Eligibility Traces (ETs)
Week #7

Introduction

- A basic mechanism of RL.
- λ in TD(λ) refers to an eligibility trace.
- TD methods such as Sarsa and Q-learning may be combined with ETs to obtain more efficient learning methods.
- Two different ways to view ETs:
  - *Forward view*
  - *Backward view*

Forward View of ET

- A more *theoretical* view: ETs are a bridge between TD and MC methods.

- TD methods extended with ETs form a family with TD methods at one end and MC methods at the other.

ET: Definition

***Definition**:* An *eligibility trace* is *a record to keep track of the extent to which a variable in an adaptive system deserves to be updated by the occurrence of a reinforcing event*.

***Example**:* The event of *how recently* an action in a state is selected determines the *extent of the eligibility* of the value of the action in the relevant state to be updated.

n-Step TD Prediction

- Value updates are based upon
  - the entire sequence of observed rewards, at one end (MC methods) and,
  - the next reward at the other end (TD methods).
- Update rules using an *intermediate number of observed rewards* are called the *n-step TD methods*.
- In this context, we call the latter TD methods using the next reward *one-step TD methods*.

Updates... from one end to the other

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T ; \quad MC \quad methods$$

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}); \quad one-step \quad TD \quad methods$$

$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2}); \quad two-step \quad TD \quad methods$$

$$\vdots$$

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}); \quad n-step \quad TD \quad methods$$

where R_t is the *complete return*, T is the *last time step* of the episode and $R_t^{(n)}$ is called the *corrected n-step truncated return*.

On-line and off-line updating

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

- The value estimates are updated with the above increment either immediately
  - $V_{t+1}(s)=V_t(s)+\Delta V_t(s)$, (*on-line updating*),

  or after the entire episode is over
  - $V(s) = V(s) + \sum_{t=0}^{T-1} \Delta V_t(s)$ (*off-line updating*)

Forward View of TD(λ)

- Increments may also be established by *any weighted combination* of the *i-step* returns where the weights add up to 1.
- Example:

$$R_t^{ave} = \left\{ \frac{1}{4} R_t^{(3)} + \frac{1}{4} R_t^{(5)} + \frac{1}{2} R_t^{(7)} \right\}$$

Forward View... cont'd (2)

- Another example: λ-return

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

- with $0 \leq \lambda \leq 1$.
- The *i-step* return is given the i^{th} largest weight

$$(1-\lambda)\ \lambda^{i-1}.$$

Forward View... cont'd (3)

- The λ-return algorithm ...

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

- ... can also be expressed in the following way:

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

Forward View... cont'd (4)

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

- λ-*return* as expressed here shows better that
  - the λ-*return* is a bridge between TD and MC methods, at two opposite ends, by the following two facts
    - with $\lambda = 0$, λ-*return* turns to TD methods ($0^0 = 1$)
    - with $\lambda = 1$, λ-*return* turns to MC methods

Forward View ... Final word

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

- Rather a *theoretical* view of the TD(λ).
- *An acausal* view (i.e., return estimate R_t^{λ} at time t is defined in terms of one or more future rewards r_{t+x}, $x \in N^+$); therefore hard to implement
- The *backward* view more *handy* for implementation

Backward View of TD(λ)

- A more *mechanistic rather than theoretical* view.
- *Causal*
- Associates each state (or action at a state) with a variable, the *eligibility trace*, that specifies how eligible the corresponding state (or action at a state) is for updates by the current reward.

Backward View of TD(λ)

- The eligibility trace, denoted by $e_t(s)$, of a state s is mathematically expressed as follows:

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

- On each step, the eligibility trace, $e_t(s)$, decays by $\gamma\lambda$, and, if visited, its value is refreshed by incrementing by 1 where γ is the discount rate and λ as introduced before is called the *trace-decay parameter*.

