Blackjack with Action Value Function Approximation

Author: ZHA Mengyue

SID: 20668901

Math Department of HKUST

More on Github Repository

Problem Statement

We study playing Blackjack by Action Value Function Approximation. Prediction methods used to update q-value function for option here are linear Monte Carlo, linear Q Learning and linear Temporal Difference. We also test the algorithm under different combination of (M, N). M is the number of decks and N denotes N-1 palyers with 1 dealer. For each configuration, we find the optimal policy after iterations. Outcomes of three pre diction methods are compared by visualization and tables.

This work is based on the previous work done for HW1. We will compare the outcomes with and without linear action value function approximation.

Since details in the game implementation are exactly the same with HW1. We omit them here and please refer to HW1 if you'd like to know more. Next we discuss the design of linear approximation.

x: is the feature of form (player_points, dealer_points, player_action)

w: a linear weight of shape (3,)

For a call on value by (s, a) pair where s = player_points, dealer_points, a = player_action

We have $Q(s,a) = w^T x$

The linear approximated Q value function Q is updated by the superviesed learning.

Homework Statement

Assume that in the Blackjack game, there are m decks of cards, and n players (a dealer vs $n-1$ players). The rules of the game are explained above.

(1) Find the optimal policy for the Blackjack, when $m = \infty$ and $n = 2$. You can use any of the methods learned so far in class (e.g. linear Monte Carlo, linear TD, or linear Q-Learning). If you use more than one method, do they reach the same optimal policy?

(2) Visualise the value functions and policy as done in Figures 5.1 and 5.2 in Sutton's book.

(3) Redo (1) for different combinations of (m,n), e.g. $m = 6, 3, 1$, and $n = 3, 4, 6$. What are differences?

Implementation

File Structure

Since the overal structure is the same with HW1. We will only talk about the new added functions.

We put most new functions in AVAF.py

- MC_linear(): update the Q value function by MC samples propagated by SGD
- QL_linear(): update the Q value function by QL samples propagated by SGD
- TD_liner(): update the Q value function by TD samples propagated by SGD
- Q_function(key, w): calculate the feature x of the given key and return the Q value of for the given key by linear approximation $w^T x$
- calculate_Q_sa(Q_sa, w): use the current linear weight w to restore the Q value table Q_sa for all keys in Q_sa.

Example

1. Prepare the environment

```
conda create -n Blackjack python=3.6
```

```
conda activate Blackjack
```

Now your working environment is the Blackjack now. Let's install the necessary packages. We have listed all packages in requirement.txt

```
pip install -r requirement.txt
```

Now your environment should be fully ready.

2. Experiments on a single Instance

   The following code blocks plays the Blackjack with m=2 decks and n=3 people where 2 are players and one is the dealer.

```
python main.py --m=2, --n=3
```

   Note that when $m = \infty$, we use --m=0 instead.

```
python main.py --m=0, --n=2
```

3. Experiments on instances of combinations of (m, n)

   Also you can test the combinations of (m, n) pairs. For example, m= 6, 3, 1 and n= 3, 4, 6

```
python main.py --m 6 3 1 --n 3 4 6
```

4. Experiments on $m = \infty$

   We use --m=0 infers to use infinite decks in the game instead.

5. The optimal policy

   We store the final Q-value function instead and the optimal poliy are derived from it by either best policy or epsilon greedy policy.

   The value.csv are stored in thecorresponding instance folder as:

   MC_best_value.csv

   MC_epsilon_value.csv

   QL_best_value.csv

   etc.

Tabular Summary for the Experiments

Choices for policy update: policy=['best', 'epsilon']

Choices for policy evaluateion(value function update): update=['MC', 'QL', 'TD']

- best: best policy evaluation
- epsilon: epsilon greedy policy evaluation
- MC_linear: Monte Carlo with liner AVAF
- QL_linear: Q Learning with liner AVAF
- TD_linear: Temporal Difference with liner AVAF

We see that the models with linear approximation perform worse than non-linear approximation. This is because we use very simple feature that underfit the real action value function.

Single Instance of $m = \infty, n = 2$

m=∞, n=2	MC_linear	QL_linear	TD_linear
best policy	18.0500%	24.2700%	24.8100%
epsilon greedy policy	20.1400%	28.6900%	27.2700%

Conclusions:

- epsilon greedy policy outperforms best policy
- With liner approximation, MC, QL, TD perform worse. This may because I did not do any feature engineering. The linear function is to simple and underfit the real Q value function.

Combination of m=[6, 3, 1], n=[3, 4, 6]

We summary the performance of (update_policy) combinations in the tables below.

