CONNECT FOUR GAME WITH Q-LEARNING AND DEEP Q-LEARINING

Saeed Tajik Hesarkuchak
dept. Electrical and Computer Engineering
Virginia Tech

***Abstract*— This paper explores the use of reinforcement learning algorithms, specifically Q-learning and deep Q-learning, to build an AI agent for the Connect Four board game. The primary challenge in training such an agent is the enormous state space, which makes it impossible to explore all possible game states in a reasonable amount of time. Therefore, we investigate the use of Q-learning and deep Q-learning to teach the agent to generalize from observed states and make informed decisions based on those generalizations. Additionally, we discuss the trade-off between exploration and exploitation, and present an epsilon-greedy exploration strategy to balance these competing goals. The experimental results show that the deep Q-learning agent outperforms the Q-learning agent, demonstrating the effectiveness of deep neural networks in handling high-dimensional state spaces. Overall, this work contributes to the growing body of research on reinforcement learning and its applications to game playing.**

I. Introduction

Due to developments in AI and machine learning, playing board games against AI opponents has grown in popularity in recent years. A more realistic and interesting experience can be had by using AI-powered game engines, which can present players with powerful opponents that can adjust to various play styles. AI agents may also go through a massive amount of game data to find patterns and methods that humans might not have noticed. Both humans and AI agents may benefit from better gaming as a result of this. Nowadays it's possible to explore strategy and tactics in a competitive setting thanks to the emergence of AI-powered board game opponents, and it will be interesting to see how this trend develops in the future.

One of the most popular two-player strategy game for decades is Connect Four [1]. The goal of the game, which is played on a 6x7 board, is to line up four of your own colored discs in a row in any direction before your opponent does the same. The game is a favorite of both competitive and casual players since it is easy to learn but offers a great lot of depth in terms of strategy.

A Connect Four learning AI bot can be developed using reinforcement learning (RL). A combination of supervised and unsupervised learning can be used to train the agent to predict game states depending on the state of the board and the player's move. The agent can also gain knowledge by self-play, in which it competes with a replica of itself to develop a better strategy over time. To increase its chances of winning the game, the agent can use RL to improve its decision-making process and learn from experience.

Fig. 1: Connect Four board game, Image from Hasbro Gaming Store

An AI agent can be taught to play Connect Four using reinforcement learning (RL), however due to the numerous different game states, the agent must learn to generalize from observed states. Deep Q-learning, which uses neural networks to estimate the value of each potential action in a given state, can be used to accomplish this. Another crucial factor to take into account when training the agent is balancing exploration and exploitation, which can be done by employing an epsilon-greedy exploration approach. By contrasting human choices with those of a skilled AI agent, RL can be used to assess and enhance human games and reveal top-level strategies and heuristics. [2]

A Kaggle competition [3] was recently organized to find the greatest Connect Four AI bot. The Connect Four competition on the Kaggle platform asked competitors to develop an AI bot that could outperform the greatest human players. Participants in the tournament came from all around the world, and the winning submission employed a combination of Monte Carlo tree search and deep reinforcement learning to achieve an impressive level of play. The contest demonstrated the potential of reinforcement learning to transform the way we play strategic games and the strength of AI.

In this paper, we discuss the notion of Reinforcement Learning, Q-learning, and deep Q-learning and try to imple-

ment each on these methods with a connect four board game. The goal is to build a connect four game AI using these Q-learning and deep Q-learning and compare it's results against another AI player which plays randomly.

II. Background

A. Reinforcement Learning

Reinforcement learning (RL) [2] is a type of machine learning that teaches an agent how to operate in a given environment in a way that will maximize a reward signal. The agent picks up new skills by trial and error, experimenting with various behaviors and analyzing the results, then modifying its behavior over time to maximize reward [4] [5].

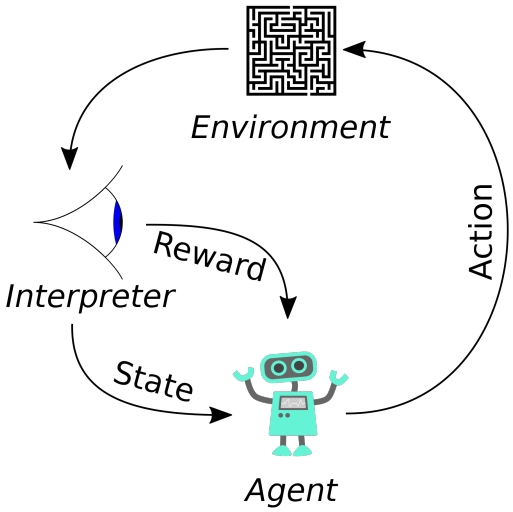

Fig. 2: The typical framing of a Reinforcement Learning (RL) scenario

As you can see in 2 an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent. The environment is typically formulated as a Markov decision process (MDP)