Backward View of TD(λ)

- This kind of ET is called an *accumulating trace* to indicate that, at every visit of a state, the state's eligibility trace accumulates then fades away with any time step it is not visited.
- The reinforcing events are the moment-by-moment TD errors and may be mathematically expressed as follows: $\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$
- Then the corresponding update becomes

$$\Delta V(s_t) = \alpha \delta_t e(s_t)$$

- Based on these update increments performed at each time step, the TD(λ) algorithm is given on next page:

Algorithm: TD(λ)

- *Initialize V(s) arbitrarily for all* $s \in S$
- *Repeat for each episode*
  - *Initialize e(s)=0 for all* $s \in S$
  - *Initialize s;*
  - *Repeat for each step of episode*
    - $a \leftarrow \pi(s)$
    - *Perform a; observe r, and next state s'*
    - $\delta \leftarrow r+\gamma V(s')-V(s)$
    - $e(s)=e(s)+1$
    - *For all s:*
      - $V(s) \leftarrow V(s)+\alpha\delta e(s)$
      - $e(s) \leftarrow \gamma\lambda e(s)$
    - $s \leftarrow s'$
  - *Until s is terminal*

Remarks

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & s_t \neq s \\ \gamma\lambda e_{t-1}(s) + 1 & s_t = s \end{cases} \qquad \delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

- Backward view of TD(λ) is causal meaning that the state/action values are a function of past (i.e., not future) state/action values.
- The past values are updated at each time step based on the current TD error depending on the state's ET.
- ***Special cases***:
  - $\lambda = 0$: $e_t(s) = 0$ except for $s = s_t$ $\rightarrow$ *TD(0) method*
  - $\lambda = 1$: λ does not decay credit given to earlier states; hence each state receives credit based upon when it is visited. $\rightarrow$ *MC method or TD(1) method.*

Sarsa(λ)

- ETs can be used to control an environment.
- As usual, we need simply learn action values $Q_t(s,a)$ rather than state values $V_t(s)$. As the relation of *TD(λ)* to *TD(0)*, the version of the Sarsa algorithm with ETs is called *Sarsa(λ)*, and the original version, from now on, *one-step Sarsa*.
- The symbol $e_t(s,a)$ denotes the eligibility trace for action a at state s, and $\delta_t = r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)$ represents the update in action values.

Sarsa(λ)... Formulae

- Further, the following formulae represent the action updates and eligibility traces in Sarsa, respectively:

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

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if} \quad s = s_t \quad \text{and} \quad a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases}$$

Sarsa(λ)... Algorithm

- *Initialize Q(s,a) arbitrarily for all s,a*
- *Repeat for each episode*
  - *e(s,a)=0 for all (s,a) pairs*
  - *Initialize s,a;*
  - *Repeat for each step of episode*
    - *Perform a; observe r, and next state s'*
    - *Choose a' from s' using policy derived from Q(e.g., ε-greedy)*
    - $\delta \leftarrow r+\gamma Q(s',a')-Q(s,a)$
    - $e(s,a)=e(s,a)+1$
    - *For all s,a:*
      - $Q(s,a) \leftarrow Q(s,a)+\alpha\delta e(s,a)$
      - $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    - $s \leftarrow s'; a \leftarrow a'$
  - *Until s is terminal*

Q(λ)

- Q(λ) is the off-policy RL method with ETs
- Two versions of *Q(λ)*
  - *Watkin's Q(λ)*
  - *Peng's Q(λ)*
- The essence of using ETs is; in order to end up with the optimal policy, to increment the values of states or state-action pairs at each time step on the basis of the extent to which it has received visits.
- Looking at *Sarsa(λ)* algorithm, we observe that at each time step the $\gamma\lambda$-*decayed* ET *e(s)* or *e(s,a)* weighting the error δ_t that adjusts the value of the state or state-action pair is or is not incremented depending upon whether or not the state or state-action pair is taken using the policy.