MC_best	n=3	n=4	n=6
m=6	27.8000%	28.5600%	27.9080%
m=3	28.0200%	27.8667%	29.2500%
m=1	27.8500%	28.7833%	28.9440%

MC_epsilon	n=3	n=4	n=6
m=6	29.0600%	28.2433%	27.8420%
m=3	28.8650%	29.1967%	28.5940%
m=1	29.4400%	28.4367%	28.8680%

QL_best	n=3	n=4	n=6
m=6	35.900%	37.6067%	36.8160%
m=3	36.8250%	36.4067%	36.5960%
m=1	36.9750%	36.7433%	36.9180%

QL_epsilon	n=3	n=4	n=6
m=6	36.6950%	36.8100%	36.6420%
m=3	36.5900%	35.4767%	36.2400%
m=1	36.5000%	36.1300%	36.3960%

TD_best	n=3	n=4	n=6
m=6	36.1950%	35.9667%	36.4840%
m=3	35.6800%	36.1833%	37.4880%
m=1	36.0950%	37.3200%	36.9300%

TD_epsilon	n=3	n=4	n=6
m=6	37.1250%	36.6633%	37.3560%
m=3	36.5000%	37.5100%	37.7860%
m=1	36.8200%	36.9600%	36.9720%

Conclusions

- epsilon greedy policy is slightly better than the best policy
- With linear approximation, MC becomes the worst one. This because when use liner appriximation actually the Q_true got by MC is not suitable for updating all keys been sampled the trajectory.
- For MC_best, the more players are in, the more chance they will win
- For MC_epsilon, fewer players is better.
- For QL_best, fewer decks is better.
- and QL_epsilon fewer players is slightly better.
- For TD_best and TD_epsilon, more players is better

Hyperparameters

All settable hyperparameters except for m and n are assigned by the instance level config.json under the instance's folder.

Some hypperparameters has finite many choices and will be generated in the main.py when different instances are created. We will write these hyperparameters into the instance level config.json that inherited from the template config.json (under the INSTANCE folder).

- update: choices in ['MC', 'QL', 'TD']
- name: choices in the combination of form 'update-epsilon' or 'update-best' for policy being epsilon greedy policy and best policy respectively.
- policy: choices in ['epsilon_greedy_policy', 'best_policy']

We also has some higher level hyperparameters that are assigned in the template config.json. Note that these hyperparameters are the same for all instances created by call main.py once. They are:

- epochs: number of iterations.
- n_zeros: a constant for determine the value of ϵ in epsilon greedy policy
- session: denotes how often we summay the performance of a given player in plot.py. For example, if session = 1000, we summary its wins losses and draws every 1000 actions.
- linear: bool type. whether use the linear action value function approxiamation.
- alpha: learning rate when linear is True.

- Remark: if bool(linear) is True then we will append '_linear' to the update.

Visualization

We illustrate the typical plots as examples and you want to see more, please visit the subfolder with path = STORAGE/INSTANCE/pic

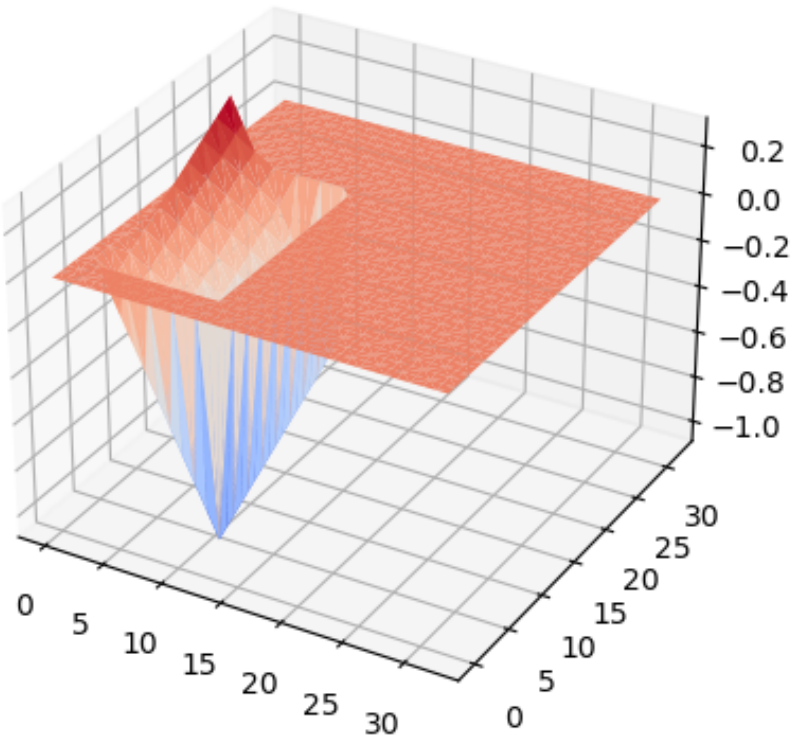
Visualization on $m = \infty, n = 2$

We only take the update=QL as example and you should refer to Blackjack/storage/m0n2/pic/ for outcomes for MC and TD

