Reinforcement Learning
Lecture 10

Gillian Hayes

8th February 2007

School of informatics

Algorithms for Solving RL: Temporal Difference Learning (TD)

- Incremental Monte Carlo Algorithm
- TD Prediction
- TD vs MC vs DP
- TD for control: SARSA and Q-learning

Incremental Monte Carlo Algorithm

Our first-visit MC algorithm had the steps:

R is the return following our first visit to s
Append R to $Returns(s)$
$V(s) = \text{average}(Returns(s))$

We can implement this incrementally:

$$V(s) = V(s) + \frac{1}{n(s)}[R - V(s)]$$

where $n(s)$ is the number of first visits to s

We can also formulate a constant-α Monte Carlo update:

$$V(s) = V(s) + \alpha[R - V(s)]$$

useful when tracking a non-stationary problem (why?).

Model-Based vs Model-Free Learning

- In RL we're generally trying to learn an optimal policy
- If a model is available, $P^a_{ss'}$, $R^a_{ss'}$, we can calculate optimal policy via dynamic programming
- If no model, either:

  learn model and then derive optimal policy (model-based methods) or

  learn optimal policy without learning model (model-free methods)
- Temporal difference (TD) learning is a model-free, bootstrapping method based on sampling the state-action space

Temporal Difference Prediction

Policy Evaluation is often referred to as the Prediction Problem: we are trying to predict how much return we'll get from being in state s and following policy π by learning the state-value function V^π.

Monte-Carlo update:

$$V(s_t) \to V(s_t) + \alpha[R_t - V(s_t)]$$

Target: actual return from s_t to end of episode

Simplest temporal difference update TD(0):

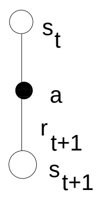
$$V(s_t) \to V(s_t) + \alpha[\underbrace{r_{t+1} + \gamma V(s_{t+1})} - V(s_t)]$$

Target: estimate of the return

Both have the same form

Temporal Difference Learning

- Doesn't need a model $P^a_{ss'}$, $R^a_{ss'}$
- Learns directly from experience
- Updates estimates of $V(s)$ based on what happens after visiting state s



Backup diagram

$$\Delta V_t(s_t) = \alpha[r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)]$$

$V_{t+1}(s_t) - V_t(s_t)$

Step size e.g. 10^{-1}

Actual reward for one step

Discounted estimate of future reward

Initial estimate of future reward

Better estimate than....................................this

TD(0) update:

$$V(s_t) \rightarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

cf Dynamic Programming update:

$$\begin{aligned} V^\pi(s) &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\} \\ &= \sum_a \pi(s,a,) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

Advantages of TD Learning Methods

- Don't need a model of the environment
- On-line and incremental so can be fast

  don't need to wait till the end of the episode so need less memory, computation
- Updates are based on actual experience (r_{t+1})
- Converges to $V^\pi(s)$ – but must decrease step size α as learning continues
- Compare backup diagrams of TD, MC and DP

Bootstrapping, Sampling

TD **bootstraps**: it updates its estimates of V based on other estimates of V

DP also bootstraps

MC does not bootstrap: estimates of complete returns are made at the end of the episode

TD **samples**: its updates are based on one path through the state space

MC also samples

DP does not sample: its updates are based on all actions and all states that can be reached from the updating state

Examples: see e.g. random walk example S+B sect. 6.2

MC vs TD updating: see e.g. S+B sect. 6.3

Difference Between TD and MC Estimates

See S+B Example 6.4:

