ST449 Artificial Intelligence and Deep Learning

Lecture 8

Temporal difference and eligibility traces

Milan Vojnovic

https://github.com/lse-st449/lectures

Topics of this lecture

- Temporal-difference (TD) learning
- TD prediction
- Sarsa: on-policy TD control
- Q-learning: off-policy TD control

Time-difference (TD) learning

- TD learning: a learning method that combines ideas from Monte Carlo (MC) and Dynamic Programming (DP) methods

- Similarities with MC methods: learning directly from experience without a model of the environment
  - Model of the environment: transition probabilities and conditional expected one-step rewards

- Differences with MC methods: incremental updates of value function estimates
  - MC methods: episode-by-episode incremental updates
  - TD methods: step-by-step incremental updates

MC prediction

- A basic every-visit MC method for non-stationary environments uses the following update rule for estimation of the value function V^{π}:

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

  where R_t is the observed return from time t, and η is a constant step size

- This method is referred to as a constant-η MC method

- We may regard R_t as a target

- The update can be performed only after the return R_t is observed (i.e. after the episode is completed)

TD prediction

- Unlike MC methods, TD methods wait only until next time step
- The simplest TD method (so called $\mathrm{TD}(0)$) is defined by the update:

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

- This update rule has $r_{t+1} + \gamma V(s_{t+1})$ as the target
- Considered as a bootstrap method: update in part based on an existing estimate

MC vs TD update

- Note that $V^{\pi}(s) = \mathbf{E}_{\pi}[R_t \mid s_t = s]$ (M)

$$
\begin{aligned}
V^{\pi}(s) &= \mathbf{E}_{\pi}[R_t \mid s_t = s] && \text{(M)} \\
&= \mathbf{E}_{\pi}[\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s] \\
&= \mathbf{E}_{\pi}[r_{t+1} + \gamma \textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s] \\
&= \mathbf{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s] && \text{(T)}
\end{aligned}
$$

- MC methods use as the target the random variable in (M)
  - Estimate of unknown expected future returns value
- TD methods use as the target the random variable in (T)
  - Estimate for two reasons: expected value in (T) and estimate of the future returns from the next step
- TD methods can be seen as combining the sample of MC and bootstrapping of DP

TD(0) method for estimating a value function

- **Initialization**: V arbitrary, π policy to be evaluated
- **Repeat** for each episode:

  Initialize s

  Repeat for each step of the episode:

    $a \leftarrow$ action given by π for s

    Take action a, observe reward r and next state s'

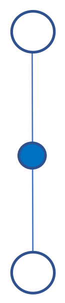
    $V(s) \leftarrow V(s) + \eta[r + \gamma V(s') - V(s)]$

    $s \leftarrow s'$

  until s is a terminal state

Backup diagram

- TD methods are referred to as sample backups
  - They involve looking ahead to a sample successor state (or state-action pair) using the value of the successor and the reward along the way to compute a backed-up value, and then changing the value of the original state
- MC methods are also sample backups
- DP methods are full backups
- The backup diagram of TD(0):



Advantages of TD prediction methods

- Bootstrapping principle: learn a guess from a guess

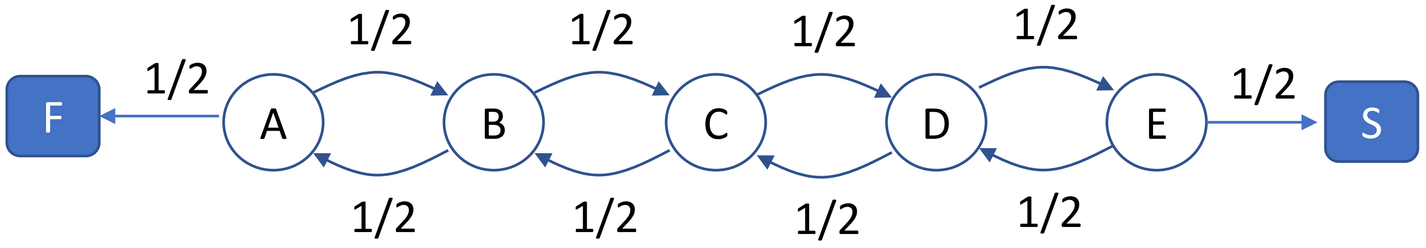
- Advantage over DP methods: no need to know the environment

- Naturally implementable in a fully incremental fashion
  - Step-by-step as opposed to episode-by-episode increments of MC methods

- TD vs MC with respect to convergence speed:
  - No formal convergence speed theorem
  - In practice, TD methods are usually faster than constant-η MC method