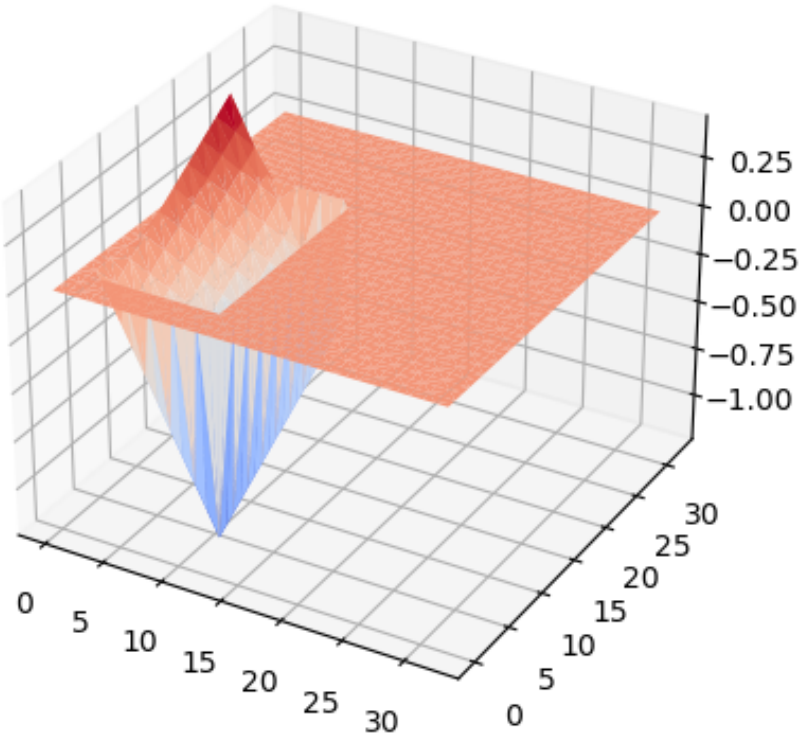
We see that the policy here is different from non-linear approximation. This is because we use very simple feature that underfit the real action value function.

