CS7641 – Machine Learning

Assignment 4 – Markov Decision Processes and Reinforcement Learning

Mason J. Kelchner  
mason.kelchner@gatech.edu

# 

# INTRODUCTION

This analysis explores the application of reinforcement learning techniques to solve Markov Decision Process (MDP) type machine learning problems. An MDP is a process described by states, actions, and rewards. For example, consider the game Connect 4 except in this scenario when playing there is a small probability that when you attempt to play in a column, that you instead accidentally place your coin in one of the adjacent columns (perhaps this has happened to you). The game follows an MDP in that after every action taken by each player, the Connect 4 board changes i.e. the state of the game is completely defined as the board state and which players’ turn it is. Each player at any point in the game (at any state) has a certain choice of actions to take. Sometimes the columns or rows fill up in which case some actions are not available to a player given a state. Likewise, the game ends when a player is able to align 4 or more coins in a column, row, or diagonal. This outcome from an agent’s perspective would achieve a reward or +1 for winning and -1 for losing. According to the Markov property, an MDP’s future evolution is independent of its history and thus we can work to optimally solve MDPs using a variety of algorithms. While the Connect 4 example is easily explained, its implementation and solution as a reinforcement learning problem is difficult and exponentially large as the game contains a total of 4,531,985,219,092 possible board states [1]. For this analysis, two MDP problems are described and solved using three reinforcement learning methods. The first is the game of blackjack which contains far less states than Connect 4 with only 250 states [2]. The second is a simple grid-world style game where an agent stochastically explores a frozen lake with the goal of reaching a treasure but potentially falling through the ice in certain states. For this problem the game was a 24x24 grid world with 576 states forming a larger state space to explore. Both of these games were implemented using the open source repository in Python called *gym* and solved using the reinforcement algorithm learning module *bettermdptools* implementations of the three algorithms [3] [4]. The three algorithms used to solve these problems are value iteration, policy iteration, and Q-learning. For both games, a random seed was used to initialize both games for reproducibility for training. Only with Blackjack, when the optimal policy from each algorithm is tested were initializations of the game allowed to vary to evaluate general performance. For the Frozen Lake game, the policy was specific to the initialized game environment, thus it would be inappropriate to reinitialize a different random game state.

# Game Descriptions

Blackjack is a popular game of cards where a dealer and player aim to build a hand as close as possible to a value of 21 without going over 21 with the winner having the highest value. Going over 21 immediately ‘busts’ and the player or dealer loses, respectively. The numbered cards having their face value, face cards having a value of 10 and an Ace able to be either 1 or 11. This implementation of the game only considered two actions for an agent: HIT or STAND. A more sophisticated implementation would consider additional actions available in many styles of Blackjack including DOUBLING DOWN, SPLITTING, and SURRENDERING. Each of these actions have their own outcomes and potentially change the rewards from playing since new states would be available from a particular starting state. Additionally, playing an INSURANCE bet which is a side bet that the dealer has blackjack is another more sophisticated betting action. Allowing these additional actions would affect the reward structure of a reinforcement learning agent in the game. However, for simplicity, only the actions STAND (take no more cards) and HIT (take one more card) are allowed in this analysis. This is an interesting MDP in that the states encountered by the agent are stochastic and a finite number of outcomes are available from a starting state. By extension, many reinforcement learning problems are formulated as games and Blackjack can be used to understand outcomes of choices i.e. actions given a state as it relates to the MDP.

