FAI Final Project – Texas Hold'em Casino

Name: 楊閎凱 ID: B11611047 Major: 工管 Grade: 3

1. Introduction

This final project explores different strategies for beating seven provided baseline poker agents in a heads-up Texas Hold'em environment. The project aims to compare rule-based, Monte Carlo, supervised neural network, and reinforcement learning, evaluating their effectiveness against each baseline.

2. Methods

1. Rule Base

In this method, I adopted several usual Texas Hold'em experiences into the agent such as different decision in each streets or considering the opponent's previous action. Below is the situations considering order:

- Stack < 150 :
  - All-in with poker hands stronger than High card of 10
  - If the required call is small: Call
  - Otherwise fold.
- Opponent's Large Raise: Call only with three of a kind or better; otherwise fold.
- Stack < 200:
  - Three of a kind or better: All-in.
  - None of opponents raise in flop or turn than bluff in a chance of 0.2.
  - Call when the required amount <=10.
- Preflop Street:
  - Call amount > 60: Call with pair >= J or both >= Q; otherwise, fold.
  - medium-strong pairs or suited connectors: Raise a small amount
  - Weaker hands: Call
- Postflop Streets:
  - Facing a Large Raise: Call only with two pair or better; otherwise, fold.
  - River Street:
    - With two pair or better, raise.
    - With one pair, raise in a chance of 0.5, otherwise call.
    - Opponents didn't raise: Call; otherwise fold.
  - Flop or Turn street: Raise with a pair or better.
  - Flop street: If no opponent has raised, bluff in a chance of 0.2.
  - If no opponent has raised, then call; otherwise fold.

The raise amount mentioned above is set to a specific number and modify according to the current min/max.

2. Monte Carlo

In this method, it estimates the win rate of the current status, by simulating many random community and opponent's probably card. Then it consider the the returned win rate and current street to decide the following action.

- Simulation: The agent will run simulations for the provided (`nb_simulation`) times which will generate the missing community cards and opponents hands with the cards aren't used independently. Then each hands will get a score by using the given system(hand_evaluator) and we are able to know the winner by comparing the scores. By calculating the final win rounds, we know the predicted win rate.
- Action Logic:
  - Very High Win Rate (`win_rate` > 0.9): Always choose to raise to the maximum amount. If not allowed, call.
  - High Win Rate (`win_rate` > `raise_thres`):
    Raise a moderate amount. The threshold depends on stack size:
    - If stack is very short (`my_stack` < 10 * `min_raise`): `raise_thres` = 0.7
    - If stack is very deep (`my_stack` > 100 * `min_raise`): `raise_thres` = 0.85
    - Otherwise: `raise_thres` = 0.75

- Moderate Win Rate (`win_rate` > 0.6): Raise a small amount if allowed, otherwise call.
- If the opponent kept raising until turn or river, fold unless the win rate is high.
- Low win rate on turn or river (`win_rate` < 0.3): Fold.
- Win rate exceeds the pot odds (the ratio of the current pot size to the bet required to continue): Call
- On flop, turn, or river(weak hand): Raise in a chance of 0.1 to bluff.
- Default: Otherwise, fold.

The times the simulator simulates is considered by (`nb_simulation`). The more times it simulate the more accurate the win rate may be. However, considering the time limit and efficiency, we should find the efficient number to simulate.

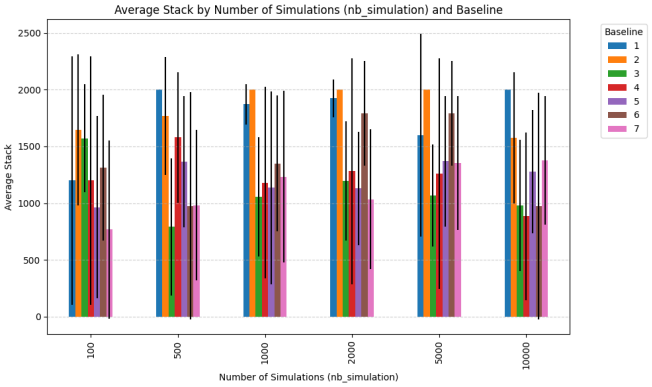

Figure a. Average stack by nb_simulation and Baselines

In graph a, each `nb_simulation` are tested 5 times with different baselines. Observing the figure, we can see that when `nb_simulation` >= 1000 the result starts to be stable and perform well. Therefore, `nb_simulation` = 1000 is chosen by considering stableness, performance, and time consumption.

3. Supervised Neural Network

In this method, I generated history playing data of the Rule Base and Monte Carlo agent by playing with each baselines. The history data is encoded and normalized.

