No Thanks! and Take 5 with Reinforcement Learning

Udesh Habaraduwa

1 Introduction

Reinforcement learning (RL) offers a paradigm where agents can learn optimal strategies directly from interaction with the environment (including other agents), without relying on pre-existing knowledge or heuristics about the task at hand [4, 5].

Several options are available for reinforcement learning (e.g Q-learning, Deep Q-networks, Proximal Policy Optimization), varying in complexity, resource requirements, and performance. For this task, Q-learning (tabular and linear function approximation) with an epsilon-greedy strategy was selected for several reasons:

- Q-learning: Q-learning is a relatively simple learning strategy that is straightforward to implement as compared to neural network-based methods like Deep Q-networks. It offers a balance of exploration and exploitation without the overhead of more complex architectures.
- Epsilon-greedy : A simple strategy for balancing exploration and exploitation. With some probability ϵ, the agent chooses a random action (exploration) otherwise choosing an action based on the best estimated Q-value (exploitation).
- Speed: Generating responses from the agent using Q-learning is very fast as it is simply a dictionary lookup (in the tabular case) or a matrix-multiplication in the case of linear function approximation as compared to neural network inference in methods like Deep Q-networks which can be computationally intensive to run and require more processing time.
- State space size: Compared to games like chess, the two games under consideration have relatively small state spaces (subject to the choice of state space representation). This makes Q-learning more feasible as it can effectively learn and map these states to values without requiring the immense memory and computational power that larger games might necessitate.

2 No Thanks! with tabular Q-learning

"No Thanks!" [1] is a strategic card game where players aim to avoid collecting cards, each of which has a point value. Throughout the game, players decide whether to take the current face-up card or to pass, hoping to avoid cards with higher points. The game's simple yet strategic decision-making process presents an intriguing challenge for reinforcement learning.

Tabular Q-learning is a one of the foundational reinforcement learning techniques [6]. The term "tabular" in this context refers to the method of representing the value of state-action pairs in a table (Q-table), where each entry in the table represents the estimated value (or Q-value) of taking a particular action from a particular state. The Q-table acts as a lookup table where the agent can find the best action to take in a given state based on the current estimates of the Q-values. As applied here, the agent must return an action choice of taking a card (return True) or passing (return False). The reward function calculates the reward based on whether a player decides to take or pass a card. If taking the card, the reward is based on the card's coins, adjusted for any change in penalty and the player's average penalty over the games played so far, with an added incentive for completing or continuing a card sequence. If passing the card, the reward is determined by the player's available coins and the potential penalty difference, with a strategic reward for passing and a penalty for being forced to take a card. In situations where the Q-table is empty for a given state-action or the agent is exploring, the agent plays the strategy of the best default AI (takeSmallAgent) instead of exploring a completely random state space.

Given the following variables:

- R: Reward for the action.
- C: Coins on the card.
- P_c: Current penalty.
- P_f: Future penalty when taking the card.
- P_d: Penalty difference, defined as $P_d = P_f - P_c$.
- P_a: Average penalty accumulated.

The reward function, R, is defined as:

$$R = \begin{cases} C - (|P_d| + P_a + 50) & \text{if taking and } P_d > 0, \\ C - (|P_d| + P_a - 100) & \text{if taking and } P_d \leq 0, \\ -(|P_d| + P_a + 50) & \text{if passing and } P_d \leq 0, \\ -(|P_d| + P_a - 50) & \text{if passing and } P_d > 0. \end{cases}$$

The Q-table is initialized to zero, and as the agent interacts with the environment, the Q-values are updated iteratively using the Q-learning update rule [5, Chapter 6], defined as :

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (1)$$

Where:

- $Q(s,a)$: Current Q-value estimate for state s and action a.
- α: Learning rate, which determines how much of the new Q-value estimate we adopt.

- r: Reward received after taking action a in state s.
- γ: Discount factor, which models the agent's consideration for future rewards. A value of 0 makes the agent short-sighted by only considering current rewards, while a value closer to 1 gives the agent a longer-term reward perspective.
- s': New state after taking action a in state s.
- $\max_{a'} Q(s', a')$: The maximum Q-value for state s' over all possible actions a'.

The following state representation was devised for simplicity:

```
``RL high card;RL low card;
RL coins;card on offer;
coins on the card"
```

As a baseline, agent was trained on 10 million games ($\alpha = 0.01, \epsilon = 1.0, \gamma = 0.9$) against the provided default agents with the lienear ϵ decay of 0.0001 every 1000 games after 1 million games have been played.

3 Take 5 with Q-learning function approximation

Take 5! [2] is a tactical card game where players seek to minimize the points they accumulate through the cards they collect. In each round, players simultaneously choose a card from their hand and reveal it. Cards are then added to rows based on their numerical order, but a row that receives a sixth card forces the player to take the previous five, each adding to their point total. The goal is to finish the game with the fewest points possible. The agent must decide on which card to play (out of n possible cards in their hand) and, if forced by the conditions on the board, decide on which row (out of 4) to play their chosen card on.

Linear Function Approximation (LFA) in Q-learning is an extension of traditional tabular Q-learning that allows for generalization across states and actions, making it more scalable to problems with larger state or action spaces [5, Chapter 9]. Unlike tabular Q-learning where a separate Q-value is learned for each state-action pair, LFA represents Q-values as a linear combination of features of the state and action. The following equation represents the Q-value for a state-action pair as a linear combination of feature values and weights.

$$Q(s, a) = \theta^T \cdot \phi(s, a) \tag{2}$$

Here:

- s and a denote the state and action, respectively.
- θ is the vector of weights, which determine the contribution of each feature to the Q-value.
- $\phi(s, a)$ is the feature vector derived from the state and action, representing the characteristics or attributes that are important for estimating the value.

