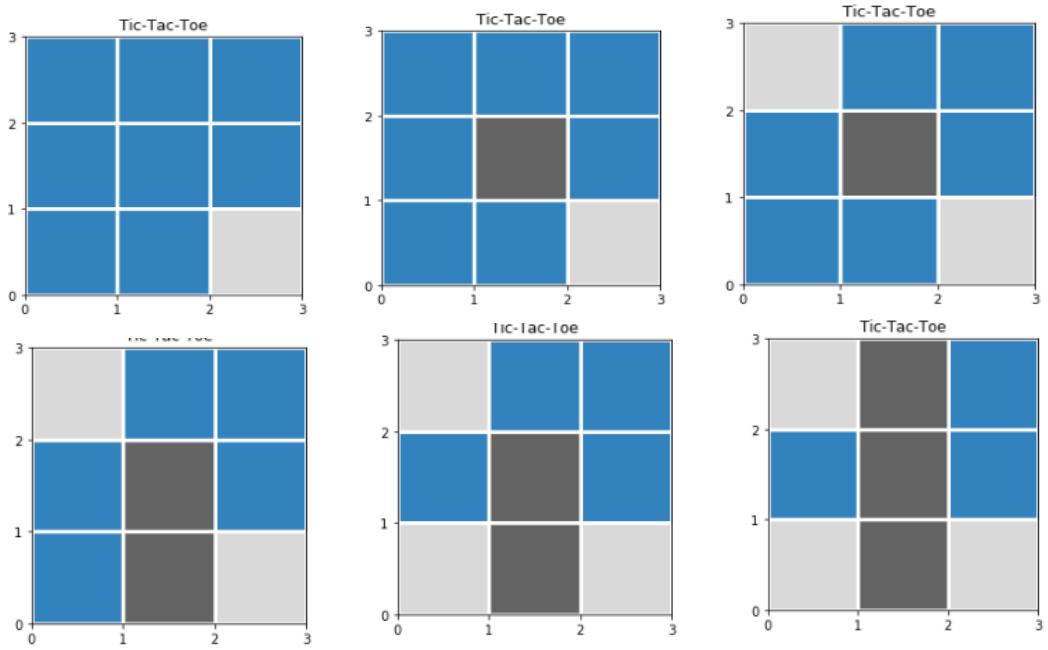
3 Setup of the Game

Setting up the environment, we use only 1 environment for both agents to observe and take action on. They both shared the environment and making their action upon on their individual reward system. Our board will be like a 3x3 grid representing the places the agent can act on.



The environment is really simple as you can see in the left. At the start, this represent no moves where each block is available for agents to act on. Each block start at the most top-left represent an action from [1 to 9]. Each step when an agent takes, it will update the board and change the block color to its individual color. For example, agent 1 takes an action in block 3, the board will then update the color for agent 1. We keep track each turn by mod the time or steps by 2. When the mod outcome is positive then it will be agent 1 turn if it is odd then it will be agent 2 turn. Let say after agent 1 act on block 3 and agent 2 act on block 4, then each turn the states changes.

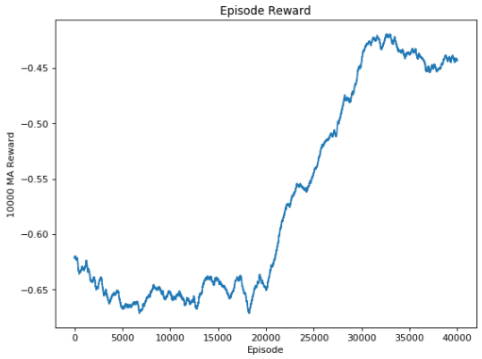
Ultimately the board game will go on until one of the agent wins or the board is filled up and no one wins which then it will lead to a tie.