1) Markov decision process: A mathematical framework called a Markov decision process (MDP) [6] is used in reinforcement learning to describe decision-making issues. A set of potential states, a set of potential actions, and the rewards connected to executing those actions in each state are all described by MDPs. The main principle of an MDP is that state transition probabilities and rewards are entirely set by the state and action that is being taken at any one time, rather than by the history of prior states and acts. This is known as the Markov property [7].

A Markov decision process is a 4-tuple (S, A, P, R) [6] [7], where as shown the 3, S is a set of agent states, A is a set of actions the agent can take. $P = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability of transition from state s to state s' under action $a \cdot P : A \times S \to \mathbb{R}^S$. Also, $R(s, s', a)$ is the immediate reward after transition from s to s' with action $a \cdot R : S \times S \times A \to \mathbb{R}$

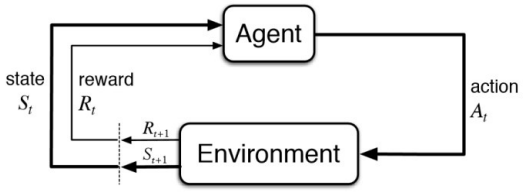

Fig. 3: A Markov decision process scenario

The goal of Markov decision processes is to find a policy for the decision maker by maximizing the sum of the rewards over all time steps [6].

$$\pi^* = \arg\max_{\pi} \sum_{t=1}^{T} R(s_t, s^*, \pi(a^*)) \tag{1a}$$

$$s^*, a^* = \arg\max_{s',a} P(s'|s_t, a) \tag{1b}$$

In this article, we discuss two approaches to create a reinforcement learning agent to play and win the game.

B. Q-Learning

The fundamental concept of Q-Learning is to map out the full observation space and then record the agent's behaviors inside that map [8]. The agent will then incrementally adjust its prior action based on whether it received a good or negative reward each time it comes across the same observation. A Q-Table is a type of data structure that stores the prior actions for each observation in the observation space [5]. The Q-Learning equation shown below is most frequently used to perform this incremental updating of the Q-Table:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_{t+1} + \gamma max Q_t(s_{t_1}, a) - Q_t(s_t, a_t)) \tag{2}$$

where r_t is the reward received when moving from the state s_t to the state s_{t+1} , and α is the learning rate $0 < \alpha < 1$. The discount factor γ determines the importance of future rewards.

The trade-off between exploration and exploitation is one difficulty that reinforcement learning faces that other types of learning do not. A reinforcement learning agent must favor activities that it has previously done and proven to be effective in creating reward if it wants to reap a large amount of reward. However, it must try actions that it has never chosen before in order to find such actions. The agent must take advantage of its past experiences in order to profit, but it must also investigate in order to choose better future courses of action. [2] To explore the state space of action a_t in the learning process we define a Epsilon-greedy policy such that:

$$\epsilon < rnd \qquad , a^* = \pi(s) = randomQ(s, a) \tag{3a}$$

$$\epsilon \geq rnd \qquad , a^* = \pi(s) = MaxQ(s, a) \tag{3b}$$

The most important concept is to teach two players at once. Another important distinction is that the award is contingent upon the success or failure of the opposing player. Player One's action should be rewarded if it resulted in the elimination of Player Two. To do so we follow the procedure in Algorithm1:

Data: A: Action state set
Data: S: Agent state set
Data: T : the end of the Game
Data: Hyper parameters : α, γ, ϵ
Data: α learning rate
Data: *rnd* a random number generator
Data: γ discount factor
Data: ϵ probability of exploration
Result: Q^1 : Qtable of player 1
Result: Q^2 : Qtable of player 2
Initialize $Q^1(s,a) = 0$, for all $s \in S$, $a \in A$
Initialize $Q^2(s,a) = 0$, for all $s \in S$, $a \in A$
Initialize s_t
Choose randomly who will start to play (player 1 or player 2)
while $t <$ T **do**
  Player 1 plays :
    Choose a_{t+1} thanks to the ϵ-greedy policy $(a_{t+1} = \pi(s_t, \epsilon, rnd))$
    Take action a_{t+1} and observe the reward r_{t+1} and new sate s_{t+1}
    Update $Q^1(s_t, a_{t+1})$ depending on α, γ, r_{t+1} and s_{t+1}
  if $t + 1 == T$ **then**
    Stop the while loop
  end
  Player 2 plays :
    Choose a_{t+2} thanks to the ϵ-greedy policy $(a_{t+2} = \pi(s_{t+1}, \epsilon, rnd))$
    Take action a_{t+2} and observe the reward r_{t+2} and new sate s_{t+2}
    Update $Q^2(s_{t+1}, a_{t+2})$ depending on α, γ, r_{t+2} and s_{t+2}
  $s_t \leftarrow s_{t+2}$
end