Q(λ) ... (2)

- In the off-policy case, there is no problem as long as the state-action pair selections in the estimation and behavior policy are the same.
- The problem starts at the point where the behavior policy branches away from the estimation policy. The first of the possible exploratory (i.e., non-greedy) actions in the behavior policy interrupts the sequence of action-response loop in the estimation policy and does not provide any correct subsequential experience on the estimation policy any more.
- So, it is no longer usable after the first exploratory action to follow the behavior policy.

Watkin's Q(λ) ...FW view

- Watkin's Q(λ) is just the same as TD(λ) with the only difference that the learning stops at whichever of the first exploratory (i.e., non-greedy) action or the end of the episode occurs first.
- To be exact, if a_{t+n} is the first exploratory action the longest backup is toward

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

- where *off-line updating* is assumed.

Watkin's Q(λ) ...BW view

- From a mechanistic viewpoint, Watkin's Q(λ) exploits ETs just the same as Sarsa(λ) with the only difference that the ETs are set to 0 whenever an exploratory (i.e., non-greedy) action is taken.
- Formally, the *trace update* is expressed as follows:

$$e_t(s,a) = I_{ss_t} * I_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if} \quad Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a) \\ 0 & \text{otherwise} \end{cases}$$

$$\text{where} \quad I_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

$$\text{and} \quad Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a)$$

$$\text{where} \quad \delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)$$

Watkin's Q(λ)... Algorithm

- *Initialize Q(s,a) arbitrarily for all s,a*
- *Repeat for each episode*
  - *e(s,a)=0 for all (s,a) pairs*
  - *Initialize s,a;*
  - *Repeat for each step of episode*
    - *Perform a; observe r and next state s'*
    - *Choose a' from s' using policy derived from Q(e.g., ε-greedy)*
    - $a^* \leftarrow argmax_b Q(s',b)$ *(if a' ties for the max, then* $a^*=a'$*)*
    - $\delta \leftarrow r+\gamma Q(s',a^*)-Q(s,a)$
    - $e(s,a)=e(s,a)+1$
    - *For all s,a:*
      - $Q(s,a) \leftarrow Q(s,a)+\alpha\delta e(s,a)$
      - *if* $a^*=a'$ *then* $e(s,a) \leftarrow \gamma\lambda e(s,a)$
      - *else* $e(s,a) \leftarrow 0$
    - $s \leftarrow s'; a \leftarrow a'$
  - *Until s is terminal*

Peng's Q(λ) ... Motivation

- Watkin's *Q(λ)* is *not sufficiently effective* if exploratory actions are taken frequently (i.e., ε *high*) since a sufficiently long sequence of experience or backups will not form, hence, learning may be only little faster than learning with one-step Q learning.
- Peng's *Q(λ)* is meant to handle this problem.
- It is a mixture of *Sarsa(λ)* and *Q(λ)*.
- The key is that there is *no distinction between the behavior and estimation policy up until the last action taken at which a greedy selection is preferred*.
- It should converge to an intermediate policy between Q^{π} and Q^{*}. The more greedy the policy is made gradually the higher the probability is made for the policy to converge to Q^{*}.