The second game for this analysis was the popular Frozen Lake environment available in the open source repository *gymnasium*. “Frozen lake involves crossing a frozen lake from Start (S) to Goal (G) without falling into any Holes (H) by walking over the Frozen (F) Lake. The agent may not always move in the intended direction due to the slippery nature of the frozen lake” [4]. The action space available to an agent is to move left, down, right, or up; however, as previously stated, this implementation includes a probability that the resulting movement of a commanded action moves perpendicular to the intended direction. This probability is a fixed 1/3 for all moves and directions. For example, commanding an action of right has the probability of moving up or down with a probability of 1/3 for each and 1/3 for moving in the intended direction. This randomness drives significant variation in a global policy for a fixed frozen lake configuration compared with a game where this randomness is not included and the actions are completely deterministic. Since actions can have more than one state outcome from the randomness, a learning agent explores additional states. Like Blackjack, this game is interesting and relevant to reinforcement learning as larger grid worlds of the frozen lake game contain far more possible states than blackjack and its states resulting from actions are not deterministic. However, all states are accessible from the starting state in the Frozen Lake game, granted with small probability, while in Blackjack only a certain subset of states are accessible given a particular initial state. For example, an agent in Frozen Lake can explore all states by making successive move throughout the board, even when the action results are random, there remains a non-zero probability of reaching all other board states. While in Blackjack, being dealt an initial hand of a 4 and a 5 (i.e. a hand total of 9), and not considering the ability to SPLIT (only when the starting two cards are the same value) an agent cannot reach a state where the hand total is less than the initial hand value of 9 during this game. Along with the size difference in states in Frozen Lake, this difference is valuable for understanding the effect of sparse rewards in a large environment on the performance of reinforcement learning techniques. For this analysis, a grid size of 20x20 was used for the Frozen Lake problem, resulting in 400 states. For comparison with later results, the initialize game state for Frozen Lake is shown in Figure 2‑1.

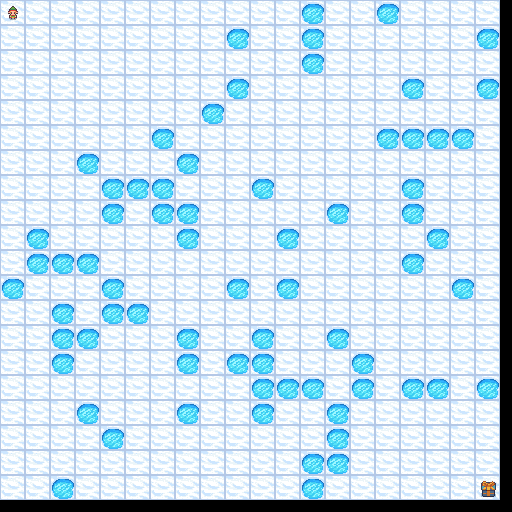


Figure 2‑1 Frozen Lake 20x20 initialize game state

# Policy and Value Iteration

Two algorithms applied to both MDPs in this analysis were policy iteration (PI) and value iteration (VI). Both involve iteratively updating a state-value-action mapping of an environment described by an MDP to find the optimal policy function (for PI) and optimal value function (VI). Given enough time and iterations, both policy iteration and value iteration will converge to the optimal solution for an MDP. Convergence for a PI and VI algorithm occurs when the value function for state transitions no longer increases during iterations. During each update step for PI and VI, the next policy and next value for a given state may not be known but it is guaranteed that the new policy and new value function is greater than or equal to the previous step’s policy and value function at every state. The optimal value fulfills the Bellman optimality equation. The PI algorithm stops once the policy improvement doesn't change π. That is the exact case when v\_π fulfills the Bellman optimality equation. Thus the policy π must be equal to an optimal policy, once policy iteration stops. In order to use PI and VI algorithms on an MDP problem, the transition and reward matrices must be known apriori. This is not always possible and deviates from some reinforcement learning problems where these matrices are not or cannot be known in advance and must be “discovered” through advancing through the MDP by making actions. Additionally, PI and VI algorithms are limited to discrete state space MDPs as they are tabular algorithms with finite bounds. Continuous state space MDPs can be discretized to use PI and VI; however, uncertainty is introduced as chunking a continuous space into discrete spaces removes intermediate states which may or may not have a significant impact on the optimal policy and value functions. Both algorithms were applied to the Frozen Lake and Blackjack MDP problems. For the Blackjack problem, with a smaller state size, shorter evolution loops and quicker reward feedback than the Frozen Lake problem, both PI and VI algorithms converged to an optimal policy/value function quickly, converging within 0.01 seconds of wall clock run time. Likewise, even with a large state space and stochastic actions in the Frozen Lake MDP, both PI and VI converged within a few seconds of wall clock runtime. Value iteration converged in approximately 1,000 iterations while policy iteration converged within 10 iterations; however, both algorithms took similar amounts of runtime to converge.

