# Temporal difference learning

AI & Agents for IET
Lecturer: Liliana Mamani Sanchez

http://www.scss.tcd.ie/~mamanisl/teaching/cs7032/

November 30, 2015

# Recall background & assumptions

- Environment is a finite MDP (i.e. $A$ and $S$ are finite).
- MDP's dynamics defined by transition probabilities:

$$\mathcal{P}^a_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a)$$

- and expected immediate rewards,

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$$

- Goal: to search for good policies $\pi$
- DP Strategy: use value functions to structure search:

$$\begin{aligned} V^*(s) &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^*(s')] \quad \text{or} \\ Q^*(s,a) &= \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')] \end{aligned}$$

- MC strategy: expand each episode and keep averages of returns per state.

# Temporal Difference (TD) Learning

- TD is a combination of ideas from Monte Carlo methods and DP methods:
  - TD can learn directly from raw experience without a model of the environment's dynamics, like MC.
  - TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (i.e. they "bootstrap"), like DP
- Some widely used TD methods: TD(0), TD($\lambda$), Sarsa, Q-learning, Actor-critic, R-Learning

# Prediction & Control

- Recall value function estimation (prediction) for DP and MC:
  - Updates for DP:
  - Update rule for MC:

# Prediction & Control

- Recall value function estimation (prediction) for DP and MC:
  - Updates for DP:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t | s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\} \end{aligned}$$

  - Update rule for MC:

# Prediction & Control

- Recall value function estimation (prediction) for DP and MC:
  - Updates for DP:

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\} \end{aligned}$$

  - Update rule for MC:

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

# Prediction & Control

- Recall value function estimation (prediction) for DP and MC:
  - Updates for DP:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t | s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\} \end{aligned}$$

  - Update rule for MC:

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

(Q: why are the value functions above merely estimates?)

# Prediction in TD(0)

- Prediction (update of estimates of $V$) for the TD(0) method is done as follows:

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

- No need to wait until the end of the episode (as in MC) to update $V$
- No need to sweep across the possible successor states (as in DP).

# Prediction in TD(0)

- Prediction (update of estimates of $V$) for the TD(0) method is done as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{1-step estimate of } R_t} - V(s_t)] \quad (1)$$

- No need to wait until the end of the episode (as in MC) to update $V$
- No need to sweep across the possible successor states (as in DP).

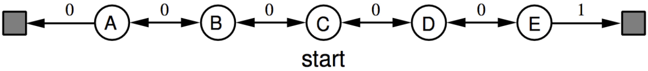
# Tabular TD(0) value function estimation

- Use sample backup (as in MC) rather than full backup (as in DP):

Algorithm 1: TD(0)

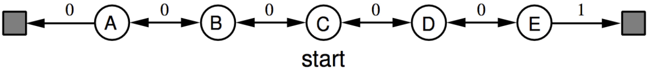
```
TabularTD0(π)
  Initialisation, ∀s ∈ S:
  V(s) ← arbitrary;
  α ← learning constant
  γ ← discount factor

  For each episode E until stop−criterion met
    Initialise s
    For each step of E
       a ← π(s)
       Take action a;
       Observe reward r and next state s′
       V(s) ← V(s) + α[r + γV(s′) − V(s)]
       s ← s′
    until s is a terminal state
```

# Example: random walk estimate

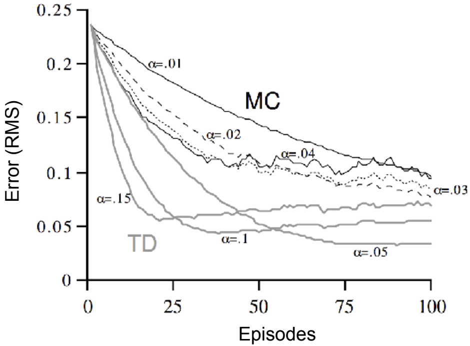



- $\mathcal{P}^a_{ss'} = .5$ for all non-terminal states
- $\mathcal{R}^a_{ss'} = 0$ except for terminal state on the right which is 1.
- Comparison between MC and TD(0) (over 100 episodes)

# Example: random walk estimate




- $\mathcal{P}^a_{ss'} = .5$ for all non-terminal states
- $\mathcal{R}^a_{ss'} = 0$ except for terminal state on the right which is 1.
- Comparison between MC and TD(0) (over 100 episodes)



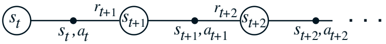
# Focusing on action-value function estimates

