UDACITY MACHINE LEARNING NANODEGREE
2019

CAPSTONE PROJECT

Application of Reinforcement Learning Methods

In

Easy21 Game.

Abdul Mdee.

March, 2019

I. Definition

a. Project Overview

Reinforcement learning is a branch of machine learning whereby an agent learns by interacting with the environment, see figure 1. It is inspired by behaviorist psychology. Reinforcement learning is very different from other types of machine learning (supervised learning and unsupervised learning) because, in a reinforcement learning, the learning occurs through interactive experience while the other types of machine learning are through learning from the dataset.

Reinforce learning is very important because it is the closest science has come to build a machine with almost human intelligent. I am attracted to reinforcement learning because of the possibility of understanding how human intelligent works. Also using the reinforcement learning techniques to solve and improve classical problems such as teaching a robot to walk. The field of reinforcement learn shows so much promise in improving / building an actual artificial intelligent (AI). As we can see example from projects like Deepmind; where AlphaGo used deep reinforcement learning to defeat a Go world champion, and arguably the strongest Go player in history.

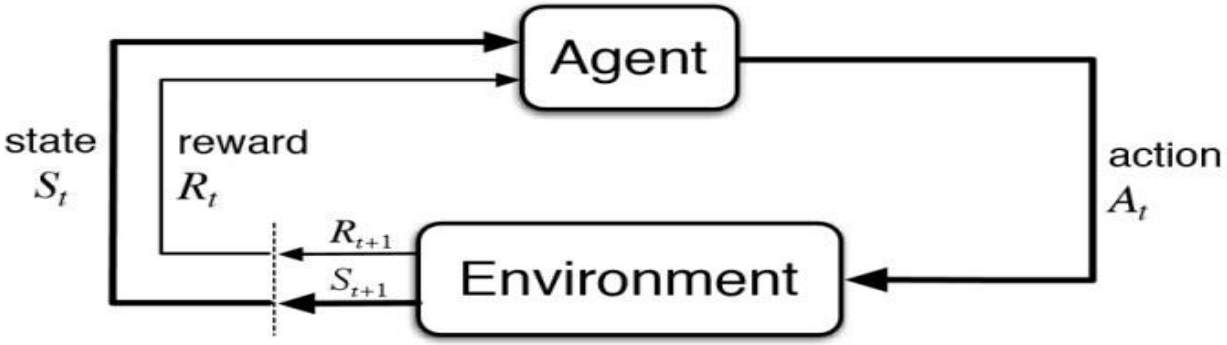

Figure 1

b. Problem Statement.

In order to actually understand the field of reinforcement learning and its applications, we have to understand how reinforcement learn works in a simplified problem such as a game of Esy21. I am going to apply reinforcement learning methods to a card game called Easy21. Easy21 is similar to the Blackjack but the rules are different and non-standard. The project problem statement is inspired by David Silver Class at UCL.

❖ *Please note that this report aims to address the questions provided in the assignment Easy21-Johannes offered in David Silver Class at UCL. This approach was selected as a way of solidifying my understanding of Reinforcement learning*.

c. Metrics

The strategy for approaching this problem will be as follow; First, build an environment named Easy21 which will include the step function. The step function will take an action 'a' as input and return the next state and a reward. Once, the environment has been established, the three reinforcement algorithms, Monte- Carlo, TD learning and linear function approximation will be implemented through the easy21 Environment.

The performance of the Monte-Carlo algorithm will be measured by comparing to the figure provide from Sutton and Barto's Blackjack example, this will be a good benchmark see Figure A. Please see the Easy21-Johannes pdf. Applying the Monte-Carlo algorithm to Easy21 environment should replicate the same figure Sutton and Barto's Blackjack example. The TD learning and linear function approximation performance will be measured by calculating the mean square error of the Q_value calculated from the

Monte-Carlo algorithm. The Monte-Carlo algorithm is guaranteed to converge to the optimal value function if given enough example.

- *A strategy for implementing and applying reinforcement learning methods to a card game called Easy21 is already predefined, as my project is completely mirrored to the assignment given in David Silver Class at UCL. Please see the Easy21-Johannes.pdf file for further clarification.*

II. Analysis.

The environment that implements the game Easy21 contain a step function, which takes action, 'a' as input as specified in the assignment. An action ‘a’ can be 1 for hit or 0 for stick and when passed to the step function it will return a sample of the next state, s (which may be terminal if the game is finished) and reward r. The environment for Easy21 is designed for a model-free reinforcement learning, and it does not explicitly represent the transition matrix for the MDP. There is no discounting ($\Upsilon = 1$). The dealer's moves as a part of the environment, i.e. calling step with a stick action will play out the dealer's cards and return the final reward and terminal state.