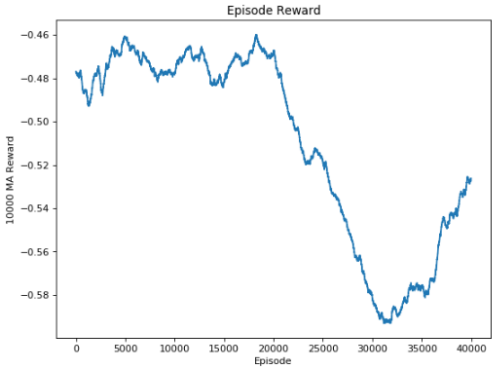


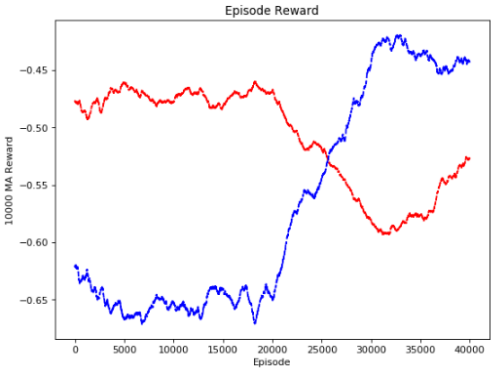
The figure above is an example of 1 episode. Here agent 2 won in the end since agent 2 got the 3 in a row first before agent 1. Agent 1 represent grey and agent 2 represent black. Both agent work on the same environment until the game is over. They communicate as the game plays on and using their own strategy and knowledge to play the game.

4 How it works

We created out neural network input as our states condition. In the beginning, the states are all initiated as 0. Either agent 1 moves or agent 2 moves, the value will be change to 1 for agent 1 and -1 for agent 2. As our dimension for our network, we will have 9 inputs as our possible states including the current state. 1 hidden layer that representing the possible next possible states. The dimension of the hidden layer will be 9x9 being the next possible states and connected to the output which represent the actions that are closest to winning condition train based on rewards. Since we are applying Q-learning, therefore there is highest reward that is provided by the environment, one of our agents can perform that action which would result in this amount of reward.





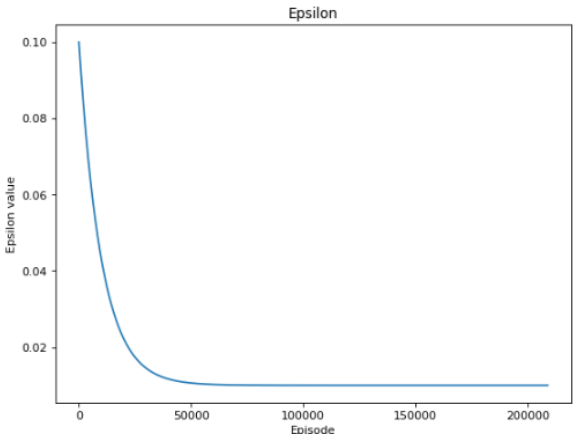


The two graphs above show the rewards for both agents for 50000 episodes. One agent is winning while another agent is losing. Based on the graph on the left, it combines both reward graphs together. As we could see one agent keeps winning at the beginning, and the other agent keeps losing. However, once both of them have learn how to play the game, we could see the lines start to change. Our prediction for episodes after 50000 is the two reward lines remind constant, and that is our goal of training the agents.

One problem for Q-learning is that at initial agents have no ideas which action to pick, therefore we introduce exponential-decay. The exponential-decay formula for epsilon can be shown as:

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda|S|}$$

The goal of introducing epsilon is to randomly select actions at the beginning based on 'exploration rate'.



The figure shown on the left is the epsilon for our game. As you can see the value initially is high as our agents perform random actions. As the agents been trained for more episodes, they would stop taking random actions, and start to take the action that result in highest rewards.

5 Summary/Future Work

Our goal is to make the game tie for both agents. But how can we relate it to real stock market scenario? In stock market, we want to maximize the profits. However, there always risk exist in the market. Comparing to we want to revel the idea of minimize the risk by exploring the stock market, and like we want our agents to explore the board as well. The idea we come up with is to train multi-agents to predict future stock prices. First agent could possibly train based on historical price of that stock, second agent may train based on news, third agent may train based on the trend of stock market itself. At the end we could possible perform a major voting, or taking the average of our agents result to be the predict of the stock price.

Reference

[1] Mnih, V., K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves et al. "Human-level control through deep reinforcement learning. Nature." (2015).