- The control (algorithm) part.
- Exploration Vs. exploitation trade-off
- On-policy and off-policy methods
- Several variants, e.g.:
  - Q-learning: off-policy method [Watkins and Dayan, 1992]
  - Modified Q-learning (aka Sarsa): on-policy method

# Sarsa

- Transitions from non-terminal states update $Q$ as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \qquad (2)$$




(for terminal states $s_{t+1}$, assign $Q(s_{t+1}, a_{t+1}) \leftarrow 0$)

**Algorithm 2: Sarsa**

Initialise $Q(s, a)$ arbitrarily
<u>for</u> each episode
  initialise $s$
  choose $a$ from $s$ using $\pi$ derived from $Q$ /* e.g. $\epsilon$-greedy */
  repreat (<u>for</u> each step of episode)
    perform $a$, observe $r, s'$
    choose $a'$ from $s'$ using $\pi$ derived from $Q$ /* e.g. $\epsilon$-greedy */
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  until $s$ is terminal state

# Q-learning

- An off-policy method: approximate $Q^*$ independently of the policy being followed:

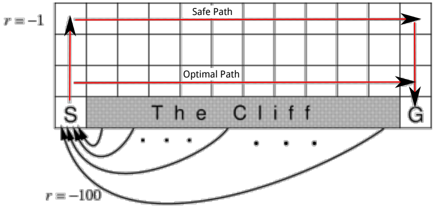
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (3)$$

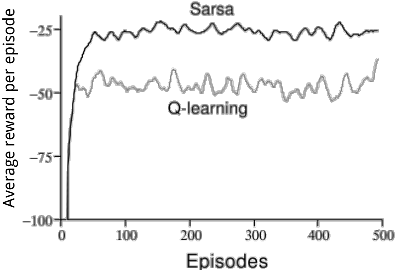
**Algorithm 3: Q-Learning**

Initialise $Q(s, a)$ arbitrarily<br>
**for** each episode<br>
  initialise $s$<br>
  repreat (**for** each step of episode)<br>
    choose $a$ from $s$ using $\pi$ derived from $Q$ /* e.g. $\epsilon-$greedy */<br>
    perform $a$, observe $r, s'$<br>
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$<br>
    $s \leftarrow s'$<br>
  until $s$ is terminal state

# A simple comparison

- A comparison between Q-Learning and SARSA on the "cliff walking" problem [Sutton and Barto, 1998]




- Why does Q-learning find the optimal path? Why is its average reward per episode worse than SARSA's?

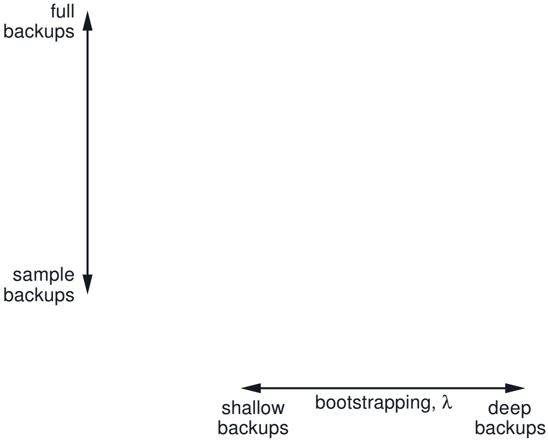
## Convergence theorems

- Convergence, in the following sense:

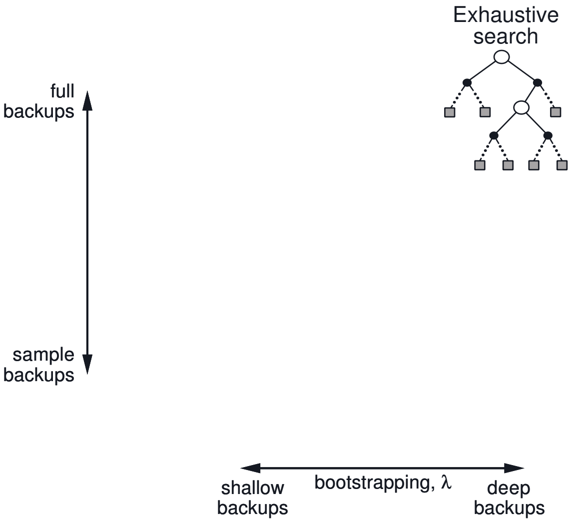
  | |
  |---|
  | $Q$ converges in the limit to the optimal $Q^*$ function, provided the system can be modelled as a deterministc Markov process, $r$ is bounded and $\pi$ visit every action-state pair infinitely often. |

- Has been proved for the above described TD methods: Q-Learning [Watkins and Dayan, 1992], Sarsa and TD(0) [Sutton and Barto, 1998]. General proofs based on stochastic approximation theory can be found in [Bertsekas and Tsitsiklis, 1996].