The environment was tested by passing a randomly generated action to the step function. The step function returned to the next state, a reward and Boolean indicating terminal state. The returned reward was used to track the wins and loss of the random actions taken for a different number of times the game played. Randomly playing the game for 100 times while randomly selecting an action, the player had a 41% chance of winning the game and 59% chance of losing the game. See figure 2. Playing the game for a 1000 times the player had a 44% chance of winning the game and 56% chance of losing

the game. See figure 3. Note, in both scenarios the player had an equal chance of selecting ‘Hit’ or ‘Stick’.

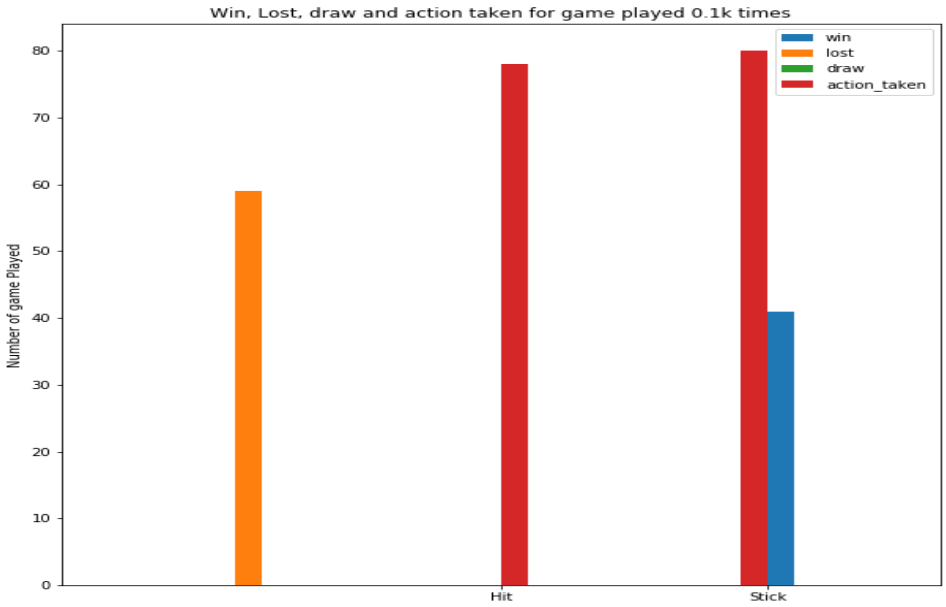

Figure 2.

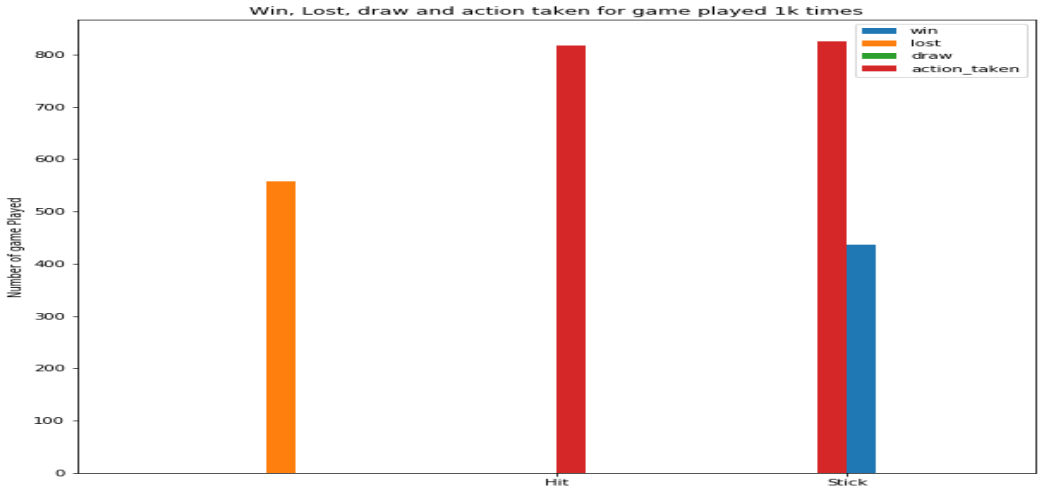

Figure 3.

- The *assignment* *Easy21-Johannes* clear specify the list of choice of algorithms to be implemented. The implemented algorithms were preselected as part of the *assignment* *Easy21-Johannes*. My intuition tells me that Monte-Carlo control was selected as it is guaranteed to converge and find the optimal value function if given enough example, TD Learning was selected as the algorithm is expected to find the optimal value function faster than Monte-Carlo because TD learning method updates the value function after every time step. Linear function approximation was selected as expectation being the fastest out of the two previous mention algorithm because it reduces the space and time complexity of an algorithm by selecting features which represent the states. States with similar feature values will be considered to be similar. The value function is represented as a weighted sum of the features.