C. Deep Q-learning

Given that it has no restrictions on the observations it may make or the actions it can take in complicated situations, Deep Q-Learning [4] might be one of the most significant algorithms in all of Reinforcement Learning. With the aid of a system of observations, actions, and rewards, an agent can repeatedly "play" an environment using this reinforcement learning technique, which also integrates deep neural networks. [5]

deep Q-learning includes teaching a deep neural network to approximative the best Q-value function for a certain job. The predicted cumulative reward that an agent can earn by taking a specific action in a specific state and then implementing the best course of action is represented by the Q-value function. In deep Q-learning, the neural network receives the environment's current state as input and produces an estimated Q-value for each potential action. After selecting the action with the highest Q-value, which is commonly done by employing an epsilon-greedy exploration approach, the agent takes that action. [5] [9]

A variation of the Q-learning algorithm is used to train the neural network, and it entails reducing the mean squared error between the anticipated Q-value and the actual Q-value determined by the Bellman equation. During training, the Q-value estimations are updated using the Bellman equation, which depicts the recursive relationship between the current Q-value and the anticipated future Q-value. The replay buffer, which saves a portion of the agent's experience and randomly samples from it to decorrelate the training data and stabilize learning, is often used to train neural networks. [5] [9]

There is an inevitable trade-off between exploration and exploitation in all Reinforcement Learning issues. Epsilon decay is a phenomenon that is often used to address this issue. The probability that an agent will select a random action as opposed to one that has been predetermined by the network is represented by the epsilon in the epsilon decay equation. The epsilon decay is exactly that: since we expect the agent to be, for lack of a better word, stupid early in the training cycle, we will let it make essentially random decisions, allowing it to explore rather than exploit since it hasn't yet learned anything to exploit. The epsilon will shrink with time, meaning that the agent will begin to rely more and more on its acquired knowledge and less and less on random behaviors. [5] [9]

III. RESULTS

In this section we implement the two RL methods to build a AI Connect four agent and in the next section we compare results of our agents from these two different methods.

A. Q-leaning

We defined the board as a 6×7 (6 rows and 7 columns) and 42 cells. Therefore an action can take a value between 1 and 7. And the state set has 42 values. So both Q-table will be a matrix of 42×7. The agent is can be expressed as its location in the board. With the board defined we follow the procedure in Algorithm1 and calculated the Q-table after 10000 number of plays and $\epsilon = 0.3$.

As a results, we have got 59.97% winning when the Q-learning agent has played against a random player. As you see this number a bit close to our randomness percentage which is 30%. Since connect four is a solved game. this clearly shows that Q-learning hasn't done a very good job. (*Gif files of the games played has been attached.)

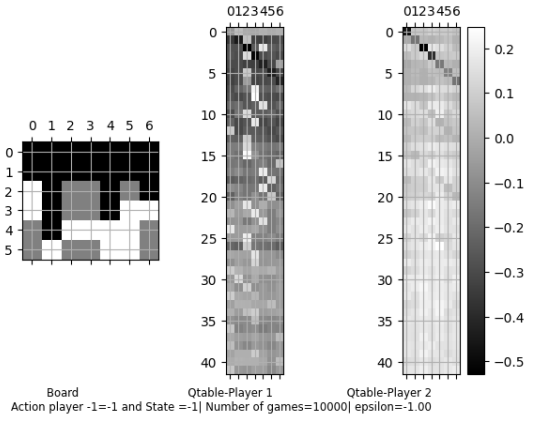

Fig. 4: $\mathrm{Q}_t ables and the board after 10000 games played$

B. Deep Q-leaning

We will develop our agent using (with a few modifications) the Connect X framework from a current Kaggle competition. Without having to create the Connect 4 game from scratch, this will make it very easy for us to get observations from the environment and deliver commands to

the environment. The only minor modification to the Kaggle-provided environment was the manual determination of the game's end time. This made it possible to determine rewards for various victories, losses, and incorrect movements.

