Reinforcement Learning
Lecture 5: Temporal Difference

Lecturer: Prof. Dr. Mathias Niepert

Institute for Artificial Intelligence
Machine Learning and Simulation Lab

University of Stuttgart
Germany

imprs-is

May 16, 2024

Outline

Admin

Lectures and Tutorials

- There will be no lectures on May 23rd and May 30th due to holidays.
- The final exam is (centrally) scheduled for the 13th of August. Please do not forget to register on time.
- There will be one recap session on July 11th and we will organize a poll on Ilias to prioritize topics
- There will be no lecture on July 18th

Lecture to Book Chapter Mapping

The exam will cover topics in lectures 1-9

1. Multi-armed bandits; chapters: 2.1, 2.2, and 2.4 (+ softmax policy)
2. MDPs (definition, values function, etc.); chapters: 3.1-3.6
3. Policy improvement with dynamic programming; chapters 4.1-4.4
4. Monte-Carlo methods; chapter 5.1-5.7
5. Temporal difference methods; 6.1, 6.2, and 6.4-6.6
6. Planning and Learning; chapters: 8.1-8.5
7. Function approximation; chapter: 9.1-9.4, 9.5.4
8. n-step bootstrapping (no eligibility traces!); chapters: 7.1-7.3
9. Policy gradient methods; chapters: 13.1-13.6

Temporal Difference Prediction

Temporal difference (TD) learning

- Incremental Monte Carlo
- TD prediction
- TD vs MC vs DP
- TD for control:
  - SARSA
  - Q-learning
  - Expected SARSA

Incremental Monte Carlo algorithm

- First–visit MC:
  - G_t is the return following our first visit to $s = S_t$ in time t
  - append G_t to $Returns(s)$
  - $V(s) = average(Returns(s))$
- We can implement this as an incremental update:

$$V(s) \leftarrow V(s) + \frac{1}{n(s)+1}\left[G_t - V(s)\right]$$

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

Incremental Monte Carlo algorithm

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

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

- Why is this useful when tracking a non-stationary problem?

Policy evaluation (prediction)

- We have some policy π which tells the agent which action a to choose in state s
- Find the value function $v_\pi(s)$ of this policy, i.e. **evaluate** the policy π

$$V(S_t) = V(S_t) + \alpha\big[\underbrace{G_t}_{\text{MC target}} - V(S_t)\big]$$

- Simplest temporal difference update TD(0):

$$V(S_t) = V(S_t) + \alpha\big[\underbrace{\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD target}} - V(S_t)}_{\text{TD error}}\big]$$

- TD error is error in the estimate made at a particular time step
- Reinforcement:
  - more reward than expected: $R_{t+1} + \gamma V(S_{t+1}) > V(S_t) \Rightarrow V(S_t) \uparrow$
  - less reward than expected: $R_{t+1} + \gamma V(S_{t+1}) < V(S_t) \Rightarrow V(S_t) \downarrow$

Temporal difference learning

TD combines **sampling** of Monte Carlo with **bootstrapping** (update based in part on existing estimate) from Dynamic Programming:

$$
\begin{aligned}
v_\pi(s) &= \underbrace{\mathbb{E}_\pi\left[G_t \mid S_t = s\right]}_{\text{MC target uses sample return}} \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \underbrace{\mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]}_{\text{DP target uses current estimate } V(S_{t+1})}
\end{aligned}
$$

- TD(0) update:

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

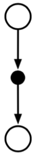
- Dynamic Programming update:

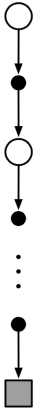
$$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma V(s')\Big] \text{ for all } s \in \mathcal{S}$$

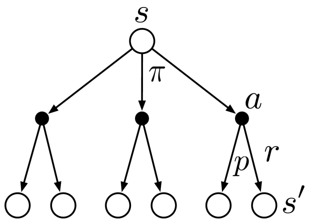
Bootstrapping and sampling

- **Bootstrapping**
  - TD updates its estimates of $v_\pi(s)$ based on other estimates of v_π
  - DP also uses bootstrapping
  - MC does not use bootstrapping; estimates returns at end of episode
- **Sampling**
  - TD updates are based on sampled paths through the state space
  - MC also samples
  - DP does not sample; updates estimates based on all states that can be reached