- Data Preparation:
  - Encoding: One-hot encoding for the card faces, street name, last opponent action, and position information.
  - Normalizing: Normalize the numerical data by dividing them with possible maximum.
  - Collect 50000 hands of data. The data will be split into 8:2 for training data and validation data. After the best epoch found the validation data will be added back to train data for final training.
  - Randomly chose the player (Rule Base or Monte Carlo) and opponent (baseline1–7) per hand.
- Model training: The neural network is trained using the collected poker hand data, with a batch size set to 1024. The architecture of the network consists of three fully connected layers: the input layer is followed by two hidden layers with 128 units each, both activated by ReLU functions. The final output layer produces a five-dimensional vector representing the probability of selecting each action (fold, call, raise_min, raise_mid, raise_max). The network is trained using the Adam optimizer which learning rate set to 0.001 and cross-entropy loss. During training, if the validation loss does not improve for 20 consecutive epochs, training is stopped and the best-performing model is saved.

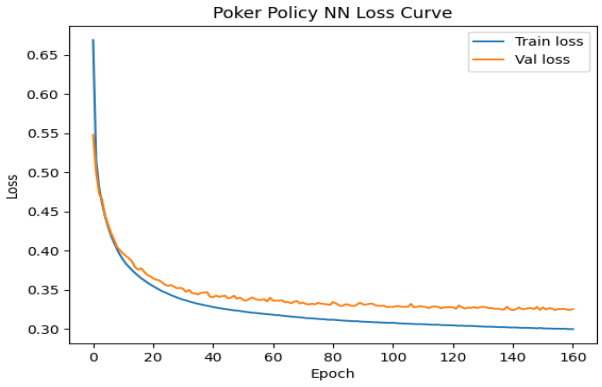

Figure b. Loss curve of neural network training

Figure b. shows that both training and validation losses decrease steadily as the number of epochs increases and begin to stabilize around 40 epochs, indicating that the model has converged.

4. Reinforcement Learning (DQN)

In this method, I implemented a Deep Q-Network to enable the agent to learn the game through self-play. Besides training the

model from random initialization, I experimented it with initializing the DQN using the neural network trained by the previous section. This approach allows the model to start with some human knowledge or intuition, potentially speeding up the learning process.

- Network Structure: The network structure is the same as the one in supervised neural network, consisting of three fully connected layers of size 128 with ReLU activations, and an output layer of dimension 5 corresponding to the available actions.
- Train Loop:
  - Load the pretrained model.
  - The transitions of each street are stored in a buffer with length 10000.
  - The agent faces a randomly chosen baseline from baseline1 to baseline7.
  - At the start of training, the agent selects actions randomly with probability epsilon = 0.3. After each training step, epsilon decays by 0.5% until reaching 0.05, gradually shifting from exploration to exploitation.
  - To increase the robustness, the network will only update the main network every 10 episodes.
  - The batch size of training is set to 64 to increase efficiency.
  - Play for 5000 and 10000 episodes.
- Reward Setting: The reward of non-ending streets is set to 0, while the ending street reward is set to the difference between the round start stack and the round end stack. With gamma set to 0.99, the agent discounts future rewards slightly to encourage long-term planning and optimize overall gains throughout the episode.

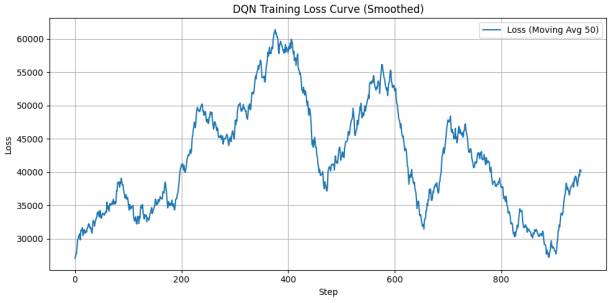

Figure c. Loss curve of Reinforcement Learning

In figure c, the loss exhibits significant fluctuations, rising and falling multiple times rather than converging smoothly. The lack of steady downward convergence may indicate that the episode number may be not enough or lack of fine tuning.

3. Experimental Results

The experiments below are conducted by testing each agent with each baseline five times.

1. Rule-Based

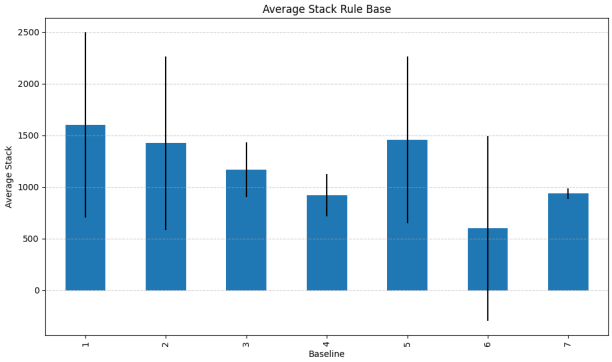

Figure d. Rule-Based Agent Average stack by Baselines

In Figure d, the rule-based agent performed well against baselines 1, 2, 3, and 5, winning most games and achieving higher average stacks. However, its performance dropped significantly against baselines 4, 6 and 7, where it won only one or no games, and the average stack was much lower. This result shows that the rule-based approach is less effective against stronger or more unpredictable opponents.