Example: random walk prediction problem

- Consider a simple random walk on a path:



- Reward for transition to state S of value 1, zero reward for any other transition

- Exercise: show that $V(A) = \frac{1}{6}, V(B) = \frac{2}{6}, V(C) = \frac{3}{6}, V(D) = \frac{4}{6}, V(E) = \frac{5}{6}$

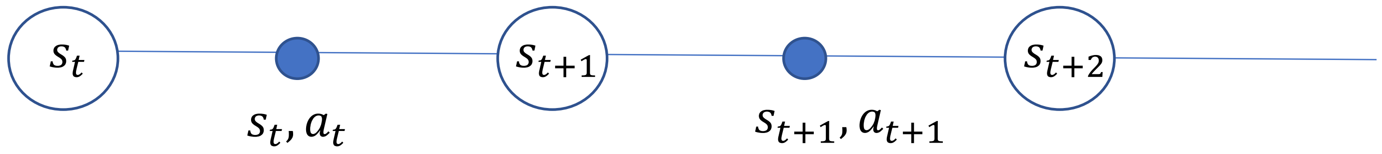
- Exercise: implement MC and TD methods for estimating $V(s)$ for $s \in S$ (seminar)

Sarsa: an on-policy TD control

- Sarsa: a TD prediction methods for the control problem
    - Follows the pattern of generalized policy iteration
    - Uses a TD method for the policy evaluation part

- Key step: estimate action value $Q^{\pi}(s, a)$ for given policy π for all states s and actions a

- An episode interpreted as an alternating sequence of state and state-action pairs:



Sarsa: an on-policy TD control (cont'd)

- The incremental estimation of the action value function:

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

  which is run for any nonterminal state s_t

  If s_{t+1} is a terminal state, then $Q(s_{t+1}, a_{t+1}) = 0$

- The update uses every element of the quintuple of variables:

$$(s_t, \quad a_t, \quad r_{t+1}, \quad s_{t+1}, \quad a_{t+1})$$

S A R S A

Sarsa algorithm

- **Initialization**: Q arbitrarily
- **Repeat** for each episode:

  Initialize s
  Choose a from s using policy derived from Q (e.g. ϵ-greedy)

  Repeat for each step of episode:

  Take action a, observe reward r and next state s'

  Choose a' from s' using policy derived from Q (e.g. ϵ-greedy)

  $$Q(s,a) \leftarrow Q(s,a) + \eta[r + \gamma Q(s',a') - Q(s,a)]$$

  $$s \leftarrow s', a \leftarrow a'$$

  until s is a terminal state

Q-learning: an off-policy TD control

- Basic one-step Q-learning update:

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

- Learning optimal action value function Q^*
  - For whichever behavior policy has been followed
  - The behavior policy has an effect on which state-action pairs are visited

- Q: Why is Q-learning an off-policy control method?

Q-learning algorithm

- **Initialization**: Q arbitrary
- **Repeat** for each episode:

Initialize s

Repeat for each step of episode:

Choose a from s using policy derived from Q (e.g. ϵ-greedy)
Take action a, observe r and s'

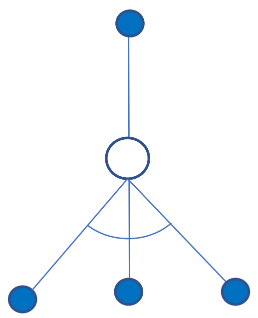
$$Q(s,a) \leftarrow Q(s,a) + \eta \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

$$s \leftarrow s'$$

until s is a terminal state

Q-learning algorithm (cont'd)

- The backup diagram:



- Q-learning has a guaranteed convergence by the condition that all (state, action) pairs continue to be updated

A: why is Q-learning an off-policy learning method?

- Q-learning is an off-policy learning method because it updates Q values using the Q value of the next state s' and action a' selected by using the greedy policy
  - Not using the behavior policy in the action value estimation

- SARSA is an on-policy learning method because it updates Q values using the Q value of next state s' and action a' selected by using the current policy