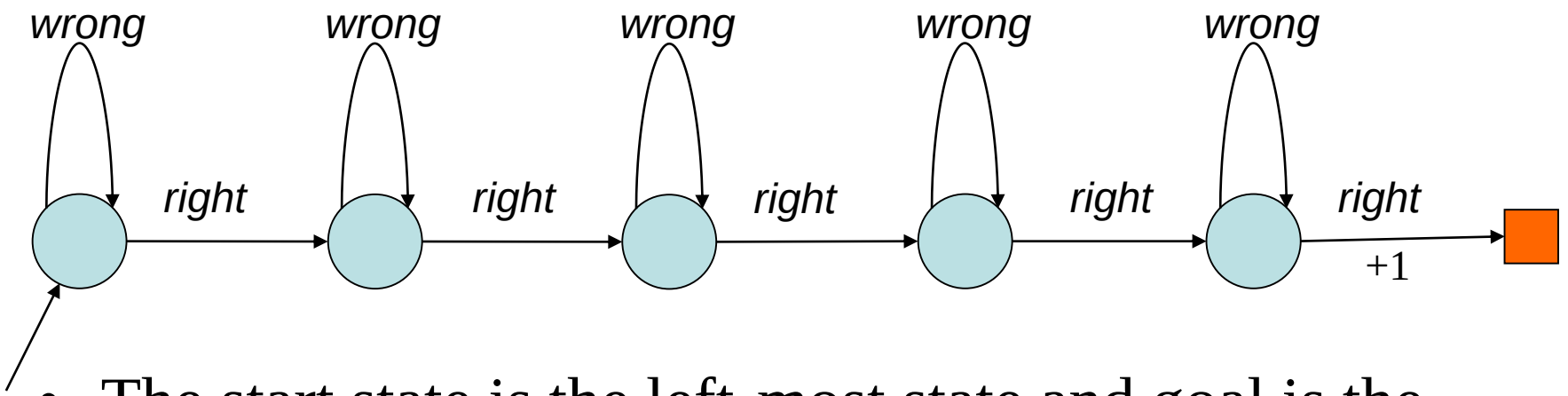
Replacing Traces

- Sometimes a better performance may be obtained using the so-called *replacing trace* with the following trace updates:

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

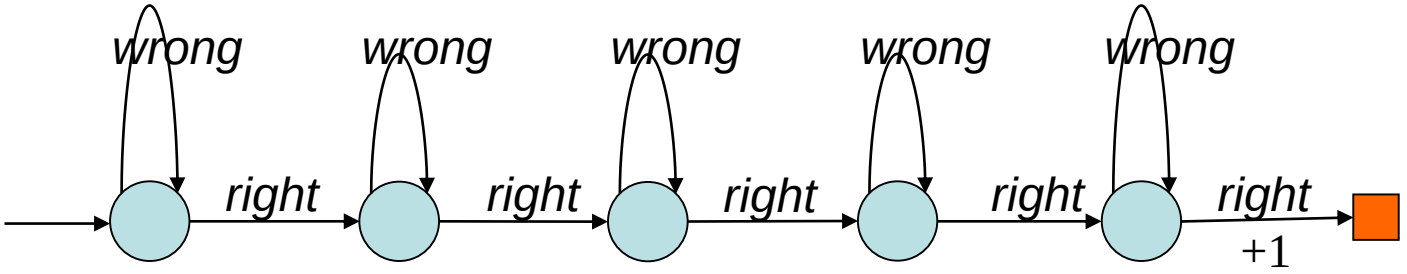
- With replacing traces the trace will never exceed 1 as opposed to accumulating traces which outperform accumulating traces in cases in which there is a good probability of taking a wrong action several consecutive times (see example 7.5 pp 186 & Fig. 7.18).

Example



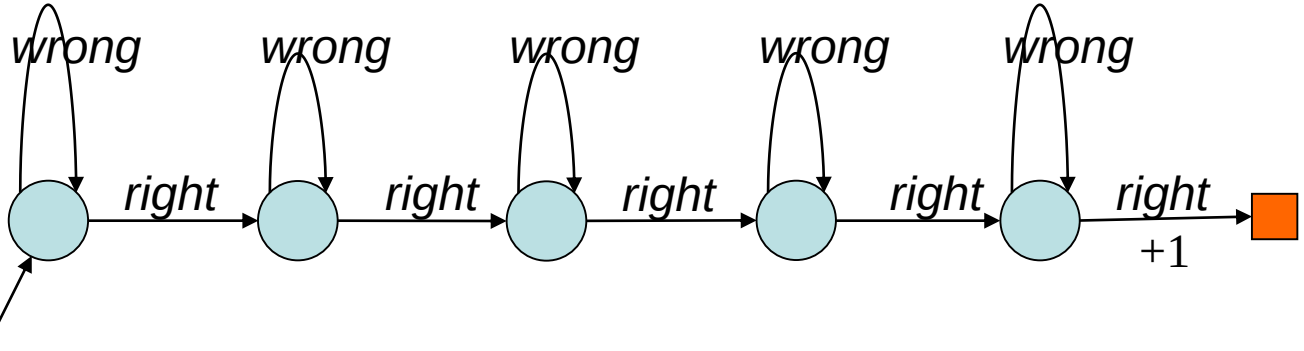
- The start state is the left-most state and goal is the orange square.
- Rewards are zero except that for the action that accesses the goal that provides a +1.
- Imagine what happens when *wrong* is taken by the agent several times before *right*.