III. Methodology.

Three different reinforcement algorithm will be implemented using the Easy21 environment described above. First, Monte-Carlo Control will be implemented in part a, followed by TD learning in part b then finally conclude with the implementation of Linea function approximation in part c. Detail of code implementation can be located in the report Jupiter notebook.

a. Monte-Carlo Control:

The assignment required to Apply Monte-Carlo control to Easy21. Initialize the value function to zero. Use a time-varying scalar step-size of $\alpha_t = 1/N(s_t,a_t)$ and an -greedy exploration strategy with ($\epsilon_t = N_0/(N_0 + N(s_t))$, where $N_0 = 100$ is a constant, $N(s)$ is the number of times that state s has been visited, and $N(s,a)$ is the number of times

that action *a* has been selected from state *s*. Feel free to choose an alternative value for N_0, if it helps producing better results. Plot the optimal value function; $V^*(s) = \max_a Q^*(s,a)$ using similar axes to the following figure taken from Sutton and Barto's Blackjack example.

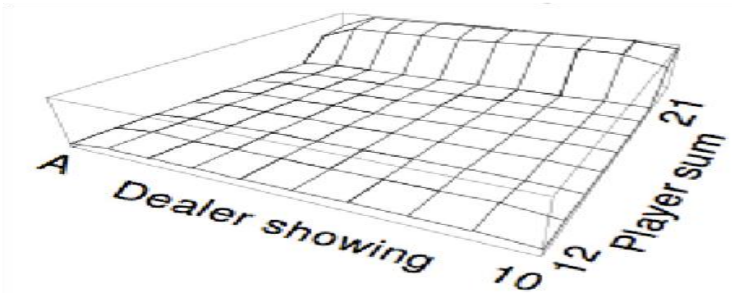

Figure A.

Monte-Carlo control was applied to Easy21. Firstly it was applied with N0 = 100 for about 50k iteration. The optimal value function was not smooth and it had a lot of noise. See figure 4. The value function became smoother as the N0 & number of iteration increased. See figure 4, 5 & 6. Figure 6 is the smoothest of the three figures at 1000k iteration and N0 = 10k. See the report notebook for code implementation.