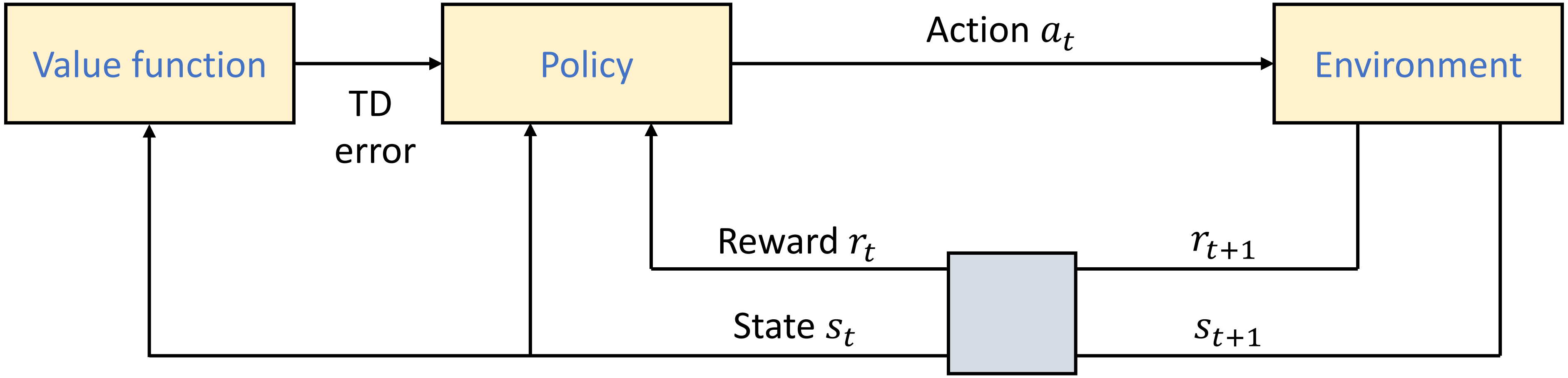
- Q-learning is an on-policy in the special case when the current policy is a greedy policy, which is of no practical interest because the current policy should explore

Actor-critic methods

- Actor-critic methods: TD methods that have a separate memory structure to explicitly represent the policy and value function
  - Actor: policy improvement
  - Critic: value prediction

- Learning is always on-policy
  - The critic must learn about and critique whichever policy is currently being followed by the actor

The actor-critic architecture



Actor-critic example

- Critic maintains value function estimate V and after each action selection, it evaluates the TD error:
$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$
- Actor uses the TD error for selection of future actions
  - If $\delta_t > 0$ increase preference for selecting action a_t
  - If $\delta_t < 0$ decrease preference for selectin action a_t

- For example, actor may use the policy $\pi_t(s,a) = \frac{e^{p(s,a)}}{\sum_{a'} e^{p(s,a')}}$

  where $p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta \delta_t$

  An alternative: $p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta \left(1 - \pi_t(s_t, a_t)\right)\delta_t$

Eligibility traces

- Eligibility traces: a bridge between TD and MC methods

- (Theoretical) forward view: eligibility traces produce a family of methods that span a spectrum of methods with MC at one end and one-step TD at the other

- (Mechanistic) backward view: an eligibility trace is a temporary record of the occurrence of an event, such as visiting of a state or taking an action
  - The trace marks the memory parameters associated with the event as eligible or undergoing learning changes

- Eligibility traces can be seen as a mechanism for temporal credit assignment
  - Similar to TD methods: when a TD error occurs, only the eligible states or actions are assigned credit or blame for the error

n-step TD prediction

- Goal: estimate state value function V^{π} from sample episodes generated by following policy π

- Backup methods:
  - Full backup: perform a backup for each state based on **entire sequence of observed rewards from that state until the end of episode** (ex. MC method)
  - 1-step backup: perform a backup based on **just one next reward**, using the value of the state one step later as a proxy for the remaining rewards (ex. TD(0) method)
  - n-step backup: perform a backup based on **next n rewards** and using the value of the state after n steps as a proxy for the remaining rewards (n-step TD methods)

n-step TD prediction cont'd

- Consider the backup applied to state s_t as a result of a state-reward sequence $s_t, r_{t+1}, s_{t+1}, r_{t+1}, \dots, r_T, s_T$ where T is the last step of the episode

- Backup targets:
  - Full: $R_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-t-1} r_T$
  - 1-step: $R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$
  - 2-step: $R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$

  ⋮
  - n-step: $R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$ (n-step return)

- Truncation: if $n \geq T - t$, then $R_t^{(n)} = R_t^{(T-t)} \equiv R_t$

n-step backup

- The n-step backup is defined by

$$V(s_t) \leftarrow V(s_t) + \Delta V_t(s_t)$$

where

$$\Delta V_t(s_t) = \eta\left(R_t^{(n)} - V_t(s_t)\right)$$

η is a positive step-size parameter, and

$$R_t^{(n)} = \begin{cases} r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}) & \text{if } n < T - t \\ R_t & \text{otherwise} \end{cases}$$