Example ...(2)



- With *accumulating* traces:
  - At the end of the first episode, *e(s,wrong)>e(s,right)*, although *right* is more recent, *wrong* is selected more frequently.
  - At the receipt of the reward, this is likely to cause *Q(s,wrong)>Q(s,right)*.
  - This will not continue endlessly. Eventually as *right* is selected more frequently, the convergence will occur on *right*; but learning will slow down.

Example ...(3)



- With *replacing* traces:
  - This will not happen since the trace's value will not accumulate but be replaced (i.e., the value of the trace will not be incremented by 1, but its highest value will be 1 whenever its relevant state is visited.)
  - Hence, a recent *right* will have higher value than a *wrong* with several less recent visits.

Control Methods with Replacing Traces

- Control methods may use replacing ETs.
- Here, the ETs should be modified to involve action selections and distinguish between the action taken and those that are not.
- The following reflects the necessary modification:

$$e_t(s,a) = \begin{cases} 1 & s_t = s \quad and \quad a = a_t \\ 0 & s_t = s \quad and \quad a \neq a_t \\ \gamma\lambda e_{t-1}(s,a) & s_t \neq s \end{cases}$$

- Testing this formula with the same example shows that this works even better.

Implementation Issues

- *Methods with ETs* may seem to *enhance the computational cost a lot* since they require the computation of the ETs of every state (or, even more dramatically, every state-action pair).
- Thank to the *fastly decaying $\gamma\lambda$ factor* of ETs, however, one can see that, for typical values of λ and γ, *the ETs of only the recently visited states are significant*. *Those of almost all other states are almost always nearly zero*.
- ***Consequence*: Sufficient to keep record of those states only with significant values of ETs!**

Variable λ

- An advanced topic;
- Open to research especially on practical applications;
- It involves allowing λ to vary in time (i.e., $\lambda = \lambda_t$).
- An interesting way to vary λ would be to have it change as a *function of states* (i.e., $\lambda_t = \lambda(s_t)$).
- How would you like to have $\lambda(s_t)$ change then?

Variable λ

- For those states whose *values are believed to be known with high certainty* should contribute to the estimate fully (meaning that the traces should be cut off for these states, *λ near 0*);
- Others with *highly uncertain value estimates* should undergo a significant amount of adjustment, meaning a *λ value closer to 1*.

Backward View of Variable λ

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

Forward View of Variable λ

- The general definition of λ-return algorithm ...

$$R_t^{\lambda} = \sum_{n=1}^{\infty} R_t^{(n)} \left(1 - \lambda_{t+n}\right) \prod_{i=t+1}^{t+n-1} \lambda_i$$

- ... can also be expressed in the following way:

$$R_t^{\lambda} = \sum_{k=t+1}^{T-1} R_t^{(k-t)} \left(1 - \lambda_k\right) \prod_{i=t+1}^{k-1} \lambda_i + R_t \prod_{i=t+1}^{T-1} \lambda_i$$

Conclusions

- MC methods are mentioned to have advantages in non-Markov tasks since they do not bootstrap.
- Because ETs make TD methods like MC methods they are also *advantageous in non-Markov tasks*.
- Methods with ETs require more computation than one-step methods, but in return they offer significantly faster learning, particularly when rewards are delayed by many steps. Hence, *ETs are useful when data are scarce and cannot be repeatedly processed, as is the case in most on-line applications*.

References

- [1] Sutton, R. S. and Barto A. G., "*Reinforcement Learning: An introduction*," MIT Press, 1998