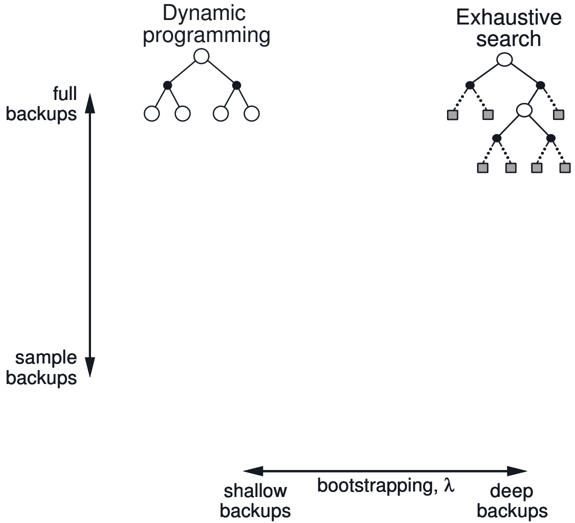
# Summary of methods



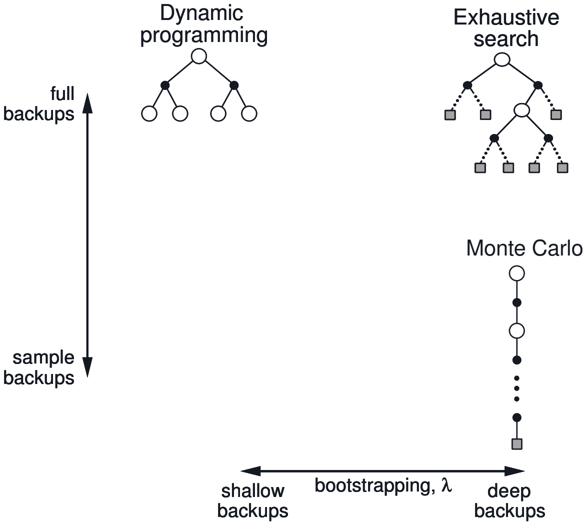
# Summary of methods



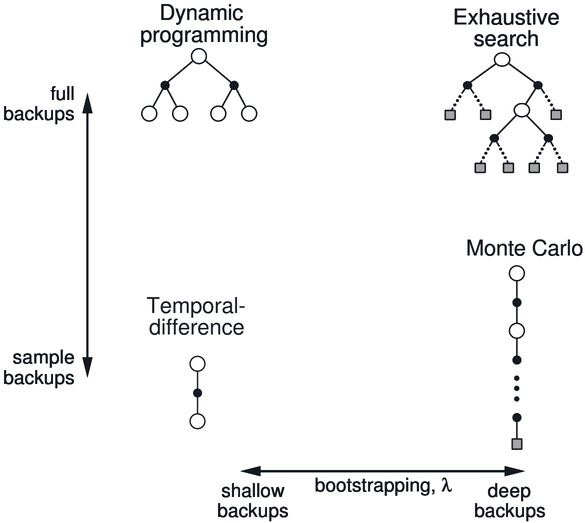
# Summary of methods



# Summary of methods

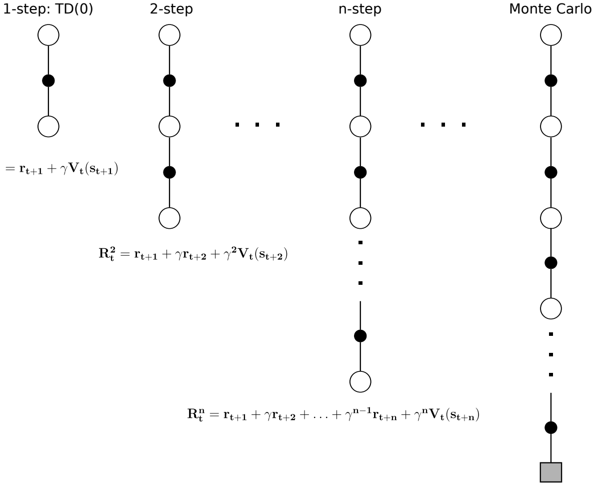


# Summary of methods



# TD($\lambda$)

- Basic Idea: if with TD(0) we backed up our estimates based on one step ahead, why not generalise this to include n-steps ahead?