Online vs offline updates

- Online updates: updates are done per each time step during the episode as soon as the increment is computed

$$V_{t+1}(s) = V_t(s) + \Delta V_t(s) \ \text{ for } s \in S$$

- Offline updates: value function increments are accumulated per step during an episode but are used for updating only at the end of the episode

$$V(s) \leftarrow V(s) + \sum_{t=0}^{T-1} \Delta V_t(s) \ \text{ for } s \in S$$

The forward view and complex backups

- Backups can be performed not just towards any n-step return, but towards any average of n-step returns

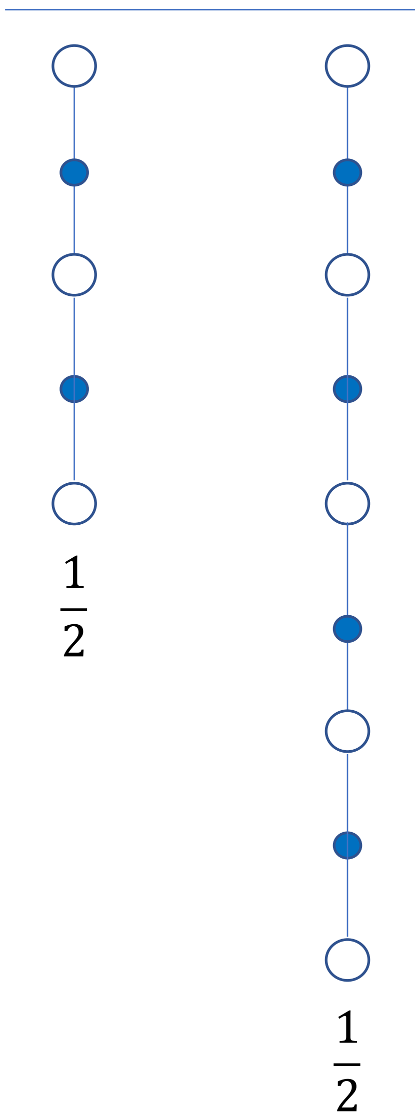
- E.g. average of 2-step and 4-step returns:

$$R_t^{avg} = \frac{1}{2} R_t^{(2)} + \frac{1}{2} R_t^{(4)}$$

- Complex backups: a backup that averages simpler component backups

Backup diagram of a complex backup

- Mixing half of a 2-step backup and half of a 4-step backup



TD(λ) method

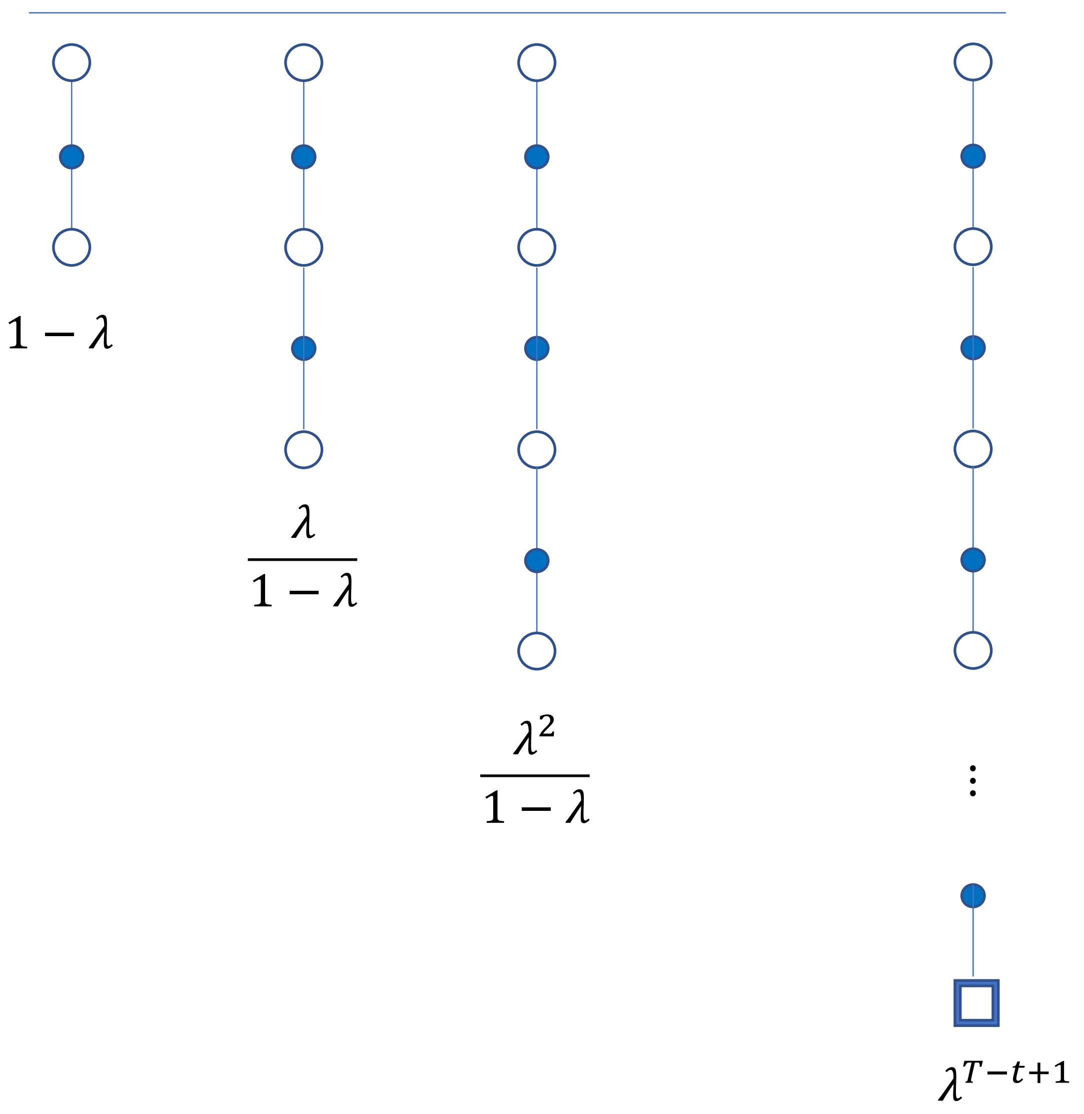
- TD(λ) method: uses an average of n-step backups, each weighted with λ^n where $\lambda \in [0,1]$ is a parameter