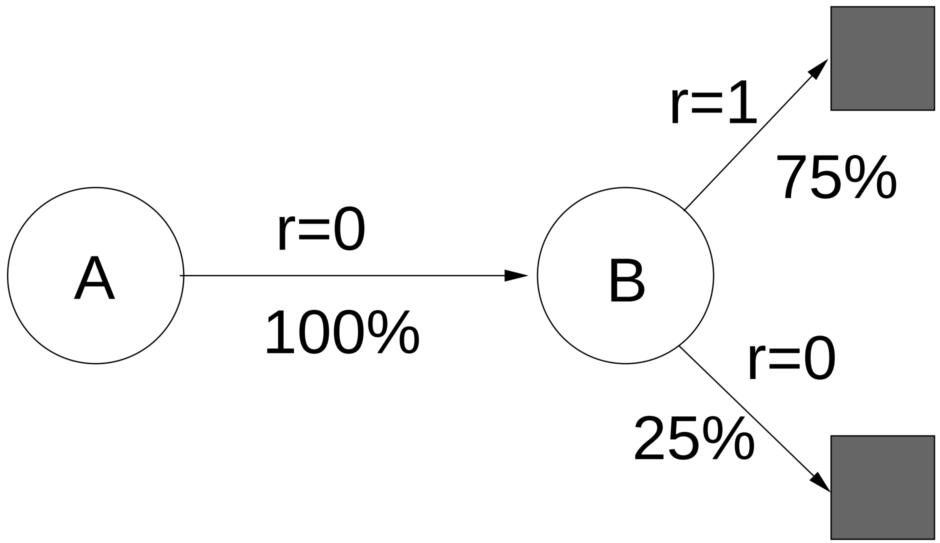
Suppose you observe the following 8 episodes:

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

First episode starts in state A, transitions to B getting a reward of 0, and terminates with a reward of 0. Second episode starts in state B and terminates with a reward of 1, etc.

What are the best values for the estimates V(A) and V(B)?

Modelling the Underlying Markov Process



V(A) = ?

TD and MC Estimates

- Batch Monte Carlo (updating after all these episodes are done) gets V(A) = 0.
  - This best matches the training data
  - It minimises the mean-square error on the training set

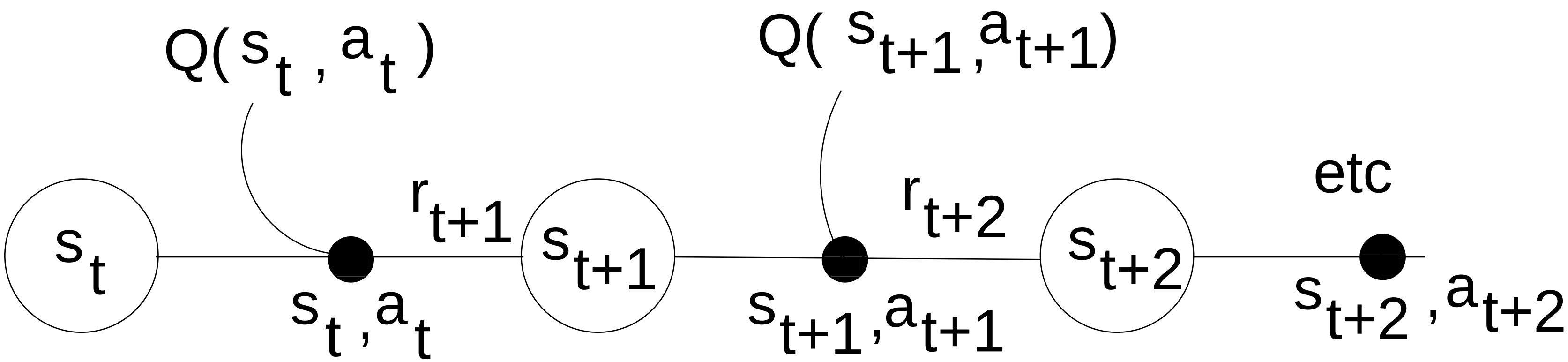
- Consider sequentiality, i.e. A goes to B goes to terminating state; then V(A) = 0.75.
  - This is what TD(0) gets
  - Expect that this will produce better estimate of future data even though MC gives the best estimate on the present data

– Is correct for the maximum-likelihood estimate of the model of the Markov process that generates the data, i.e. the best-fit Markov model based on the observed transitions
– Assume this model is correct; estimate the value function – "certainty-equivalence estimate"

TD(0) tends to converge faster because it's moving towards a "better" estimate.

TD for Control: Learning Q-Values

Learn action values $Q^{\pi}(s,a)$ for the policy π



SARSA update rule:

$\Delta Q_t(s_t, a_t) = \alpha[r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$

• Choose a behaviour policy π and estimate the Q-values (Q^π) using the SARSA update rule. Change π towards greediness wrt Q^π.