$$\mathbf{R_t^1 = r_{t+1} + \gamma V_t(s_{t+1})}$$

$$\mathbf{R_t^2 = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})}$$

$$\mathbf{R_t^n = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})}$$

$$\mathbf{R_t = t_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T}$$

# Averaging backups

- Consider, for instance, averaging 2- and 4-step backups:

$$R_t^{\mu} = 0.5R_t^2 + 0.5R_t^4$$

- TD($\lambda$)is a method for averaging all n-step backups.
  - Weight by $\lambda^{n-1}$ $(0 \leq \lambda \leq 1)$:

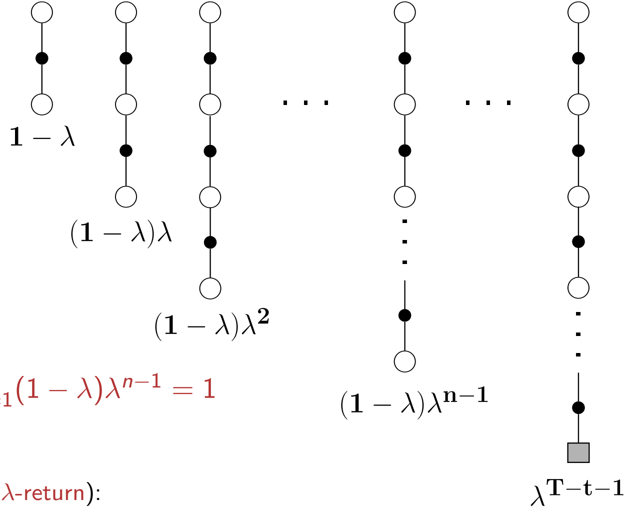
$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^n$$

  - Backup using $\lambda$-return:

$$\Delta V_t(s_t) = \alpha[R_t^{\lambda} - V_t(s_t)]$$

- [Sutton and Barto, 1998] call this the Forward View of TD($\lambda$)

# TD($\lambda$) backup structure




$\lim_{n\to\infty} \sum_{i=1}^{n} (1-\lambda)\lambda^{n-1} = 1$

Backup value ($\lambda$-return):

$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^n$

or

$R_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^n + \lambda^{T-t-1} R_t$

# Implementing TD($\lambda$)

- We need a way to accummulate the effect of the trace-decay parameter $\lambda$
- Eligibility traces:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) + 1 & \text{if} s = s_t \\ \gamma\lambda e_{t-1}(s) & \text{otherwise.} \end{cases} \tag{4}$$

- TD error for state-value prediction is:

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{5}$$

- Sutton and Barto call this the Backward View of TD($\lambda$)

# Equivalence

- It can be shown [Sutton and Barto, 1998] that:

$$\boxed{\text{The Forward View of TD}(\lambda)\text{ and the Backward View of TD}(\lambda)\text{ are equivalent.}}$$

# Tabular TD($\lambda$)

**Algorithm 4:** On-line TD($\lambda$)

```
Initialise V(s) arbitrarily
            e(s) ← 0 for all s ∈ S
for each episode
  initialise s
  repeat (for each step of episode)
    choose a according to π
    perform a, observe r, s′
    δ ← r + γV(s′) − V(s)
    e(s) ← e(s) + 1
    for all s
      V(s) ← V(s) + αδe(s)
      e(s) ← γλe(s)
    s ← s′
  until s is terminal state
```

- Similar algorithms can be implemented for control (Sarsa, Q-learning), using eligibility traces.

# Control algorithms: SARSA($\lambda$)

## Algorithm 5: SARSA($\lambda$)

Initialise $Q(s,a)$ arbitrarily
        $e(s,a) \leftarrow 0$ for all $s \in S$ and $a \in A$
for each episode
  initialise $s, a$
  repeat (for each step of episode)
    perform $a$, observe $r, s'$
    choose $a', s'$ according to $Q$ (e.g. $\epsilon$−greedy
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $e(s, a) \leftarrow e(s, a) + 1$
    for all $s$
      $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$
      $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$  $a \leftarrow a'$
  until $s$ is terminal state

# References

Notes based on [Sutton and Barto, 1998, ch 6].


Bertsekas, D. P. and Tsitsiklis, J. N. (1996).
*Neuro-Dynamic Programming*.
Athena Scientific, Belmont.

Sutton, R. S. and Barto, A. G. (1998).
*Reinforcement Learning: An Introduction*.
MIT Press, Cambridge, MA.

Watkins, C. J. C. H. and Dayan, P. (1992).
Technical note Q-learning.
*Machine Learning*, 8:279–292.