Chapter 7: Eligibility Traces
Midterm

Mean = 77.33   Median = 82
N-step TD Prediction

- **Idea:** Look farther into the future when you do TD backup (1, 2, 3, …, n steps)
Mathematics of N-step TD Prediction

- **Monte Carlo:** 
  \[ R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T \]

- **TD:** 
  \[ R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}) \]
  - Use V to estimate remaining return

- **n-step TD:**
  - 2 step return: 
    \[ R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2}) \]
  - n-step return: 
    \[ R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}) \]
Learning with N-step Backups

- Backup (on-line or off-line):
  \[ \Delta V_t(s_t) = \alpha \left[ R^{(n)}_t - V_t(s_t) \right] \]

- Error reduction property of n-step returns
  \[ \max_s |E_\pi \{ R^n_t | s_t = s \} - V_\pi(s) | \leq \gamma^n \max_s |V(s) - V_\pi(s) | \]

- Using this, you can show that n-step methods converge
Random Walk Examples

How does 2-step TD work here?

How about 3-step TD?
A Larger Example

- Task: 19 state random walk

- Do you think there is an optimal n (for everything)?
Averaging N-step Returns

- n-step methods were introduced to help with TD(λ) understanding
- Idea: backup an average of several returns
  - e.g. backup half of 2-step and half of 4-step
    
    $R_t^{avg} = \frac{1}{2} R_t^{(2)} + \frac{1}{2} R_t^{(4)}$

- Called a complex backup
  - Draw each component
  - Label with the weights for that component
Forward View of TD($\lambda$)

- TD($\lambda$) is a method for averaging all n-step backups
  - weight by $\lambda^{n-1}$ (time since visitation)
  - $\lambda$-return:
    \[
    R_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}
    \]
- Backup using $\lambda$-return:
  \[
  \Delta V_t(s_t) = \alpha \left[ R_t^\lambda - V_t(s_t) \right]
  \]
  \[
  \sum = 1 \quad \lambda^{T-t-1}
  \]
\( \lambda \)-return Weighting Function

- Weight given to the 3-step return
- Decay by \( \lambda \)
- Weight given to actual, final return
- Total area = 1

Time

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Relation to TD(0) and MC

- \( \lambda \)-return can be rewritten as:

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

Until termination  After termination

- If \( \lambda = 1 \), you get MC:

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

- If \( \lambda = 0 \), you get TD(0)

\[
R_t^\lambda = (1 - 0) \sum_{n=1}^{T-t-1} 0^{n-1} R_t^{(n)} + 0^{T-t-1} R_t = R_t^{(1)}
\]
Forward View of TD(\(\lambda\)) II

- Look forward from each state to determine update from future states and rewards:
**λ-return on the Random Walk**

- Same 19 state random walk as before
- Why do you think intermediate values of λ are best?
Backward View of TD(\(\lambda\))

- The forward view was for theory
- The backward view is for mechanism

- New variable called *eligibility trace* \(e_t(s)\) \(\sum^+\)
  - On each step, decay all traces by \(\gamma\lambda\) and increment the trace for the current state by 1
  - Accumulating trace

\[
e_t(s) = \begin{cases} 
\gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t \\
\gamma \lambda e_{t-1}(s) + 1 & \text{if } s = s_t
\end{cases}
\]
On-line Tabular TD(\(\lambda\))

Initialize \(V(s)\) arbitrarily and \(e(s) = 0\), for all \(s \in S\)

Repeat (for each episode):

 Initialize \(s\)

 Repeat (for each step of episode):
   
   \(a \leftarrow \) action given by \(\pi\) 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 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
Backward View

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

- Shout \( \delta_t \) backwards over time
- The strength of your voice decreases with temporal distance by \( \gamma \lambda \)
Relation of Backwards View to MC & TD(0)

- Using update rule:
  \[ \Delta V_t(s) = \alpha \delta_t e_t(s) \]

- As before, if you set \( \lambda \) to 0, you get to TD(0)
- If you set \( \lambda \) to 1, you get MC but in a better way
  - Can apply TD(1) to continuing tasks
  - Works incrementally and on-line (instead of waiting to the end of the episode)
Forward View = Backward View

- The forward (theoretical) view of TD(\(\lambda\)) is equivalent to the backward (mechanistic) view for off-line updating
- The book shows:

\[
\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) = \sum_{t=0}^{T-1} \Delta V_t^{\lambda}(s_t) I_{ss_t}
\]

Backward updates  Forward updates

- On-line updating with small \(\alpha\) is similar
On-line versus Off-line on Random Walk

- Same 19 state random walk
- On-line performs better over a broader range of parameters
Control: Sarsa($\lambda$)

- Save eligibility for state-action pairs instead of just states

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

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

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

$$\sum = 1$$
Sarsa(\(\lambda\)) Algorithm

Initialize \(Q(s,a)\) arbitrarily 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 \(Q\) (e.g. \(\epsilon\) - greedy)
      \(\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)\)
      \(e(s,a) \leftarrow e(s,a) + 1\)
   For all \(s,a\) :
      \(Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)\)
      \(e(s,a) \leftarrow \gamma \lambda e(s,a)\)
   \(s \leftarrow s';a \leftarrow a'\)
Until \(s\) is terminal
Sarsa(\(\lambda\)) Gridworld Example

- With one trial, the agent has much more information about how to get to the goal
  - not necessarily the best way
- Can considerably accelerate learning
Three Approaches to $Q(\lambda)$

- How can we extend this to $Q$-learning?
- If you mark every state action pair as eligible, you backup over non-greedy policy
  - Watkins: Zero out eligibility trace after a non-greedy action. Do max when backing up at first non-greedy choice.