2. Monte Carlo

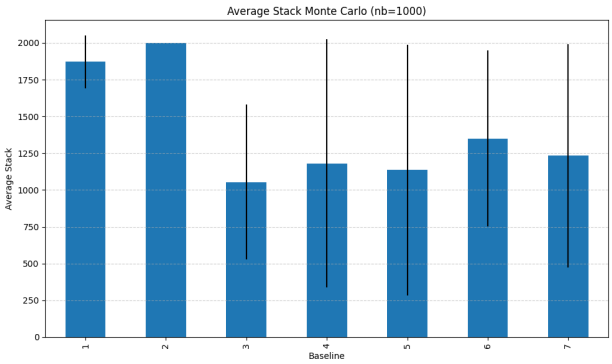

Figure e. Monte Carlo Agent Average stack by Baselines

In Figure e, the Monte Carlo agent achieved strong performance against baselines 1 and 2, winning all games and high average stack. Against baselines 3 to 7, the agent's win rate decreased, and the average stack was lower with larger variance, indicating more unstable outcomes. This may be because success in this game depends not only on hand strength or win rate estimation, but also on strategic betting or opponent behavior.

3. Supervised Neural Network

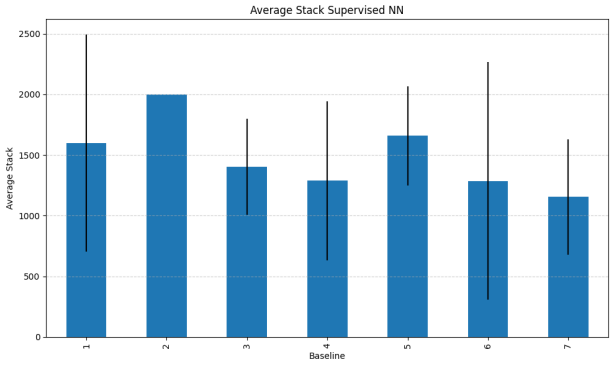

Figure f. NN Agent Average stack by Baselines

In Figure f, the NN performs well against baselines 1, 2, and 5, achieving high average stacks and win counts similar to its teacher agents. However, its performance is modest against baselines 3, 4, 6, and 7, which is similar to the result of both rule-based and Monte Carlo agents. By integrating the strengths of both agents, the NN agent achieves more robust performance on difficult baselines than either single approach alone.

4. Reinforcement Learning

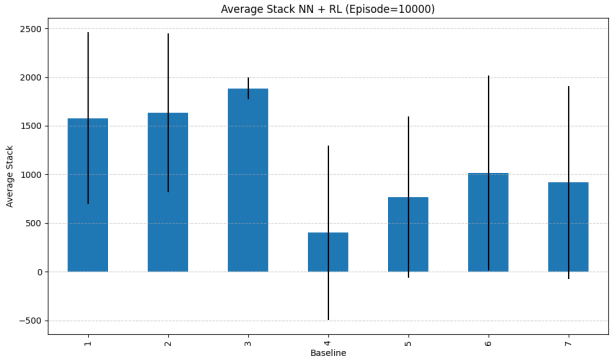

Figure g. RL Agent Average stack by Baselines

In figure g, the RL result turned out to underperform the original NN result with only baseline3 exists a huge improvement. However, the result didn't outperform the other agents above too much which may be due to the sparse rewarding mechanism and the episode wasn't large enough for RL to perceive a better playing method. These changes suggest that reinforcement learning enabled the agent to adapt to certain opponents, but also introduced greater variance and new weaknesses, highlighting the importance of fine tuning RL's parameters and need of longer episodes.

4. Discussion and conclusion

1. Pros and Cons of each models

Model	Pros	Cons
Rule Base	Fast, robust, easy to debug, high explainability	Cannot adapt to sophisticated or adaptive opponents, low flexibility
Monte Carlo	Easy to program, performs well in simple scenarios	High time consumption, win rate estimation is noisy and unstable
Supervised NN	Easy to accelerate with GPU, leverages knowledge of multiple agents	Data quality depends on teacher agents, limited adaptability
Supervised + RL	High performance on some baseline	Long training time, sample inefficient; require more tuning

2. Final Decision of agent

Considering the result in the third section, the agent trained by supervised neural network has the best performance in average. Therefore, the final agent chosen is the supervised one.