Backup diagrams





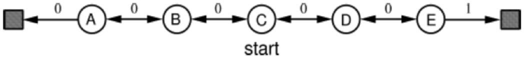


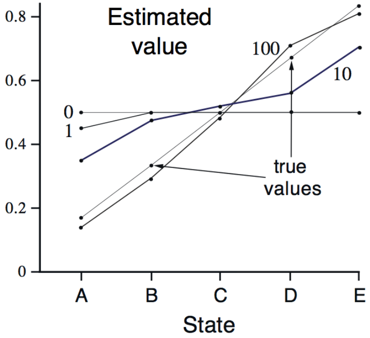
TD prediction versus MC prediction

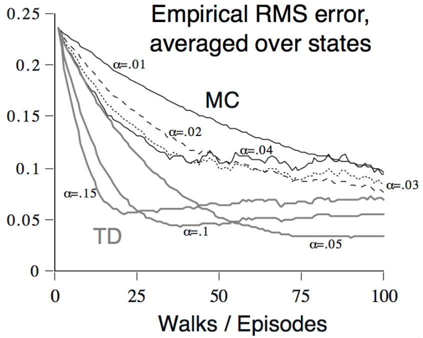
- TD can learn before episode termination
- TD can be used for either non–episodic or episodic tasks
- Update depends on single stochastic transition: *lower* *variance*
- Updates use bootstrapping: estimate has a *bias* - TD updates exploit the Markov property

- MC learning must wait until the episode terminate
- MC only works for episodic tasks
- Update depends on a sequence of many stochastic transitions: *larger* *variance*
- MC is an *unbiased* estimate
- MC updates does not exploit the Markov property; can be effective in non–Markovian environments

Random walk example – Markov reward process







The value of each state is a prediction of the probability of terminating on the right

$$V(\text{A}) = \frac{1}{6}, V(\text{B}) = \frac{2}{6}, V(\text{C}) = \frac{1}{2} \dots$$

Difference between TD and MC estimates

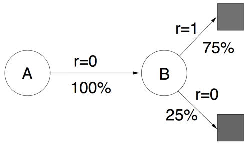
- Suppose we observe the following 8 episodes from a Markov Reward Process:

  1. A, 0, B, 0
  2. B, 1
  3. B, 1
  4. B, 1
  5. B, 1
  6. B, 1
  7. B, 1
  8. B, 0

- First episode starts in state A, transitions to B getting reward of 0, and terminates with a reward of 0
- Second episode starts in state B and terminates with reward of 1
- . . .
- What are the best estimates for the values $V(\text{A})$ and $V(\text{B})$?

Modeling the underlying Markov process

1. A, 0, B, 0
2. B, 1
3. B, 1
4. B, 1
5. B, 1
6. B, 1
7. B, 1
8. B, 0



- Option 1: Monte Carlo
  - B terminates with 1 in 6/8 cases and with 0 in 2/8 cases $\Rightarrow V(\text{B}) = 0.75$
  - Only episode starting from A terminates with 0 $\Rightarrow V(\text{A}) = 0$
- Option 2: modeling the underlying Markov process
  - A transitions to B in 100% of the cases
  - B has value of 0.75, with $\gamma = 1 \Rightarrow V(\text{A}) = 0.75$
  - This is also the outcome when running TD(0) ...

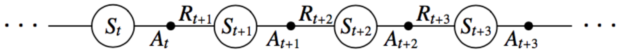
$$V(A) \leftarrow V(A) + \alpha\big[0 + \gamma V(B) - V(A)\big]$$

TD and MC batch estimates

- Batch versions of α-MC and TD(0) compute increments for each episode in the training data, and update with the sum of increments
- Both batch versions are guaranteed to converge to a single value function, but these value function are potentially different
- Batch Monte Carlo results in $V(A) = 0$
  - best match for the training data
  - minimises the mean-square error on the training set
- Batch TD(0) results in $V(A) = 0.75$
  - $V(A) \leftarrow V(A) + \alpha\big[R_{t+1} + V(B) - V(A)\big]$
  - correct maximum–likelihood estimate of the model (Markov reward process)
  - we expect that this will produce better estimates for future data

Temporal Difference Control

TD for control



- Estimate action–value function instead of state–value function
- Estimate $q_\pi(s,a)$ for the current behavior policy π (on–policy)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\big]$$

- Update after every transition from a *non–terminal* state S_t
  - $Q(S_{t+1}, \cdot) = 0$ if S_{t+1} is *terminal*
- The **Sarsa** update rule

On–policy TD control with Sarsa

- Estimate the Q-values (Q_π) using the Sarsa update rule (behaviour policy π)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\big]$$