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

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

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)$$
Watkins’s Q(\(\lambda\))

Initialize \(Q(s,a)\) arbitrarily 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 \(Q\) (e.g. \(?\) - greedy)

\[a^* \leftarrow \arg\max_b Q(s',b)\] (if \(a\) ties for the max, then \(a^* \leftarrow a'\))

\[\delta \leftarrow r + \gamma Q(s',a') - Q(s,a^*)\]

\[e(s,a) \leftarrow e(s,a) + 1\]

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(\(\lambda\))

- Disadvantage to Watkins’s method:
  - Early in learning, the eligibility trace will be “cut” (zeroed out) frequently resulting in little advantage to traces

- Peng:
  - Backup max action except at end
  - Never cut traces

- Disadvantage:
  - Complicated to implement
Naïve Q(λ)

- Idea: is it really a problem to backup exploratory actions?
  - Never zero traces
  - Always backup max at current action (unlike Peng or Watkins’s)
- Is this truly naïve?
- Works well is preliminary empirical studies

What is the backup diagram?
Comparison Task

- Compared Watkins’s, Peng’s, and Naïve (called McGovern’s here) Q(\(\lambda\)) on several tasks.
  - See McGovern and Sutton (1997). Towards a Better Q(\(\lambda\)) for other tasks and results (stochastic tasks, continuing tasks, etc)

- Deterministic gridworld with obstacles
  - 10x10 gridworld
  - 25 randomly generated obstacles
  - 30 runs
  - \(\alpha = 0.05, \gamma = 0.9, \lambda = 0.9, \varepsilon = 0.05\), accumulating traces

From McGovern and Sutton (1997). Towards a better Q(\(\lambda\))
Comparison Results

From McGovern and Sutton (1997). Towards a better $Q(\lambda)$. 

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Convergence of the $Q(\lambda)$’s

- None of the methods are proven to converge.
  - Much extra credit if you can prove any of them.
- Watkins’s is thought to converge to $Q^*$
- Peng’s is thought to converge to a mixture of $Q^\pi$ and $Q^*$
- Naïve - $Q^*$?
Eligibility Traces for Actor-Critic Methods

- **Critic:** On-policy learning of $V^\pi$. Use TD($\lambda$) as described before.

- **Actor:** Needs eligibility traces for each state-action pair.

- We change the update equation:

$$p_{t+1}(s,a) = \begin{cases} p_t(s,a) + \alpha \delta_t & \text{if } a = a_t \text{ and } s = s_t \\ p_t(s,a) & \text{otherwise} \end{cases} \quad \text{to} \quad p_{t+1}(s,a) = p_t(s,a) + \alpha \delta_t e_t(s,a)$$

- Can change the other actor-critic update:

$$p_{t+1}(s,a) = \begin{cases} p_t(s,a) + \alpha \delta_t [1 - \pi(s,a)] & \text{if } a = a_t \text{ and } s = s_t \\ p_t(s,a) & \text{otherwise} \end{cases} \quad \text{to} \quad p_{t+1}(s,a) = p_t(s,a) + \alpha \delta_t e_t(s,a)$$

where

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

- Using accumulating traces, frequently visited states can have eligibilities greater than 1
  - This can be a problem for convergence

Replacing traces: Instead of adding 1 when you visit a state, set that trace to 1

\[
e_t(s) = \begin{cases} 
\gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t \\
1 & \text{if } s = s_t
\end{cases}
\]
Replacing Traces Example

- Same 19 state random walk task as before
- Replacing traces perform better than accumulating traces over more values of $\lambda$
Why Replacing Traces?

- Replacing traces can significantly speed learning
- They can make the system perform well for a broader set of parameters
- Accumulating traces can do poorly on certain types of tasks

Why is this task particularly onerous for accumulating traces?
More Replacing Traces

- Off-line replacing trace TD(1) is identical to first-visit MC

- Extension to action-values:
  - When you revisit a state, what should you do with the traces for the other actions?
  - Singh and Sutton say to set them to zero:

\[
e_t(s, a) = \begin{cases} 
1 & \text{if } s = s_t \text{ and } a = a_t \\
0 & \text{if } s = s_t \text{ and } a \neq a_t \\
\gamma \lambda e_{t-1}(s, a) & \text{if } s \neq s_t 
\end{cases}
\]
Implementation Issues

- Could require much more computation
  - But most eligibility traces are VERY close to zero
- If you implement it in Matlab, backup is only one line of code and is very fast (Matlab is optimized for matrices)
Variable $\lambda$

- Can generalize to variable $\lambda$

\[
\lambda_t = \begin{cases} 
\gamma \lambda_t e_{t-1}(s) & \text{if } s \neq s_t \\
\gamma \lambda_t e_{t-1}(s) + 1 & \text{if } s = s_t 
\end{cases}
\]

- Here $\lambda$ is a function of time
  - Could define

\[
\lambda_t = \lambda(s_t) \text{ or } \lambda_t = \lambda^{t/\tau}
\]
**Conclusions**

- Provides efficient, incremental way to combine MC and TD
  - Includes advantages of MC (can deal with lack of Markov property)
  - Includes advantages of TD (using TD error, bootstrapping)
- Can significantly speed learning
- Does have a cost in computation
Something Here is Not Like the Other

a) Backward View

b) Forward View