THE UNIVERSITY OF QUEENSLAND
AUSTRALIA
CREATE CHANGE

COMP3702/COMP7702
Artificial Intelligence

Module 4: Learning to act — Part 1

Dr Archie Chapman

Semester 2, 2020

The University of Queensland
School of Information Technology and Electrical Engineering

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati

Week 10: Logistics

- **Assignment 3 due Friday Oct 23**
- If you haven't seen them, Tutorials 8 and 9 will help you with Assignment 2, see the code on BB
- Marking Assignment 2 is ongoing
- RiPPLE round 3 closes this evening
- RiPPLE round 4 opens tomorrow
- Assignment 4 will be released soon, due date **extended** to Monday Oct 16

Table of contents

Overview of Module 3 — Reasoning and Planning under uncertainty

Learning Objectives - Reinforcement Learning

At the end of the class you should be able to:

- Explain the explore-exploit dilemma and solutions
- Explain the relationship between decision-theoretic planning (MDPs) and reinforcement learning
- Explain the difference between model-based and model-free reinformcement learning
- Implement basic state-based reinforcement learning algorithms: Q-learning and SARSA
- Explain the difference between on-policy and off-policy reinforcement learning

Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world
  possible actions
- **Observations** current state of world
  immediate reward / punishment
- **Goal** act to maximize accumulated (discounted) reward

Like decision-theoretic planning, except model of dynamics and model of reward not given.

Assumptions on environment for reinforcement learning in Module 4

- **flat** or modular or hierarchical
- **explicit states** or **features**
- static or finite stage or **indefinite stage or infinite stage**
- **fully observable** or partially observable
- deterministic or **stochastic** dynamics
- goals or **complex preferences**
- **single agent** or multiple agents
- knowledge is given or **knowledge is learned**
- **perfect rationality** or bounded rationality

Reinforcement Learning Examples

- Game — reward winning, punish losing
- Dog — reward obedience, punish destructive behaviour
- Robot — reward task completion, punish dangerous behaviour

Experiences

- We assume there is a sequence of experiences:

$$state, action, reward, state, action, reward, ...$$

- The agent has to choose its action as a function of its history.
- At any time it must decide whether to
  - **explore** to gain more knowledge
  - **exploit** knowledge it has already discovered

Why is reinforcement learning hard?

- What actions are responsible for a reward may have occurred a long time before the reward was received.
- The long-term effect of an action depend on what the agent will do in the future.
- The explore-exploit dilemma: at each time should the agent be greedy or inquisitive?

Reinforcement learning: main approaches

1. Learn a model consisting of state transition function $P(s'|a, s)$ and reward function $R(s, a, s')$; solve this as an MDP.
2. Learn $Q^*(s, a)$, use this to guide action.
3. Search through a space of policies (controllers) .

All face the problem of exploration vs exploitation...

Multi-armed Bandits: Exploration vs Exploitation

Exploration vs Exploitation

All the methods discussed so far had some condition like "assuming we visit each state enough", or "taking actions according to some policy"

A 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 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 fixed, unknown average payoff

Which arm has the best average payoff?

How do we maximise the sum of rewards over time?

Consider a row of three poker machines

$R(win) = 1$ for all machines

A

$P(A, win) = 0.6$

B

$P(B, win) = 0.55$

C

$P(C, win) = 0.4$

Expected utility theory tells us that A is the best arm $\rightarrow$ but we don't know that!

Multi-Armed Bandit Problem

- Want to explore all arms
  - BUT, if we explore too much, may sacrifice reward we could have gotten
- Want to exploit promising arms more often
  - BUT, if we exploit too much, can get stuck with sub-optimal values
- Want to minimise regret = loss from playing non-optimal arm
- Need to balance between exploration and exploitation

Exploration Strategies

An exploration strategy is a rule for choosing which arm to play at some time step t, given arm selections and outcomes of previous trials at times $0, 1, \ldots, t-1$ (also called a *policy* in the MAB literature, but we'll reserve that word for MDP/RL state-action policies).

- ϵ-**greedy strategy**: choose random action with probability ϵ and choose a best action with probability $1 - \epsilon$.
- **Softmax** or **Boltzmann** strategy: in state s, choose a with probability $\frac{e^{Q(s,a)/\tau}}{\sum_a e^{Q(s,a)/\tau}}$ where $\tau > 0$ is the *temperature*.
- "Optimism in the face of uncertainty": initialize Q to values that encourage exploration.
- **Upper confidence bounds**: take into account average plus variance information...

Upper confidence bounds

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.

Upper confidence bound exploration strategy: Intuition

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

- Higher estimated reward $\hat{v}_i$ is better (exploit)
- Expect "true value" to be in some confidence interval around v_i
- Confidence interval is large when number of trials n_i is small, shrinks in proportion to $\sqrt{n_i}$
- High uncertainty about move, larger exploration term
- Sample more if number trials is much less than total number of trials

Can use this to sample nodes to expand in "selection" phase of MCTS, leads to UCT algorithm.