- Change π towards greediness wrt Q_π
- Use soft policies, e.g. ϵ–greedy
- Converges with probability 1 to optimal policy and action–values if it visits all state–action pairs infinitely many times and policy converges to greedy policy, e.g. by arranging for $\epsilon \rightarrow 0$
- Learning optimal action–values is of course useful for control since it tells us immediately which is (or are) the optimal action(s)

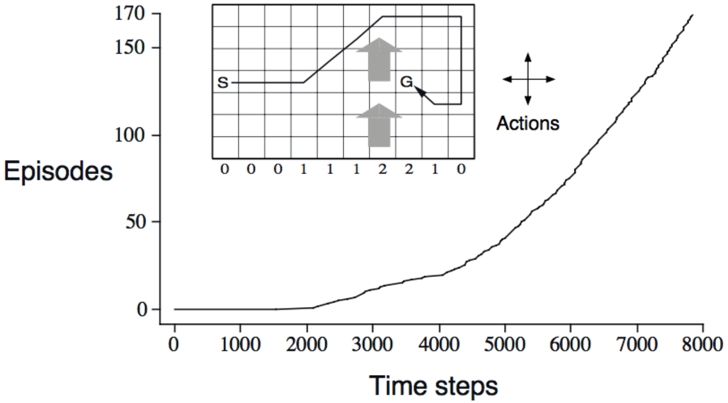
Sarsa (on–policy TD control)

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Windy gridworld example



Upward wind. Constant reward of -1 until reaching G.

Axis of ordinates: episodes = cumulative number of times the goal has been reached

Q–learning

- Sarsa is an example of *on–policy* learning
- Q–learning is an *off–policy* TD control algorithm
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \big]$
- Always update using maximum Q–value available from next state
- **Target policy:** is the greedy policy: $\pi(a) = \arg\max_a Q(s, a)$
- **Behavior policy:** soft policy, e.g. ϵ–greedy, $\epsilon > 0$
- *Off–policy* learning: target policy $\neq$ behavior policy

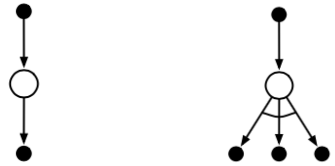
Q–learning (off–policy TD control)

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Backup diagrams for Sarsa and Q-learning



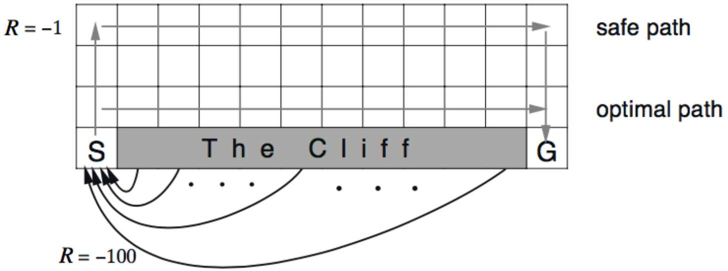
- Sarsa backs up using the actual action chosen by the behavior policy

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

- Q–learning backs up using the best next action (i.e. highest value under current estimates)

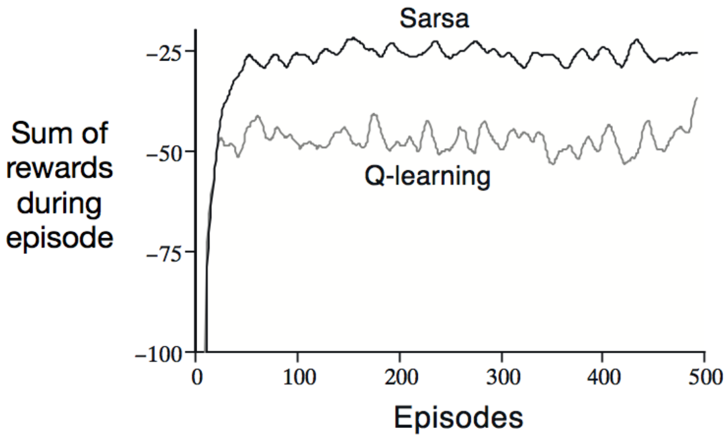
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Cliff walk example



- Deterministic transitions (actions: up, down, left, right)
- ϵ–greedy action selection ($\epsilon = 0.1$)

Cliff walk example



Expected Sarsa

- Just like Q–learning, but *expectation* instead of *maximization*

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\big] \\ &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \end{aligned}$$

- *Deterministically* moves in the same direction as Sarsa moves *in expectation*
- Given same amount of experience it generally performs slightly better than Sarsa
- Can be used off–policy, where the behavior policy differs from π
- What if the behavior policy is exploring and π is greedy wrt to current estimates?