To learn the optimal policy, the agent interacts with the environment and updates the weight vector θ using the following rule:

$$\theta \leftarrow \theta + \alpha \cdot (r + \gamma \cdot \max_{a'} \theta^T \cdot \phi(s', a') - \theta^T \cdot \phi(s, a)) \cdot \phi(s, a) \tag{3}$$

Where:

- α is the learning rate, determining the extent to which the new knowledge overrides the old.
- r is the reward received after taking action a in state s.
- γ is the discount factor, capturing the agent's consideration for future rewards.
- s' is the subsequent state after taking action a in state s.
- $\max_{a'} \theta^T \cdot \phi(s', a')$ represents the maximum estimated Q-value for the next state s' over all potential actions a'.

This update rule strives to minimize the difference between the predicted Q-values (estimated using the current weight vector) and the true Q-values (observed from the environment).

In LFA, as opposed to tabular Q-learning, there is an aspect of feature engineering involved. The Q-value for a given state is calculated as a linear combination of some features derived from the state of the game. These features have to be derived by the developer based on heuristics or game relevant knowledge. As the agent plays, the weighting of each feature is updated.

At present, LFA is implemented only for the card selection. For the row selection, ten percent of the time the agent plays a random row but otherwise chooses to always pick the row with the lowest penalty. The features for the state of the game for the card selection weights are as follows:

- Value of the card
- Penalty on the card being considered
- The difference between the card being considered and the highest card on the table
- Length of the nearest row to the card being considered
- The total penalty in the row being considered
- Average penalty accumulated thus far

Given the following variables:

- R: Reward for the action.

- P_{next}: Penalty in the next hand.
- P_{prev}: Penalty in the previous hand.
- L: Length of the nearest row.
- P_{row}: Penalty in the nearest row.

The reward function, R, is defined as:

$$R = \begin{cases} P_{\text{prev}} - P_{\text{next}} & \text{if } L \neq 5, \\ -(P_{\text{row}}) & \text{if } L = 5. \end{cases} \quad (4)$$

The agent was trained against the default AIs for 1 million games, with linear ϵ decay of 0.0001 every one thousand games after 900,000 games have been played.

4 Results & Conclusion

4.1 Results

The performance of agents after training for NoThanks! and Take 5 are shown in table 1 and 2 respectively.

Table 1: Performance of Different Agents : Take 5

Name	Games	Average
RL Agent	1000	10.912
HighToLow	1000	11.738
AvoidPenalty	1000	12.291
LowPenalty	1000	14.414
LowToHigh	1000	14.619
HighToLowPenalty	1000	15.671
LowToHighPenalty	1000	15.696
RandomPlayer	1000	18.525

Table 2: Performance of Different Agents: No Thanks!

Name	Games	Average
TakeSmall	1000	32.543
LowerPenalty	1000	34.744
CoinsDependent	1000	35.305
RL Agent	1000	36.589
CutOff14	1000	44.086
Random Player	1000	68.480

4.2 Conclusion

The results show that the tabular Q-learning approach maybe unsuitable for the game of No Thanks! with the agent unable to out perform the hard-coded strategies even after ten million games. On the other hand, the LFA approach for Take 5 was successful at out performing all the hard-coded agents with only one million games played. It's possible that matters might be improved by calibrating the reward function and introducing a more detailed state space representation for both approaches. However, given the success of LFA in Take 5, it would be worth trying it on No Thanks! as well.

The primary challenges faced for the present project were:

- Hyperparameter optimizaztion: Both strategies involve several hyper parameters the tuning of which can impact the agent's ability to learn fast enough (α), appropriately weight future rewards (γ), and manage the balance between exploitation and exploration (ϵ). Indeed, these 3 variables alone represent a large search space for possible configurations and finding the optimal balance is a known challenge in reinforcement learning [5, Chapters 3, 5, and 6].
- Randomness: Both games contain a high degree of randomness (e.g., in the card drawn from the deck) which makes for a large search space. The larger the search space, the more it needs to be explored which in turn means more computation time.
- State representation: Devising an appropriate state space representation was a challenge. The state representation needs to be sufficiently detailed to capture meaningful aspects of the environment, but not so complex that it makes learning slow or infeasible. Indeed, for No Thanks! the state space is extremely simple. For example, an actual player would be able to see all of their cards, not just the lowest and highest.
- Reward function: The reward function needs to be able to capture the benefits that are immediate or only a few steps away without sacrificing positions that are beneficial in the long term.

Reinforcement learning is an approach that requires an extensive amount of trial and error, not only in part of the agent but also in part of the developer as various hyperparameters, state representations and reward functions need to be considered. Indeed, the appeal of deep neural models (e.g., Deep Q-learning), over the methods used herein, lies in their potential to automatically extract relevant features from raw inputs, mitigating the need for extensive hand-crafting of features or state space representations. However, these models introduce their own complexities (e.g., non-stationary targets [4]) as well as the challenges associated with training deep (reinforcement) networks in general (e.g., sparse rewards) [3]. It is feasible that with exploration in the search space of these architectural variables and more time spent training on the task, and perhaps the inclusion of deep neural networks, superior performance can be acheived on the games at hand.

References

[1] Thorsten Gimmler. *No Thanks!* Card game. 2004.

[2] Wolfgang Kramer. *Take 5! (6 nimmt!)* Card game. 1994.

[3] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[4] David Silver et al. "Mastering the game of go without human knowledge". In: *nature* 550.7676 (2017), pp. 354–359.

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[6] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989).