THE UNIVERSITY OF QUEENSLAND
AUSTRALIA
CREATE CHANGE

COMP3702

Tutorial 10

Semester 2, 2022

Aryaman Sharma (aryaman.sharma@uq.edu.au)

The University of Queensland
School of Information Technology and Electrical Engineering

Multi-armed Bandits

Exploration vs Exploitation

- We have the fundamental question **: if we don't know the system dynamics, should we take exploratory actions that will give us more information, or exploit current knowledge to perform as best we can?**
- If we use a **Greedy policy**, bad initial estimates in the first few cases can drive policy into sub-optimal region, and never explore further.
- Key idea: instead of acting according to greedy policy, act according to a sampling strategy that will explore state-action pairs until we get a "good" estimate of the value function.

Multi-Armed Bandit Problem

- Assumptions:
  - Choice of several arms/ machines
  - Each arm pull is independent of other arm pulls
  - Each arm has a fixed, unknown average payoff
- Which arm has the best average payoff?
- How do we maximise the sum of rewards over time?

Multi-Armed Bandit Problem: key idea

- We want to **explore** all arms. **BUT** by exploring too much we sacrifice rewards.
- Want to **exploit** promising arms, **BUT** can get stuck with sub-optimal values.
- Want to balance between **exploration** and **exploitation**

Exploration Strategies

- ϵ-**greedy strategy**: Choose random action with $\mathbb{P}$ ϵ and choose best action with probability 1-ϵ.
- **Upper confidence bounds:** Take into account average plus variance information.

UCB1 algorithm (Auer et al 2002)

1. Pull every arm $k \geq 1$ times, then
2. at each time step, choose arm i that maximises the UCB1 formula for the upper confidence bound

$$UCB1_i = \hat{v}_i + C\sqrt{\frac{\ln(N)}{n_i}}$$

where

- $\hat{v}_i$ is the current value (mean) estimate for the arm i,
- C is a tunable parameter,
- N is the total number of arm pulls, and
- n_i is the number of times arm i has been pulled.

Q-learning

Asynchronous VI for MDPs, storing Q(s,a)

If we knew the model, we would have:

- a reward function $R(s, a, s')$, and
- a transition function $P(s'|s, a)$

and we could use value iteration to compute the optimal policy:

Initialize a table of Q-values, $Q(S, A)$, arbitrarily

Repeat forever:

1. Select state s, action a
2. $Q(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma \max_{a'} Q(s', a') \right)$

The catch is, now we don't know $P(s'|s, a)$ or $R(s, a, s')$...

Model-based RL

When we don't have $P(s'|s,a)$ or $R(s,a,s')$, there is a simple approach: just estimate the MDP from observed data.

Agent acts in the world (according to some policy), and observes experience:

$$s_0, a_0, r_0, s_1, a_1, r_1, \ldots s_m, a_m, r_m$$

Form the empirical estimate of the MDP via the counts:

$$\hat{P}(s'|s,a) = \frac{\sum_{i=0}^{m} \mathbf{1}(s_i = s, a_i = a, s_{i+1} = s')}{\sum_{i=0}^{m} \mathbf{1}(s_i = s, a_i = a)}$$

$$\hat{R}(s) = \frac{\sum_{i=0}^{m} \mathbf{1}(s_i = s) r_i}{\sum_{i=0}^{m} \mathbf{1}(s_i = s)}$$

where $\mathbf{1}(\cdot)$ is an indicator function $=1$ if the condition is true.

24

Model-based RL

When we don't have $P(s'|s,a)$ or $R(s,a,s')$, there is a simple approach: just estimate the MDP from observed data.

Agent acts in the world (according to some policy), and observes experience:

$$s_0, a_0, r_0, s_1, a_1, r_1, \ldots s_m, a_m, r_m$$

Form the empirical estimate of the MDP via the counts:

$$\hat{P}(s'|s,a) = \frac{\sum_{i=0}^{m} \mathbf{1}(s_i = s, a_i = a, s_{i+1} = s')}{\sum_{i=0}^{m} \mathbf{1}(s_i = s, a_i = a)} = \frac{\#\text{times we end up in } s' \text{ via action } a \text{ from state } s}{\#\text{times we've tried action } a \text{ from state } s}$$

$$\hat{R}(s) = \frac{\sum_{i=0}^{m} \mathbf{1}(s_i = s) r_i}{\sum_{i=0}^{m} \mathbf{1}(s_i = s)} = \frac{\text{sum of rewards achieved in state } s}{\#\text{times we've been in state } s}$$

where $\mathbf{1}(\cdot)$ is an indicator function $=1$ if the condition is true.

Now solve the MDP $\langle S, A, \hat{P}, \hat{R} \rangle$, e.g. using value iteration

Model-based RL

Model-based RL will converge to correct MDP (and hence correct value function / policy) given enough samples of each state

How can we ensure we get the "right" samples? (a challenging problem for all methods we present here, stay tuned)

Advantages (informally): makes "efficient" use of data

Disadvantages: requires that we build the actual MDP models, which is not much help if state space is too large

Q-learning: pseudocode

Iteratively estimate the table $\hat{Q}(S, A)$ from experience:

initialise $\hat{Q}(s, a)$ arbitrarily (i.e. all zeros)

observe current state s

repeat for each episode, until convergence:

select and carry out an action a

observe reward r and state s'

$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \left(r + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a) \right)$

$s \leftarrow s'$

for each state s

$\pi(s) = \arg\max_a \hat{Q}(s, a)$

return $\pi, \hat{Q}$

Q-learning gridworld

10.3

10.3

Exercise 10.3.

a) List the dimensions of complexity for the Gridworld reinforcement learning environment on the following page:
Modularity, Planning horizon, Representation, Computational limits, Learning, Sensing uncertainty, Effect uncertainty, Preference, Number of agents, Interaction.

b) In RL, what is the connection between the environment's state transition function and the exploration policy used by the agent?