First, the model is just a dense neural network constructed using the keras module. Due to the small size of the board, our initial attempt at using a convolutional neural network did not turn out as well as we had hoped. Seven neurons make up the network's output, each of which represents the action of dropping a chip into one of the board's seven separate columns.

Each observation/activity pair is assigned a loss value that is inversely proportional to the reward obtained by the loss function, which essentially takes the total rewards for an episode and applies them to each input and output of the network (observation and action, respectively). For instance, if the agent wins a game, the greatest reward available under our reward structure of 20 will be awarded to all observation/action pairings throughout the game. The inverse relationship is caused by the modest loss value that will follow. Since the objective of any standard optimizer is to reduce the network's loss, this inverse relationship enables our optimizer to direct network optimization toward positive rewards. Theoretically (and considerably less frequently in practice), decreasing loss will then boost reward, increasing the network's capacity to make Connect 4 game-winning judgments.

For trade-off between exploration and exploitation. This agent will employ the epsilon decay function, where x is the quantity of episodes learnt and $\epsilon = .99985x$.

Now that the entire structural system has been constructed We begin by creating the memory object that will be used to transmit data from the environment to the network. Following that, we repeat the process for each episode—in this case, 1,000—properly communicating observations and rewards to the network in accordance with the requirements of the aforementioned functions.

The following is a game played by our agent (blue) against the random agent (grey).

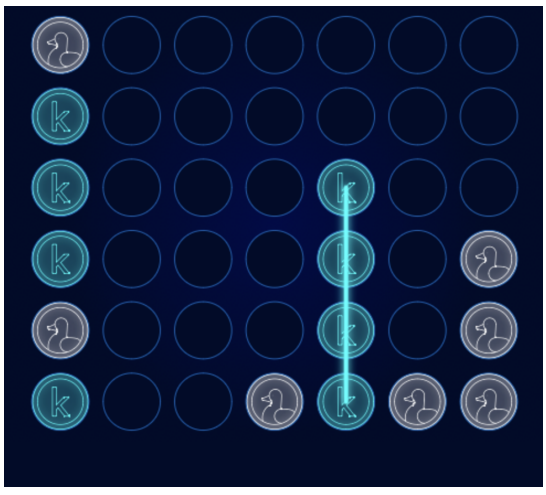

Fig. 5: A game played by the trained agent

As a results, we have got 85.00% winning when the trained agent has played against a random player. Which is much better than the winning percentage that we got using Q-learning.

1) Play against the agent: Although the agent's actions are undoubtedly impressive, But it could not compete with human-level intelligence. However, you can use the "play with agent" section in the code to play with our trained agent and test it.

IV. Conclusion

In Q-learning approach, The agent is only defined by its location in the board. This definition is quite blind. It means that the agent state does not reflect the entire state of the game. In this modelling, the agent state is very a narrow and a very local view of the game (the last move). This is a limitation of this approach.

For a game with few variations, such as tic-tac-toe, where maintaining a table to compress all the expected rewards for every potential state-action combination would only require a few hundred rows, the Q-learning strategy may seem feasible. The strategy of maintaining a single table to hold all of this information becomes impractical as games become more complicated since there are millions of state-action combinations to keep track of. According to the online Encyclopedia of Integer Sequences, there are 4 quadrillion 4,531,985,219,092 scenarios that need to be stored in a Q-table for Connect4. A table of this size would not fit on the hard drives of the majority of modern computers.

For Deep Q-learning, I'm amazed with the agent's success because it's clear that it has found out how to block victories, win for itself, and only play in columns that aren't already full. However, using Deep Q-Learning to address this situation is probably not the best course of action. The ideal approach, in my opinion, would be to use a deep learning assisted Monte Carlo Tree Search method if I were actually competing in the Kaggle competition with this agent. Also, we should consider that we only trained the model for 1000 iterations. Increasing this number would increase the performance of our agent.

References

[1] wikipedia. (1974) Connect four game. [Online]. Available: https://en.wikipedia.org/wiki/Connect$_Four$

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[3] Kaggle. (2020) Kaggle connect x. [Online]. Available: https://www.kaggle.com/c/connectx

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[6] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[7] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957. [Online]. Available: http://www.jstor.org/stable/24900506

[8] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.