Q-Learning

Peter (Petr) Ladur

56445760

Chapter 1

Introduction

The aim of this project is to explore Q-Learning in the context of tic-tac-toe and other applications using python. Theoretical Q-Table was generated using the "minimax" algorithm, and was compared to Q-Tables generated through training against both, random and perfect opponent. Results of agents trained on different policies playing against each other were recorded. How hyperparameters (α, τ) affect training results was investigated and an optimal function for hyperparameters was proposed. Deep-Q-Learning was investigated in the context of different games in the "Open-AI gymnasium".

Chapter 2

Background

2.1 Q-Table

Q-Table is a table which contains contains expected outcomes after every possible action in a particular state. In the context of tic-tac-toe the state is the game position at that particular moment, and actions are all the empty squares. As seen in table 2.1, for that particualr state there are two actions that lead to a win and three actions which lead to a draw (assuming perfect play).

	(1, 1)	(1, 2)	(1, 3)	(2, 1)	(2, 2)	(2, 3)	(3, 1)	(3, 2)	(3, 3)
_ _ o _ x _ x o _		1	1			0	0	0	

Table 2.1: Two actions lead to a win, three actions lead to a draw assuming perfect play from both sides

2.2 Q-Learning

In the context of Q-Learning, the Q-Table can be filled up by playing games and updating the values of actions taken during that game according to the Bellman equation.

$$Q(s_t, a) \leftarrow (1 - \alpha) \cdot Q(s_t, a) + \alpha(r + \gamma \cdot \max_a Q(s_{t+1}, a)) \tag{2.1}$$

Where:

- $Q(s, a)$ is the predicted outcome for the action at a particular state s
- α is the learning rate

- γ is the discount factor
- r is the immidiate reward for achiveing a state

The learning rate α determines how much does the new information replace the already known information. A learning rate of 1 would that each time training iteration the $Q(s, a)$ is completely replaced by $(r + \gamma \cdot \max_a Q(s_{t+1}, a))$. A learning rate of 0 would imply that $Q(s, a)$ is not affected by training any more. The learning rate can be adjusted during training as the model becomes more stable.

The immidiate reward in the context of tic-tac-toe is 1 for a game that is won by X, 0 for a game that is drawn and -1 for a game that is won by O.

Chapter 3

Q-Learning on tic-tac-toe

3.1 Generating theoretical Q-Table

The "minimax" algorithm can be used in turn-based games to calculate the best policy. It is designed to minimise the potential loss and maximise the potential gain. The algorithm works by first generating all the possible final states and then taking turns "undoing" the moves that could have lead to that state by removing **X**'s and **O**'s until the initial state (empty board) is reached. Each time an **X** or **O** is removed the worst case scenario is considered, filling up the theoretical Q-Table.

```
terminal_states = generate_all_terminal_states()

for parent_state in parent_states:

    for x/o in parent_state:

        child_state = parent_state with an x/o removed

        if child_state is terminal_state:
            Q_Table[child_state] = [game_result] * 9
        else:
            \\min or max depends on whether it's x/o removed respectively
            Q_Table[child_state][index of x/o] = max/min(Q_Table[parent_state])

            parent_states = child states
```

3.2 Q-Learning implementation

During the Q-Learning algorihtm batches of games were played. The agent would choose moves using a weighted probability values according to a slightly

modified bolztman function as seen in (3.1). States s, actions taken a and results r are stored in a queue. The queue is then emptied and Q-Table is updated using the Bellman equation as seen in (3.2) and (3.3).

$$p_i(\tau) = \frac{\tau^{\epsilon_i}}{\sum_{j=1}^{9} \tau^{\epsilon_j}} \tag{3.1}$$

If terminal state:

$$Q(s_t, a) \leftarrow (1 - \alpha) \cdot Q(s_t, a) + \alpha r \tag{3.2}$$

If non terminal state:

$$Q(s_t, a) \leftarrow (1 - \alpha) \cdot Q(s_t, a) + \alpha \max_a Q(s_{t+1}, a) \tag{3.3}$$

As agents learn their drawrate or winrate depending on if they are playing against a perfect or random opponent increases in the form of the decaying exponential as seen in . **X** learns against optimal **O** the draws increase in the form of a decaying exponential draws $\% = e^{-k \cdot \text{games played}}$ as seen in Figure 3.1.

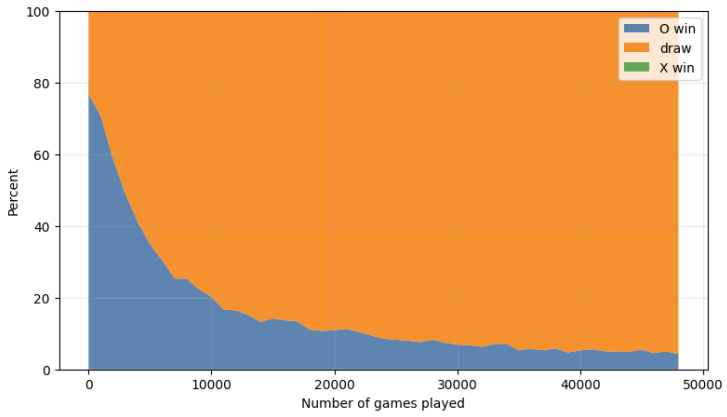

Figure 3.1: **X** training against perfect **O** causes the drawrate in the form of a decaying exponential

3.3 Variations in α and τ

The values of hyperparameters τ and α matter significantly. Different models for τ and α have been tested and two models are proposed as best candidates as seen in

$$\alpha(\text{turn}, a) = \alpha_{inital} \cdot \frac{1}{1 + a \cdot \text{turn}} \tag{3.4}$$

$$\tau(\text{turn}, \text{total games}, t) = 1 + (t - 1) \cdot (\frac{\text{turn}}{\text{total games}})^3 \tag{3.5}$$

3.4 Optimal and non optimal opponent

Agent	**X** trained on perfect	**X** trained on random
O trained on perfect	**X** winrate: drawrate: **O** winrate:	**O** winrate:
O trained on random	Value 2,1	Value 2,2

Table 3.1: Matrix of results of agents trained on different opponents playing against each other.

Chapter 4

Deep Q-Learning

4.1 Neural Network and the loss fucntion

4.2 cart-pole game

4.3 moon-landing game

Chapter 5

Conclusion

Chapter 6

Bibliography