10.3a

For the Gridworld environment, we have the following dimensions of complexity:

Dimension	Values
Modularity	**flat**, modular, hierarchical
Planning horizon	non-planning, finite stage, **indefinite stage, infinite stage**
Representation	**states, features**, relations
Computational limits	**perfect rationality**, bounded rationality
Learning	knowledge is given, **knowledge is learned**
Sensing uncertainty	**fully observable**, partially observable
Effect uncertainty	deterministic, **stochastic**
Preference	goals, **complex preferences**
Number of agents	**single agent**, multiple agents
Interaction	offline, **online**

In RL in general, we may have other variants, e.g. partially observable, multi-agent RL, offline, etc.

10.3b

In Reinforcement learning problems, when the transition function is unknown / uncertain, we should have a higher exploration term to learn information about the environment. As it becomes more certain, we should lower the exploration term

10.6

10.6

In reinforcement learning problems, often the goal is to learn the value function V^π from episodes of experience under policy π. Recall that the *return* is defined as the total discounted reward: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots \gamma^{T-1} R_T$ and the value function $V^\pi(s) = E[G_t \mid S_t = s]$, is the expected return.

a) Compare the state value ($V(S_t)$) update formula for Monte Carlo vs Temporal Difference (TD) reinforcement learning with a 1-step lookahead, i.e. TD(0).

b) Compare the update formula for the state value, $V(S_t)$, in TD learning vs the update formula for $Q(s, a)$ in Q-learning.

c) Consider Q-learning with linear value function approximation, where:

$$Q(s, a) = \sum_i^n f_i(s, a) w_i = w_1 f_1(s, a) + w_2 f_2(s, 1) + \cdots + w_n f_n(s, a).$$

What does the update function for Q-learning with linear q-functions become?

10.6a

(a) In MC and TD formulas, we have: $V(S_t) \leftarrow V(S_t) + \alpha[\text{TARGET} - V(S_t)]$, but the target differs in each:

- In the **Monte Carlo** reinforcement learning algorithm, we use the true final reward (return) G_t from the completion of the episode to update the value for each state:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)],$$

- In **TD(0)** learning we don't need to wait till the end of an episode to update $V(S_t)$, and instead use a one step look ahead based on the current estimates, which is also called *bootstrapping*. That is, at time $t+1$, the TD method uses the observed reward R_{t+1} and immediately forms a TD target $R_{t+1} + \gamma V(S_{t+1})$, updating $V(S_t)$ with the TD error. Therefore, the TD(0) update is defined as:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

10.6b

(b) For Q-learning we replace $V(S_t)$ with $Q(s, a)$. In addition, the value at the next step is estimated from the *optimal* policy, where we use the action that maximises the Q-value i.e. $V(S_{t+1})$ becomes $\max_{a'} Q(s', a')$:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R_{t+1} + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

10.6c

(c) In linear value function approximation, updates to $Q(s, a)$ are replaced by updates to the weights w:

$$w_i \leftarrow w_i + \alpha\delta f_i(s, a),$$

where $\delta = [R_{t+1} + \gamma \max_{a'} Q_{\bar{w}}(s', a') - Q_{\bar{w}}(s, a)]$, and $Q_{\bar{w}}$ is the Q-table indexed by features.

This is effectively adjusting the feature weights, w_i, to reduce the difference between the sampled value and the estimated expected value.

10.7

10.7

Exercise 10.7. Consider a reinforcement learning problem in the game of Pacman with linear value function approximation and Q-learning. The actions available to the agent are 'Up', 'Down', 'Left', 'Right'. Assume we observe an 'Up' action where Pacman's next state would result in being eaten and receiving a reward of -500, and that we are using two linear features to represent $Q(s,a)$: a feature representing the proximity to food, f_{food}, and the other representing the proximity to ghosts, f_{ghost}.

Using linear function approximation, update the weights, using a learning rate $\alpha = \frac{1}{250}$, given the following:

$$Q(s,a) = 4.0 f_{food}(s,a) - 1.0 f_{ghost}(s,a)$$

$$\begin{aligned} f_{food}(s, Up) &= 0.5 \\ f_{ghost}(s, Up) &= 1.0 \\ Q(s,a) &= +1 \\ R_{t+1} = R(s,a,s') &= -500 \end{aligned}$$

10.7

$$
\begin{aligned}
\delta &= -501 \\
w_i &\leftarrow w_i + \alpha\delta f_i(s,a) \\
w_{food} &\leftarrow 4.0 + \alpha[-501]0.5 \\
w_{ghost} &\leftarrow -1.0 + \alpha[-501]1.0
\end{aligned}
$$

Therefore, the updated weights in the Q function are:

$$Q(s,a) = 3.0 f_{food}(s,a) - 3.0 f_{ghost}(s,a)$$