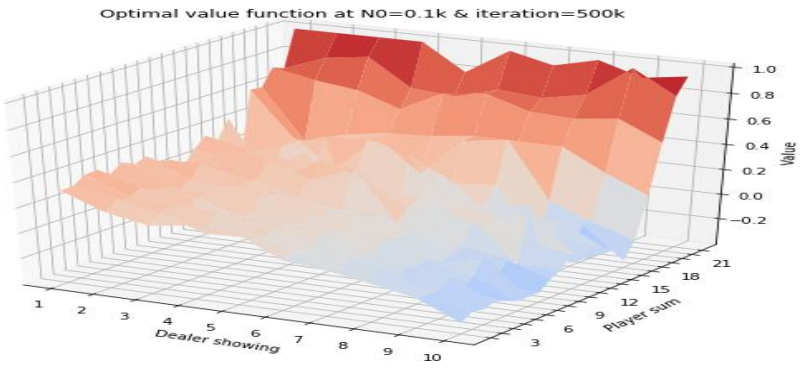

Figure 4

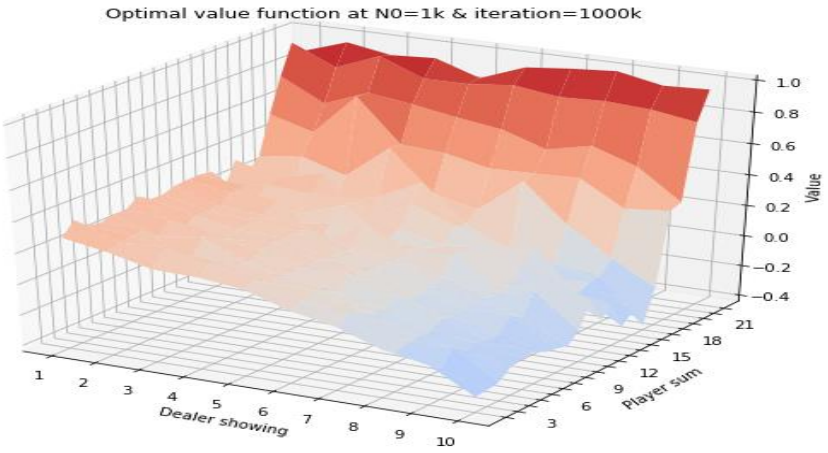

Figure 5

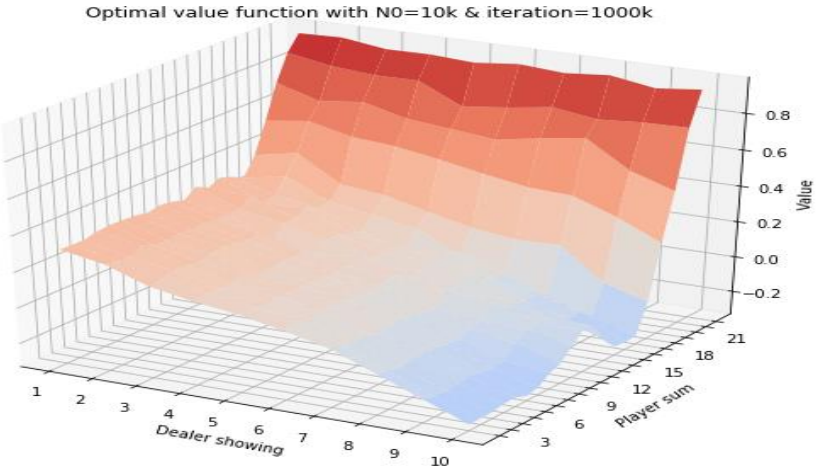

Figure 6.

From figure 6, the optimal value function is higher when the player sum is big, that means the player has a high chance of winning the game if the player implements the

policy corresponding to the optimal value function indicated in figure 6. See the report notebook for all the code implementation. Feel free to change the N_0 value for the more accurate optimal value function.

b. TD Learning in Easy21.

The assignment requires to implement Sarsa(λ) in 21s. Initialize the value function to zero. Use the same step-size and exploration schedules as in the previous section. Run the algorithm with parameter values $\lambda \in \{0,0.1,0.2,...,1\}$. Stop each run after 1000 episodes and report the mean-squared error $\sum_{s,a}(Q(s,a) - Q^*(s,a))^2$, over all states s and actions a, comparing the true values $Q^*(s,a)$ computed in the previous section with the estimated values $Q(s,a)$ computed by Sarsa. Plot the mean squared error against λ. For $\lambda = 0$ and $\lambda = 1$ only, plot the learning curve of mean-squared error against episode number.