- The λ-return:

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

- Special cases:
  - Case $\lambda = 0$: $R_t^\lambda = R_t^{(1)}$ (1-step TD method)
  - Case $\lambda = 1$: $R_t^\lambda = R_t$ (constant-η MC method)

Backup diagram for TD(λ)



λ-return algorithm

- For each time step t:

$$\Delta V_t(s) = \begin{cases} \eta\left[R_t^\lambda - V_t(s_t)\right] & \text{if } s = s_t \\ 0 & \text{otherwise} \end{cases}$$

The backward view of TD(λ)

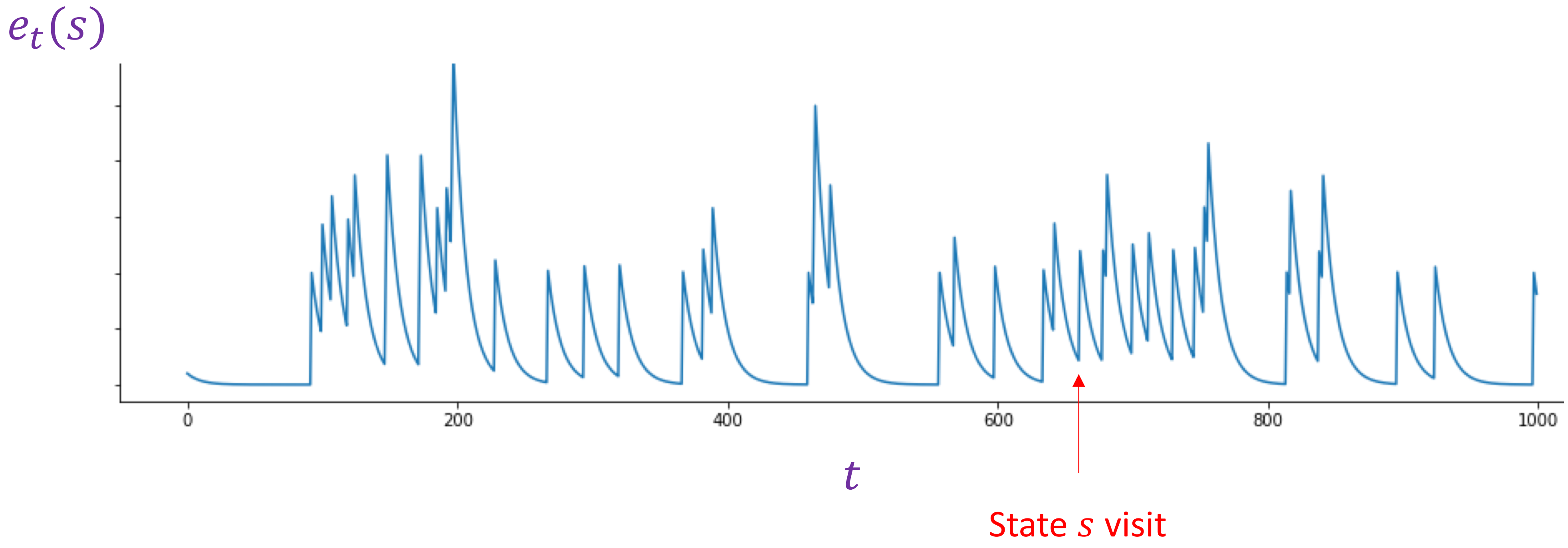
- Backward view: causal incremental method for approximating the forward view