# Q-Learning

Q-Learning is a model free reinforcement learning class of algorithms which doesn’t require the state and reward matrices apriori and instead builds a model through interacting with the environment, receiving rewards, and selecting new actions. Additionally, Q-Learning algorithms allow making “off-policy” actions allowing a reinforcement learning agent to explore the environment and move into states that may have not been previously explored with the goal of discovering potentially better actions in each state based on the discounted rewards received. The Q-learning process balances this off-policy exploration with the on-policy selected action of exploitation using a simple probability during a training episode which can be changed from episode to episode. Initial training episodes typically start with a high probability of selecting a random action in a given state, this is a high epsilon resulting in a high probability of exploration. As the agent receives rewards, it builds a Q matrix containing the Q values for taking a particular action, a, in a particular state, s. This matrix is updating according to the Bellman equation including the learning rate alpha and the discounted reward rate gamma. As the agent continuously re-explores the environment, the exploration parameter, epsilon, is reduced to increase the probability that the agent will select the on-policy best action in a state by selecting the action with the highest Q-value for that state-action pair. Q-learning is more flexible than PI and VI algorithms in that it is model free and doesn’t require knowing the state-value functions in advance which in some MDPs can be impossible to recover. However, Q-learning requires significant computation especially for large state space MDPs since the agent must explore the environment for many iterations to build a Q matrix which accurately represents the true state-action-state transitions. Additionally, Q-learning is highly sensitive to a particular MDP’s state space and reward function and requires significant parameter tuning to improve the performance of the learning algorithm and for the reward function to potentially altering the reward outcomes. Sparse rewards in a large state space MDP are particularly difficult for Q-learning in that many state-action-state transitions must occur during an iteration before a reward is obtain and propagated through the Q-table through discounting. This results in a slow learning process and increases the number of iterations necessary to converge to an optimal policy. Reward shaping can be implemented to produce intermediate rewards for actions not resulting in ending an iteration, this can improve the balance of exploration and exploitation and benefit the converge of the Q-learning algorithm. Reward shaping and parameter tuning were implemented for Q-learning for both of the MDPs in this analysis and particularly for the large state space Frozen Lake problem.

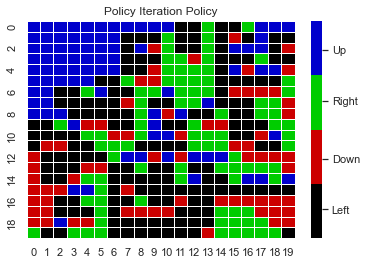
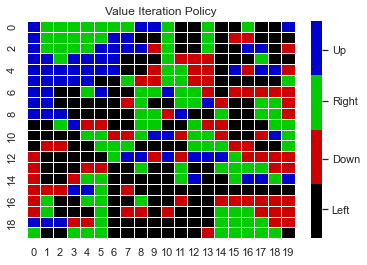
# Result Comparison

For the Blackjack game, each algorithm was tested by simulating 10,000 games. A parameter grid search was performed using the Q-learning algorithm with the best criteria being determined after training by evaluating the percent of wins and losses of the algorithm versus the theoretical optimal percentages in Blackjack without counting cards. For Blackjack, the optimal policy wins and loss percentages are approximately 43% and 48%, respectively, with the remainder of games resulting in ties. The grid search of hyper parameters for Q-Learning varied the learning and epsilon initial, minimum and decay rates. The performance of the final tuned Q-learning algorithm after simulating 10,000 games is shown in Table 3‑1.

Table 3‑1 Policy Performance for Blackjack

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Wins | Losses | Ties |
| Policy Iteration |  |  |  |
| Value Iteration |  |  |  |
| Q-Learning |  |  |  |

Note that the final policy determined by both PI and VI are



# Summary

# References

|  |  |
| --- | --- |
| [1] | E. a. Kissmann, "Number of legal 7 X 6 Connect-Four positions after n plies". |
| [2] | "Playing Blackjack with Machine Learning". |
| [3] | "Bettermdptools". |
| [4] | M. Towers, J. K. Terry and e. al, "Gymnasium: An API standard for single-agent reinforcement learning environments, with popular reference environments and related utilities (formerly Gym)," 2023. |
| [5] | F. Pedregosa and e. al, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011. |
| [6] | G. B. e. al, "OpenAI Gym," 2016. |