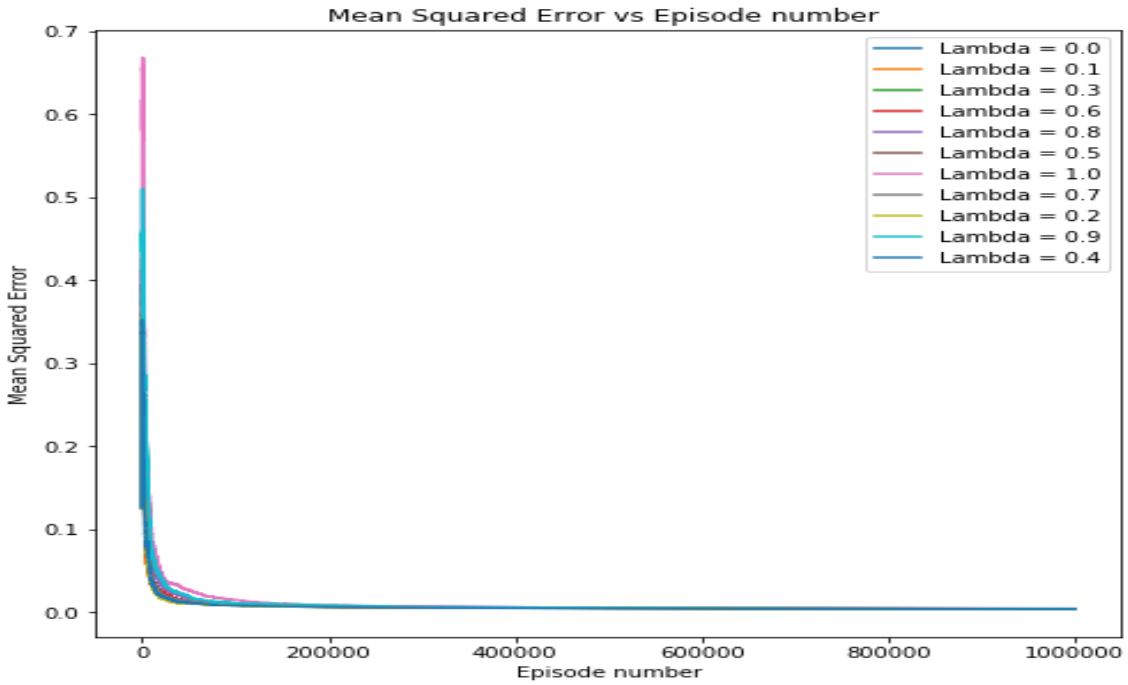

Figure 8

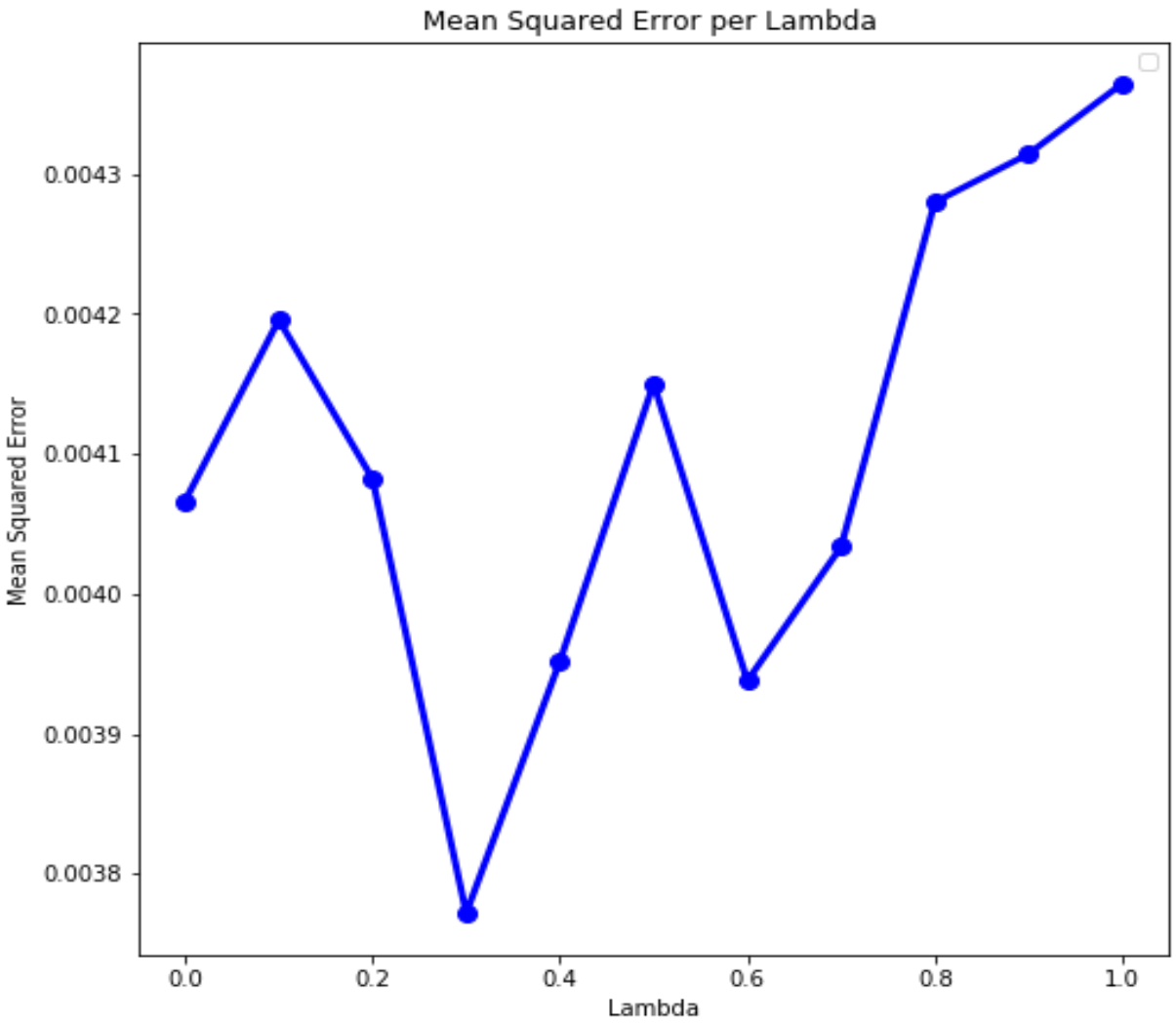

Figure 9.

- Fig 8, shows that after 200k iteration the mean square error does not decrease. The lambda with smallest error is 0.2 & 0.4 lambda. See figure 8.

c. Linear Function Approximation in Easy21

We now consider a simple value function approximator using coarse coding. Use a binary feature vector $\varphi(s,a)$ with $3 * 6 * 2 = 36$ features. Each binary feature has a value of 1 iff (s,a) lies within the cuboid of state-space corresponding to that feature,

and the action corresponding to that feature. The cuboids have the following overlapping intervals:

dealer(*s*) = {[1,4],[4,7],[7,10]}

player(*s*) = {[1,6],[4,9],[7,12],[10,15],[13,18],[16,21]} a = {*hit*,*stick*}

Where:

- *dealer*(*s*) is the value of the dealer's first card (1–10)
- *sum*(*s*) is the sum of the player's cards (1–21)

Repeat the Sarsa(λ) experiment from the previous section, but using linear value function approximation $Q(s,a) = \phi(s,a)^T\theta$. Use a constant exploration $\epsilon = 0.05$ and a constant step-size of 0.01. Plot the mean-squared error against λ. For $\lambda = 0$ and $\lambda = 1$ only, plot the learning curve of mean-squared error against episode number.