- Eligibility trace: for state $s \in S$, eligibility trace $e_t(s) \in \mathbf{R}^+$ has jumps of size 1 at each state s visit and exponential time discounting:

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{1}_{\{s_t = s\}}$$

- Each eligibility trace indicates the degree to which the corresponding state is eligible for undergoing learning changes

Eligibility trace illustration



On-line TD(λ) algorithm

- Initialization: $V(s)$ arbitrary and $e(s) = 0$ for all $s \in S$
- **Repeat** for each episode:

    Initialize s

    Repeat for each step of the episode:

        $a \leftarrow$ action given by policy π for s

        Take action a, observe reward r and next state s'

        $\delta \leftarrow r + \gamma V(s') - V(s)$

        $e(s) \leftarrow e(s) + 1$

        For each s'':

            $V(s'') \leftarrow V(s'') + \eta e(s'')\delta$

            $e(s'') \leftarrow \gamma\lambda e(s'')$

        $s \leftarrow s'$

    until s is terminal

Equivalence of forward and backward views

- Forward view update: $\Delta V_t^{\lambda}(s_t) = \eta\left(R_t^{\lambda} - V_t(s_t)\right)$

- Backward view update: $\Delta V_t^{TD}(s) = \eta e_t(s)\delta_t$

- The equivalence property: for offline updating it holds

$$\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) = \sum_{t=0}^{T-1} \Delta V_t^{\lambda}(s_t)\mathbf{1}_{\{s_t=s\}} \text{ for all } s \in S$$

- For online updating, the equality holds approximately for small step size η (small changes of the value function within an episode)

Proof of the equivalence property

- Basic fact:

$$e_t(s) = \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathbf{1}_{\{s_k=s\}}$$

- $\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) = \sum_{t=0}^{T-1} \eta \delta_t \, e_t(s)$

$$= \sum_{t=0}^{T-1} \eta \delta_t \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathbf{1}_{\{s_k=s\}}$$

$$= \sum_{k=0}^{T-1} \eta \sum_{t=0}^{k} (\gamma\lambda)^{k-t} \delta_k \mathbf{1}_{\{s_t=s\}}$$

$$= \sum_{t=0}^{T-1} \eta \mathbf{1}_{\{s_t=s\}} \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \delta_k$$

Proof of the equivalence property (cont'd)

- Basic facts: (a) $\frac{1}{\eta}\Delta V_t^{\lambda}(s_t) = R_t^{\lambda} - V_t(s_t)$

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

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

$$= (1-\lambda)\sum_{n=0}^{\infty} \lambda^{n}\left[\sum_{k=0}^{n} \gamma^k r_{t+k+1} + \gamma^{n+1} V_t(s_{t+n+1})\right] - V_t(s_t)$$

$$= \sum_{n=0}^{\infty}(\gamma\lambda)^n r_{t+n+1} + \sum_{n=0}^{\infty}\left[(\gamma\lambda)^n \gamma V_t(s_{t+n+1}) - (\gamma\lambda)^{n+1} V_t(s_{t+n+1})\right] - V_t(s_t)$$

$$= \sum_{n=0}^{\infty}(\gamma\lambda)^n r_{t+n+1} + \sum_{n=0}^{\infty}(\gamma\lambda)^n\left[\gamma V_t(s_{t+n+1}) - V_t(s_{t+n})\right]$$

$$= \sum_{n=0}^{\infty}(\gamma\lambda)^n\left[r_{t+n+1} + \gamma V_t(s_{t+n+1}) - V_t(s_{t+n})\right]$$

$$= \sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k = \sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\delta_k$$

Proof of the equivalence property (cont'd)

- From the last slide, it follows

$$\sum_{t=0}^{T-1} \Delta V_t^{\lambda}(s_t)\mathbf{1}_{\{s_t=s\}} = \sum_{t=0}^{T-1} \eta\left(\sum_{k=t}^{T-1} (\gamma\lambda)^{k-t}\delta_k\right)\mathbf{1}_{\{s_t=s\}}$$

$$= \sum_{t=0}^{T-1} \eta \mathbf{1}_{\{s_t=s\}} \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t}\delta_k$$

