COMP3702 Tutorial 10

Matt Choy | matthew.choy@uq.edu.au

Deep Q-Networks

- If we don’t know the system dynamics, should we take exploratory actions (that give us more information about the environment) or exploit current knowledge to perform as best as we can?
- Using a greedy policy (exploiting current knowledge), bad initial estimates in the first few cases can drive policy into a sub-optimal region, and never explore further
- Instead of acting according to a greedy policy, we act according to a sampling strategy that will explore (state, action) pairs until we get a “good” estimate of the value function

Deep Q-Networks

Deep Q-Networks (DQNs) approximate a state-value function in a Q-Learning framework using a neural network. In the case of Atari Games (like Breakout shown in lectures), they may take in several frames of the game as an input and output state values for each action as an output. More generally, the neural network learns to transform a state to estimated Q-values for each possible action, i.e. $Q(s, a)$.

a) Consider the CartPole environment of the OpenAI Gym, where the objective is to move a cart left or right in order to balance an upright pole for as long as possible as in Figure 1.

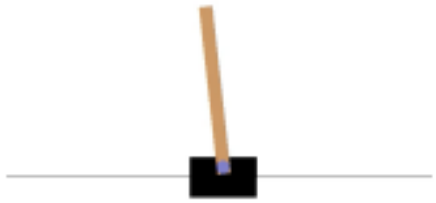

Figure 1: CartPole-v1 from Open AI Gym

Exercise 11.1

The Reinforcement Learning states, actions and rewards can be formalised as follows:

- The **state** is specified by four parameters (x, v, θ, ω), where:
  - x: the horizontal position of the cart (+ve = right)
  - v: the horizontal velocity of the cart (+ve = moving to the right)
  - θ: the angle between the pole and the vertical position (+ve = clock-wise)
  - ω: angular velocity of the pole (+ve = rotating clock-wise)
- The **actions** that the agent can perform are:
  - 0: push the cart to the left
  - 1: push the cart to the right
- The game terminates ("is done") when the pole deviates more than 15 degrees from vertical ($|\theta| \geq \pi/12 \approx 0.26$). In each time step, if the game is not "done", then the cumulative "reward" increases by 1. The goal of the game is to accumulate the highest cumulative reward.

Q: Explain why standard Q-learning using a table of state-action values can't be used in this environment?

Exercise 11.1a

Q: Explain why standard Q-learning using a table of state-action values can't be used in this environment?

- The state in the cartpole environment is uniquely identified by four parameters:
$$(x, v, \theta, \omega)$$
- These four values are continuous in nature.
- To update the Q-Values in our table-based Q-Learning algorithm, we would need to discretise the values.

Exercise 11.1a

Q: Explain why standard Q-learning using a table of state-action values can't be used in this environment?

- The state in the cartpole environment is uniquely identified by four parameters:
$$(x, v, \theta, \omega)$$
- These four values are continuous in nature.
- To update the Q-Values in our table-based Q-Learning algorithm, we would need to discretise the values.

- Instead, the values can be input into a value function approximator, and performing function approximation (using a neural network), to estimate the value of each action for a given state.

Exercise 11.1b

b) Consider that the CartPole is now controlled by an analog joystick, where instead of only being able to move the cart left and right, you may move faster left or faster right (the actions are now continuous). What is a limitation of the output format of Deep Q-Networks for this problem? What alternative algorithm could provide a solution?

- Deep Q-Networks only output discrete values, and therefore cannot output analogue values.
- We could use Policy Gradients, which can output continuous probability distributions
- We could describe the output as how fast to move on the horizontal axis.