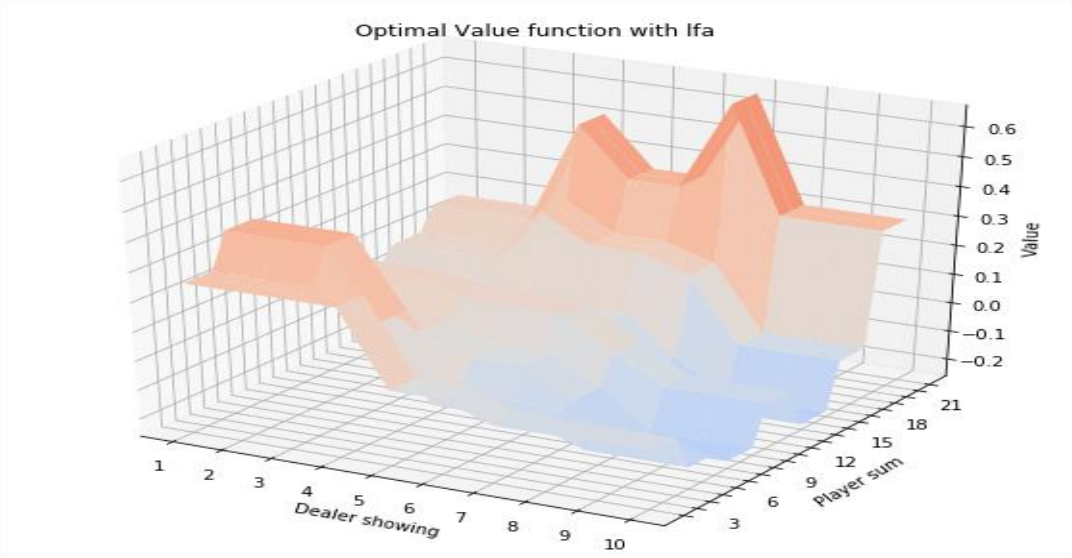

Figure 10

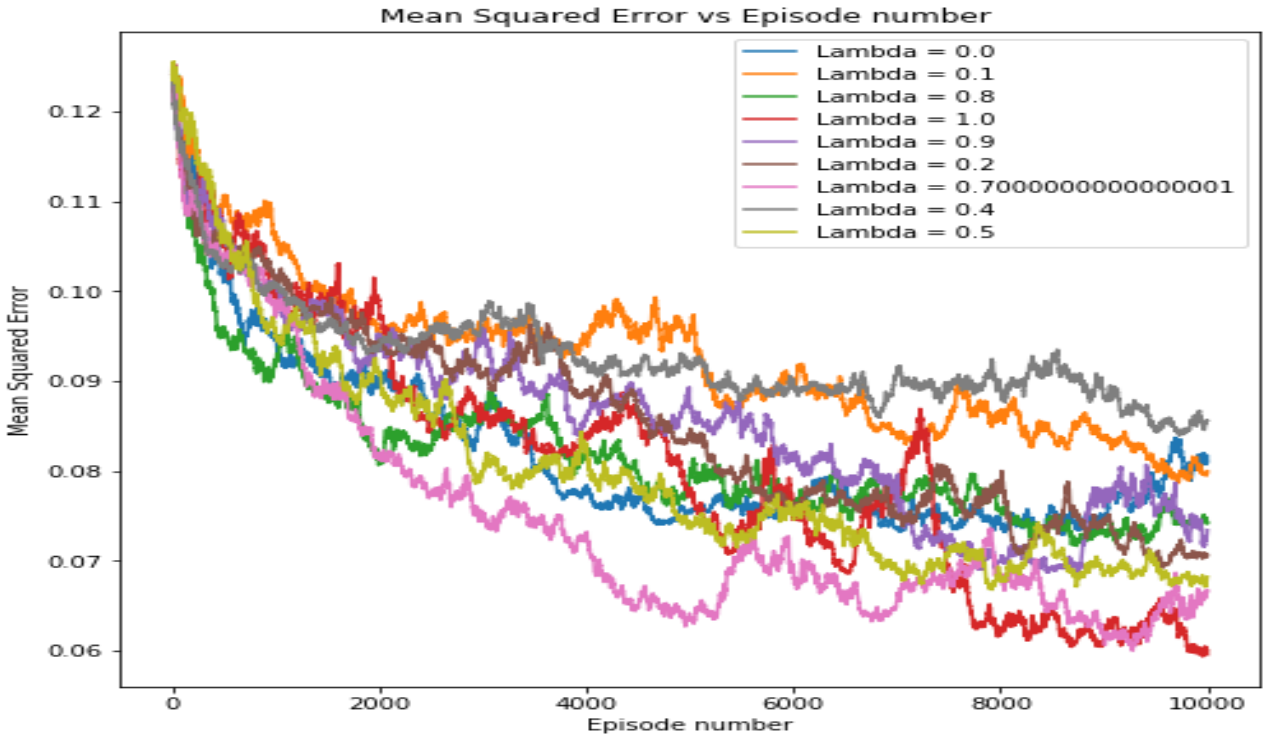

Figure 11

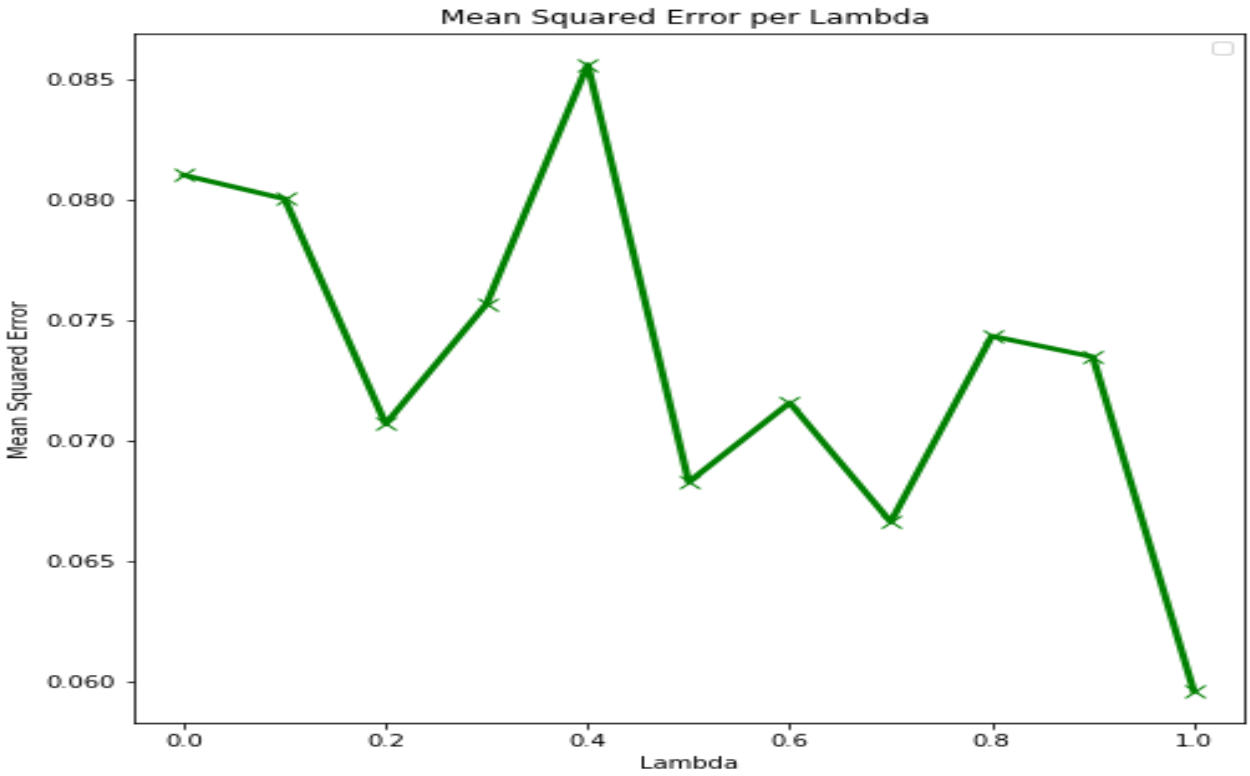

Figure 12

- Figure 10 shows a value function which is not as smooth as the value function seen in Figure 6. Also the rate at which the mean square error decreased as the number of episode increased is not the same as that observed in Figure 8. The highest mean square error was between lambda 0.2 & 0.4 as seen in figure 12 opposite to what was observed in figure 9

❖ The implementation of all three reinforcement algorithm, Monte-Carlo, TD learning and linear function approximation to Easy21 proceed as expected. The challenging process of implementing all the three algorithms was to code an environment which all the three algorithms can be implemented efficiently. Also, the coding of the linear function approximation needs to be improved because out of all three algorithms, linear function approximation was expected to be faster compared to the rest but that was not the case, linear function approximation had the slowest running time compeered to the other two algorithms implemented.

IV. Result.

a. Model Evaluation and Validation.

The pros and cons of using bootstrapping involving helping the agent to learn to take better actions more quickly as the algorithm update on the current rewards and intermediate function steps. It reduces the variance as observe in figure 8 & 9. The mean square error of the value function was optimal reduce after 200k iteration with lower values of lambdas having the lowest mean square error. Due to a bias-variance tradeoff, bootstrapping is prone to be biased as the cost of reducing variance. It

should be noted that it's feasible for a Monte Carlo agent to converge to an optimal value function if given enough examples. Bootstrapping will be more helpful in a task that has a longer running time such Easy21 game compared to blackjack as it makes update and adjustment prior to the end of an episode/task to update the value function.