- The proof follows because we have shown that $\sum_{t=0}^{T-1} \Delta V_t^{TD}(s)$ is also equal to the right-hand side in the above equation

Sarsa(λ)

- Eligibility traces are used not just for prediction, but also for control
- A popular approach is to learn the action values $Q_t(s,a)$
- Basic idea: apply the TD(λ) prediction method to state-action pairs

$$Q_{t+1}(s,a) = Q_t(s,a) + \eta e_t(s,a)\delta_t \text{ for all } s,a$$

where $\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$

and $e_t(s,a) = \gamma\lambda e_{t-1}(s,a) + \mathbf{1}_{\{s=s_t\}\cap\{a_t=a\}}$

Sarsa(λ) algorithm

- Initialization: $Q(s,a)$ arbitrary and $e(s,a) = 0$ for all s, a

- **Repeat** for each episode:

  Initialize s, a

  Repeat for each step of episode:

  Take action a, observe r, s'

  Choose a' from s' using policy derived from Q (ex ϵ-greedy)

  $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$

  $e(s,a) \leftarrow e(s,a) + 1$

  For each s'', a'':

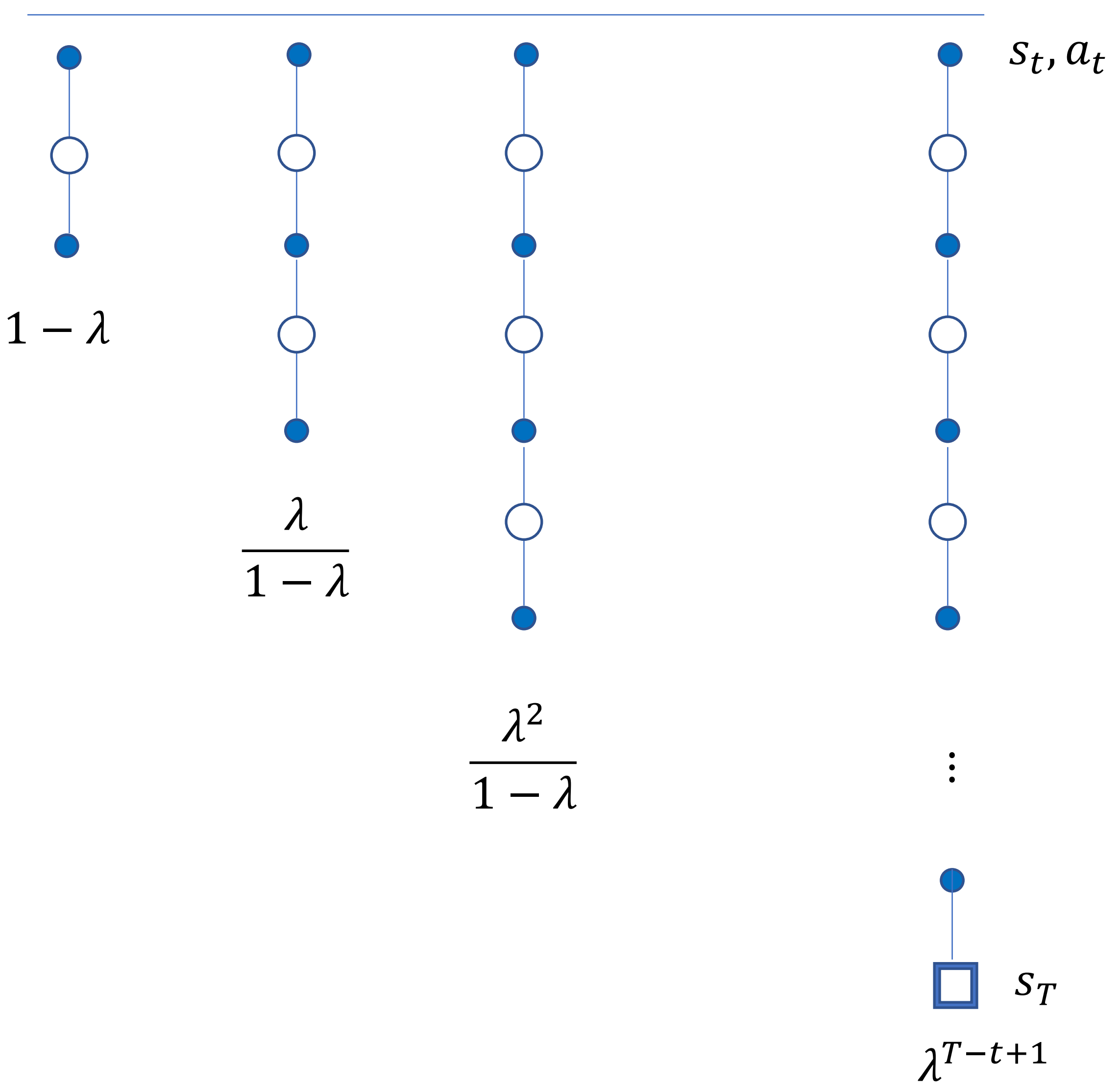
  $Q(s'',a'') \leftarrow Q(s'',a'') + \eta e(s'',a'')\delta$