• Use ϵ-greedy or ϵ-soft policies.

• Converges with probability 1 to optimal policy and Q-values if visit all state-action pairs infinitely many times and policy converges to greedy policy, e.g. by arranging for ϵ to tend towards 0.

Remember: learning optimal Q-values is useful since it tells us immediately which is(are) the optimal action(s) – have the highest Q-value

SARSA Algorithm

- Initialise $Q(s, a)$
- Repeat `many times`
  - Pick s, a
  - Repeat `each step to goal`
    - Do a, observe r, s'
    - Choose a' based on $Q(s', a')$ ϵ`-greedy`
    - $Q(s, a) = Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    - $s = s'$, $a = a'$
  - Until s terminal (where $Q(s', a') = 0$)

Use with policy iteration, i.e. change policy each time to be greedy wrt current estimate of Q

Example: windy gridworld, S+B sect. 6.4

Q-Learning

SARSA is an example of **on-policy** learning. Why?

Q-LEARNING is an example of **off-policy** learning

Update rule:

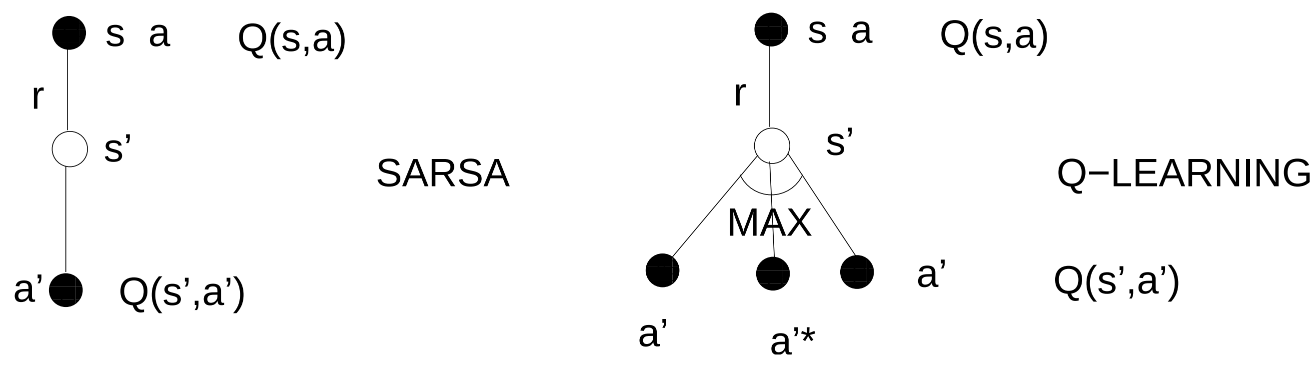
$$\Delta Q_t(s_t, a_t) \quad = \quad \alpha[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)]$$

Always update using *maximum* Q value available from next state: then $Q \Rightarrow Q*$, optimal action-value function

Q-Learning Algorithm

- Initialise $Q(s,a)$
- Repeat `many times`
  - Pick s `start state`
  - Repeat `each step to goal`
    - Choose a based on $Q(s,a)$ ϵ`-greedy`
    - Do a, observe r, s'
    - $Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    - $s = s'$
  - Until s terminal

Backup Diagrams for SARSA and Q-Learning



SARSA backs up using the action a' actually chosen by the behaviour policy.

Q-LEARNING backs up using the Q-value of the action a'^{*} that is the *best* next action, i.e. the one with the highest Q value, $Q(s', a'^{*})$. The action actually chosen by the behaviour policy *and followed* is not necessarily a'^{*}

Example: The cliff S+B sect. 6.5

Q-Learning vs SARSA

QL: $Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ off-policy

SARSA: $Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$ on-policy

In the cliff-walking task:

QL: learns optimal policy along edge

SARSA: learns a safe non-optimal policy away from edge

ϵ-greedy algorithm

For $\epsilon \neq 0$ **SARSA** performs better online. Why?

For $\epsilon \rightarrow 0$ gradually, both converge to optimal.