The pros and cons of applying function approximation in Easy21 involve a reduction in space and time complexity of algorithm due to fewer variables to learn but with a cost of losing accuracy as the function approximation can be prone to bias. Also, it should be noted that the step-size is kept constant in the learning process, some regions of the value function receive much less training than other parts, which can attribute to the decreasing of algorithm accuracy. See figure 10 compare to figure 6. The value function in figure 10 is not as smooth as in figure 6. In the future, I suggest we explore the idea of using a neural network to approximate the linear function. Also, further investigation on how to efficiently (computational) implement the linear function approximation will help to get a better result.

b. Justification.

The figure provided in the assignment Figure A or see Easy21-Johannes.pdf section 2 under Monte-Carlo Control was used as benchmark of this project. Our algorithm performed very well as it was able to replicate Figure A. See Figure 6.

V. Conclusion

a. Free-Form Visualization.

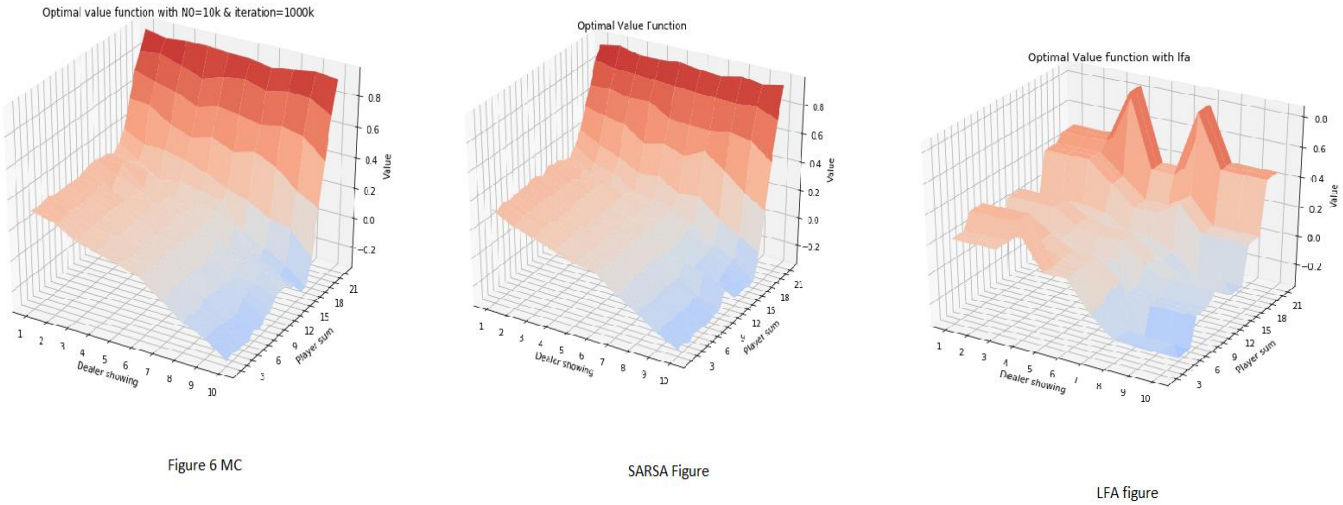

Figure 6 MC

SARSA Figure

LFA figure

Free-form Fig.

- The above figure shows that all out algorithm performed as expected. All the value function obtained from the different algorithm were not very far off from the benchmark case which was provided in the Easy21-Johannes.pdf assignment (figure taken from Sutton and Barto's Blackjack example) or figure A.

b. Reflection.

Monte-Carlo Control algorithm is guaranteed to converge to the optimal value function if given enough example. In our case, the agent play as many as 1000k games as shown in figure 6, the agent will find the optimal value function. The value function generated from 1000k is much smoother compared to the rest which was generated with less 1000k games see figure 4 & 5.

TD Learning algorithm found the optimal value function much faster compared to the Monte Carlo Control algorithm. The algorithm was able to reduce the mean square error after 200k games. The same observation was observed with small lambdas compared to large lambdas. See figure 8 & 9.

Linear function approximation. It was supposed to reduce the space and time complexity of algorithm due to fewer variables but when it was implemented it took much longer time to run compared to the previous two implemented algorithms. Further investigation on how to compute and efficiently implement the linear function approximation will help to get better results.

c. Improvement

For future reference, MDP can be implemented to compare the performance and result to that of reinforcement learning result. Also, with the advance in neural network, I think it will be beneficial if I can find a way to incorporate neural network architecture in this assignment. I intend to improve this project as acquire more knowledge about Reinforcement.

VI. Reference :

a. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

b. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/Easy21-Johannes.pdf

c. https://github.com/dennybritz/reinforcement-learning

d. http://www.incompleteideas.net/book/RLbook2018.pdf