Q-Learning

Peter (Petr) Ladur

Chapter 1

Introduction

The aim of this project is to explore Q-Learning in the context of tic-tac-toe and as other applicationsusing python. How variations in hyperparameters (α, τ, r) affect results was investigated to find optimal parameters for different scenarios. Deep-Q-Learning was explored in the context of a game and in the context of the stock market (haven't done yet, but will do once finish the report).

1.1 Q-Table

Chapter 2

Background

2.1 Q-Learning

Q-Learning is a form of reinforcement used in environment with discrete states.
It relies on the Q-Table to store predicted expected values for each state $Q(\text{state}, \text{action})$.
Initially the Q-Table can be initialised to random values, and after repeatidly exploring different states updating the predicted expected values for each state according to the "Bellman equation"

$$Q_{new}(s_t, a) = (1 - \alpha) \cdot Q(s_t, a) + \alpha(r + \gamma \cdot \text{max} Q(s_{t+1}, a)) \tag{2.1}$$

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

2.2 Tic-Tac-Toe

Tic-Tac-Toe is a simple two-player game on a 3 by 3 grid. The game originated in ancient Egypt atleast 1300 BC. Players take turns placing X's and O's in the grid with the goal of getting 3 in a row. The game is drawn with perfect play from both sides, but O's have to be precise to garantee a draw. Using the "minimax" algorithm -assuming the oppponent will play the optimal move- a theoretical Q-Table can be derived.

2.3 Software

The project heavily used python NumPy package to efficiently apply mathematical operations on arrays.

Chapter 3

Q-Learning on tic-tac-toe

3.1 Algorithm

3.2 Optimal and non optimal opponent

3.3 Variations in *tau*, *alpha* and *r*

Chapter 4

Deep Q-Learning

Chapter 5

Conclusion

Chapter 6

Bibliography