  $e(s'',a'') \leftarrow \gamma\lambda e(s'',a'')$

  $s \leftarrow s'$ and $a \leftarrow a'$

  until s is terminal

Backup diagram for Sarsa(λ)



References

- R. S. Sutton and A. G. Barto, Reinforcement Learning, Chapters 6 and 7, 1998

Seminar exercises

- TD(0) for random walk problem
- Sarsa: The Windy GridWorld
- Q-learning: The Cliff Walking problem

Extras

MC vs TD(0) exercise: you are the predictor

- Observed (state, reward) outcomes for eight episodes:

  1. $(A, 0), (B, 0)$
  2. $(B, 1)$

  …

  7. $(B, 1)$
  8. $(B, 0)$

- Q1: What are estimates of $V(A)$ and $V(B)$ according to the MC method?
- Q2: What are estimates of $V(A)$ and $V(B)$ according to the TD(0) method?

- Discuss your answers

The error reduction property

- For any value function V:

$$\max_s \left|\mathbf{E}_\pi\left[R_t^{(n)} \mid s_t = s\right] - V^\pi(s)\right| \le \gamma^n \max_s |V(s) - V^\pi(s)|$$

- By using the error reduction property, it can be shown that online and offline TD prediction methods using n-step backups converge to the correct predictions under certain technical conditions

Example: random walk revisited

- Suppose in the first episode the sequence of states is C, D, E
- Assume $V_0(s) = 1/2$ for all $s \in S$

- 1-step method: changes the value estimate only for the last state, $V(E)$, which is incremented towards 1

- 2-step method: changes the value estimates only for the last two states, $V(D), V(E)$, both incremented towards 1

- Three (or more)-n step methods: change the value estimates for all states, $V(C), V(D), V(E)$, incrementing them all towards 1

- Q: which method is better?

Q(λ) learning method

- Q-learning: an off-policy method
  - Learning about the greedy policy while following another policy (typically a policy involving exploratory actions)
  - This requires a care when introducing eligibility traces
- Example:
  - Consider backing up the state-action pair s_t, a_t at time t
  - Suppose in the next two time steps the agent selects greedy actions, but in the on the third it selects an exploratory nongreedy action
  - For learning the value of the greedy policy we can only use the two-step returns
- In general, subsequent experience can be used as long as the greedy policy is being followed

Two Q(λ) learning approaches

- Watkings's Q(λ): only looks ahead as far as the next exploratory action
  - Other principles are much alike to TD(λ) and Sarsa(λ)
  - If a_{t+n} is the first exploratory action, then the longest backup is toward:

$$r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max_a Q_t(s_{t+n}, a)$$

- Peng's Q(λ): avoids cutting off traces every time an exploratory action is taken
  - Cutting off traces every time an exploratory action is taken loses advantage of using eligibility traces
  - The implementation is more complex

Watkins's Q(λ) algorithm

- Initialization: $Q(s,a)$ arbitrary and $e(s,a) = 0$ for all s, a

- **Repeat** for each episode:

  Take action a, observe r, s'
  Choose a' from s' using policy derived from Q (ex ϵ-greedy)
  $a^* \leftarrow \text{argmax}_a\, Q'(s', b)$ (if a' is in argmax then take $a^* \leftarrow a'$)
  $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s,a)$
  $e(s,a) \leftarrow e(s,a) + 1$
  For each s'', a'':
  $Q(s'', a'') \leftarrow Q(s'', a'') + \eta\delta e(s'', a'')$
  If $a' = a^*$ then $e(s'', a'') \leftarrow \gamma\lambda e(s'', a'')$
  else $e(s'', a'') \leftarrow 0$
  $s \leftarrow s'$ and $a \leftarrow a'$

  until s is terminal

Replacing traces

- A capping of eligibility trace: a replacing trace for state s defined by

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases}$$

- Different from the standard definition according to which an eligibility trace can assume a value larger than 1

- It can produce a significant improvement in learning rate