Value Function Visualization

QL_best_value visualization

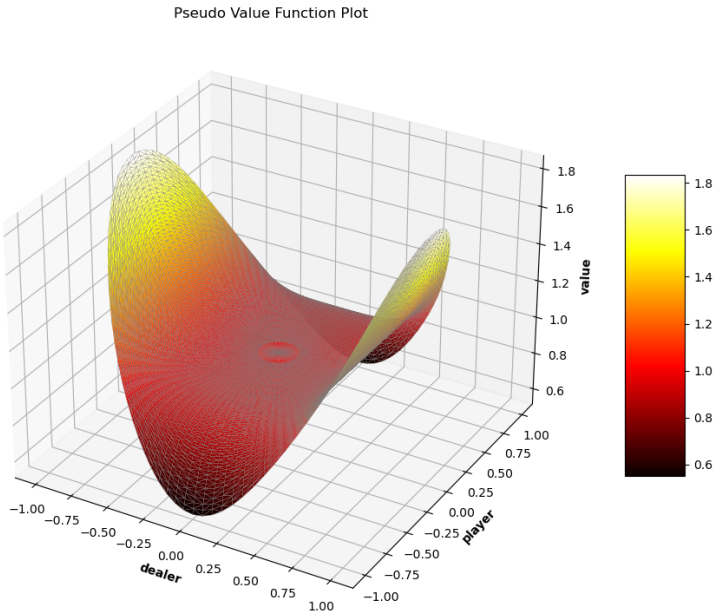


QL_epsilon_value visualization



Remark

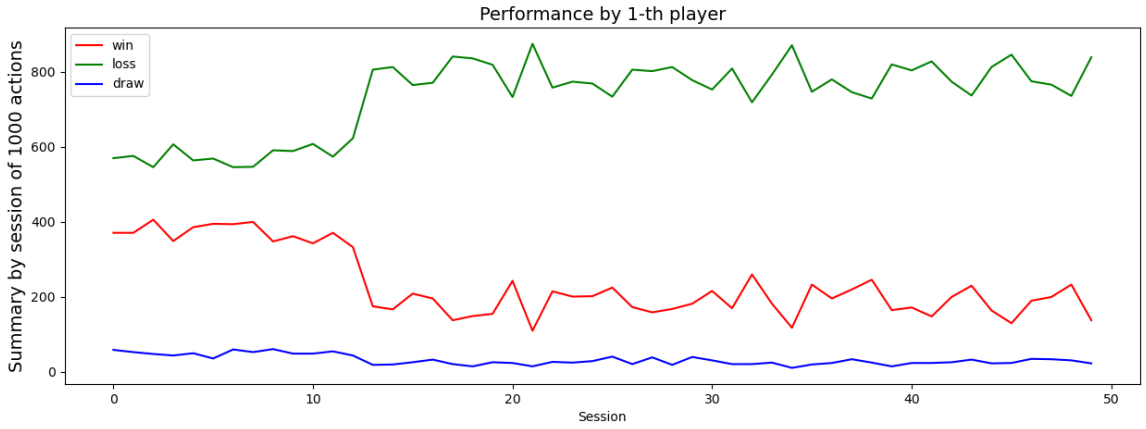
Since I forgot to add the labels for x-axis, y-axis and z-axis when doing the experiment, their position and labels are denoted by the following Pseudo Value Function Plot. All axes' arrangements in the figures of this repository follow the left-hand rule. You may refer to the following pic to identify the arrangement and meaning of the x, y, z axes.



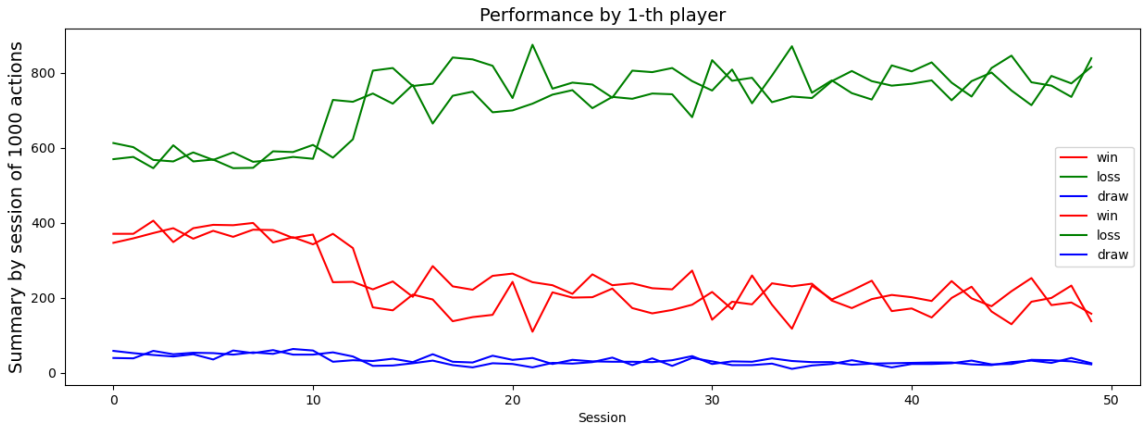
Player Performance Visualization

Visualize MC

MC_best_player_1 visualization



MC_epsilon_player_1 vs. MC_best_player_1 visualization



We see clearly that under the update rule MC, the player with epsilon greedy policy performs consistently better than they player with the deterministic best policy. The outcome shows that **expolration is important !!!**

Compare MC, QL, TD and best, epsilon

We have the following conclusions by observing the player performance visualization on update=[MC, QL, TD] and policy=[best, epsilon]

- epsilon greedy policy outperforms the best policy consistently no matter which update strategy we adopt.
- For a fixed policy, the performances of update strategies are TD>QL>MC

  The **reason** we guess is that since the TD style fit the requirements by linear action value function approximation better. Because in the setting of TD, supervised learning is easier and this improves the qualioty of linear approximated action value function.

Citation

If you use my Blackjack in any context, please cite this repository:

```
@article{
  ZHA2021:RL_Blackjack_linear,
  title={Blackjack with Action Value Function Approximation},
  author={ZHA Mengyue},
  year={2021},
  url={https://github.com/Dolores2333/Blackjack_linear}
}
```

This work is done by ZHA Mengyue for Homework2 in MATH6450I Reinforcement Learning lectured by Prof Bing-yi Jing in HKUST. Please cite the repository if you use the code and outcomes.