Model-based RL

Recall: Asynchronous VI for MDPs, storing $Q(s, a)$

(If we knew the model:)

Initialize the table $Q(S, A)$ 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

A simple approach: just estimate the MDP from 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:

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

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

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

Now solve the MDP $\langle S, A, \hat{P}, \hat{R} \rangle$

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 the actual MDP models, which is not much help if state space is too large

Q-learning

Temporal Differences

- Suppose we have a sequence of values:

$$v_1, v_2, v_3, \ldots$$

and want a running estimate of the average of the first k values:

$$A_k = \frac{v_1 + \cdots + v_k}{k}$$

Temporal Differences (cont)

- Suppose we know A_{k-1} and a new value v_k arrives:

$$\begin{aligned} A_k &= \frac{v_1 + \cdots + v_{k-1} + v_k}{k} \\ &= \frac{k-1}{k} A_{k-1} + \frac{1}{k} v_k \end{aligned}$$

  Let $\alpha_k = \frac{1}{k}$, then

$$\begin{aligned} A_k &= (1 - \alpha_k) A_{k-1} + \alpha_k v_k \\ &= A_{k-1} + \alpha_k (v_k - A_{k-1}) \end{aligned}$$

  "TD formula"

- Often we use this update with α fixed.
- We can guarantee convergence to average if

$$\sum_{k=1}^{\infty} \alpha_k = \infty \text{ and } \sum_{k=1}^{\infty} \alpha_k^2 < \infty \quad \text{e.g. if } \alpha_k = k^{-1} \text{ or } \alpha_k = a(b+k)^{-1}$$

Q-learning

Recall: $Q^*(s,a) = \sum_{s'} P(s' \mid a,s)\,(R(s,a,s') + \gamma \max_a Q^*(s',a))$

Idea: store $Q(state, Action)$; update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).

- Suppose the agent has an experience (s,a,r,s')
- This provides one piece of data to update $Q(s,a)$.
- An experience (s,a,r,s') provides a new estimate for the value of $Q^*(s,a)$:

$$r + \gamma \max_{a'} Q(s',a')$$

which can be used in the TD formula, giving:

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

Q-learning: pseudocode

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

initialize $\hat{Q}^*(s, a)$ arbitrarily

observe current state s

repeat 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

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

return π, Q

Properties of Q-learning

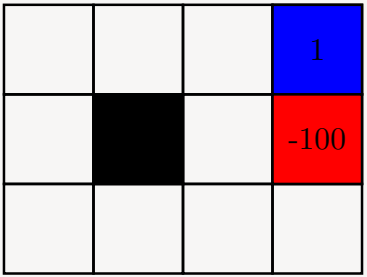
Q-learning converges to an optimal policy, no matter what the agent does, as long as it tries each action in each state enough times.

But what should the agent do? Use an exploration strategy to:

- exploit: when in state s, select an action that maximises $Q(s, a)$
- explore: select another action

How? See MABs earlier.

Q-learning example



Agent can move up, down, left or right. Knows the state space

Doesn't know transitions: Hit a wall and the agent stays where it is; Moves successfully with $p = 0.8$, or perpendicular to direction with $p = 0.1$, each direction.

Doesn't know rewards (on grid).

Problems with Q-learning

- It does one backup between each experience.
  - Is this appropriate for a robot interacting with the real world?
  - An agent might be able to make better use of the data by
    — doing multi-step backups
    — building a model, and using MDP methods to determine optimal policy.
- It learns separately for each state, might be able to learn better over collections of states
  $\rightarrow$ feature-based RL

SARSA (state-action-reward-state-action)

On-policy Learning

- Q-learning does off-policy learning: it learns the value of an optimal policy, no matter what it does.
- This could be bad if the exploration policy is dangerous.
- On-policy learning learns the value of the policy being followed.
  e.g., act greedily 80% of the time and act randomly 20% of the time
- Why? If the agent is actually going to explore, it may be better to optimize the actual policy it is going to do.
- SARSA uses the experience (s, a, r, s', a') to update $Q(s, a)$.

SARSA

initialize $\hat{Q}^{\pi}(s, a)$ arbitrarily

observe current state s

select an action a

repeat until convergence:

carry out an action a
observe reward r and state s'
$\hat{Q}^{\pi}(s, a) \leftarrow \hat{Q}^{\pi}(s, a) + \alpha \left(r + \gamma \hat{Q}^{\pi}(s', a') - \hat{Q}^{\pi}(s, a) \right)$
$s \leftarrow s'$
$a \leftarrow a'$

for each state s

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

Attributions and References

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati for their materials.

Other materials derived from Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 3E, Prentice Hall, 2009.

All remaining errors are Archie's — please email if you find any: archie.chapman@uq.edu.au