Cliff walk example

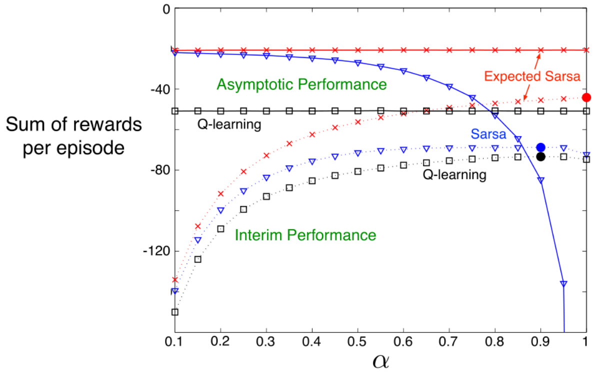
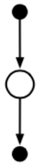
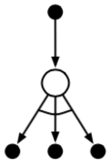
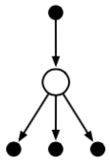


Figure 6.3: Interim and asymptotic performance of TD control methods on the cliff-walking task as a function of α. All algorithms used an ε-greedy policy with $\varepsilon = 0.1$. Asymptotic performance is an average over 100,000 episodes whereas interim performance is an average over the first 100 episodes. These data are averages of over 50,000 and 10 runs for the interim and asymptotic cases respectively. The solid circles mark the best interim performance of each method. Adapted from van Seijen et al. (2009).

Backup diagrams for Sarsa, Q-learning, Expected Sarsa







Extensions

Maximization bias

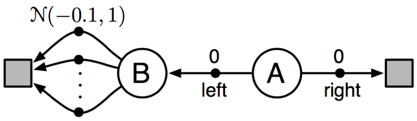
- Many control algorithms involve maximization in the construction of their target policy

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

- Maximum over estimated action–values is used implicitly as an estimate of the maximum value:

$$\max_a \mathbb{E}[Q(s,a)] \neq \mathbb{E}[\max_a Q(s,a)]$$

- This can lead to a significant *positive maximization bias*

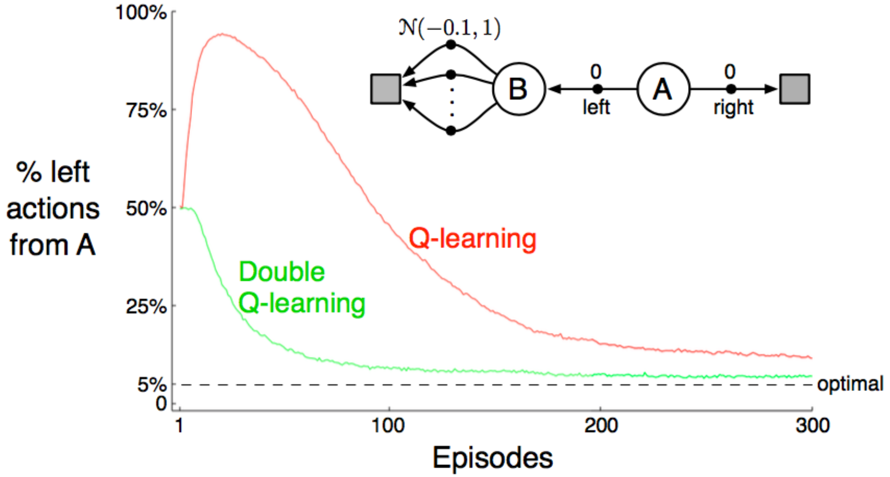


Double Q-learning

- Learn two independent estimates $Q_1(s, a)$ and $Q_2(s, a)$
- Divide time steps in two (flipping a coin), update only one estimate at each time step

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \, Q_2 \, (S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right]$$

- The two approximate value functions are treated completely symmetrical (i.e. switch Q_1 for Q_2 for other side of coin)
- Behavior policy may use both approximations, e.g. ϵ–greedy wrt $Q_1 + Q_2$


$\mathcal{N}(-0.1, 1)$
B
0
left
A
0
right
100%
75%
50%
25%
5%
0
% left actions from A
Q-learning
Double Q-learning
optimal
1
100
200
300
Episodes

Summary

- TD is an alternative to MC for solving the prediction problem
- Extension to control problem via GPI (prediction + local policy improvement)
- Here prediction is based on actual *experience*
  - we need to maintain sufficient exploration
  - *on–* vs. *off–policy* learning
- Sarsa (on–policy), Q–learning (off–policy), Expected Sarsa (both)