Reinforcement Learning

Monte Carlo and TD(λ) learning

Mario Martin

Universitat politècnica de Catalunya

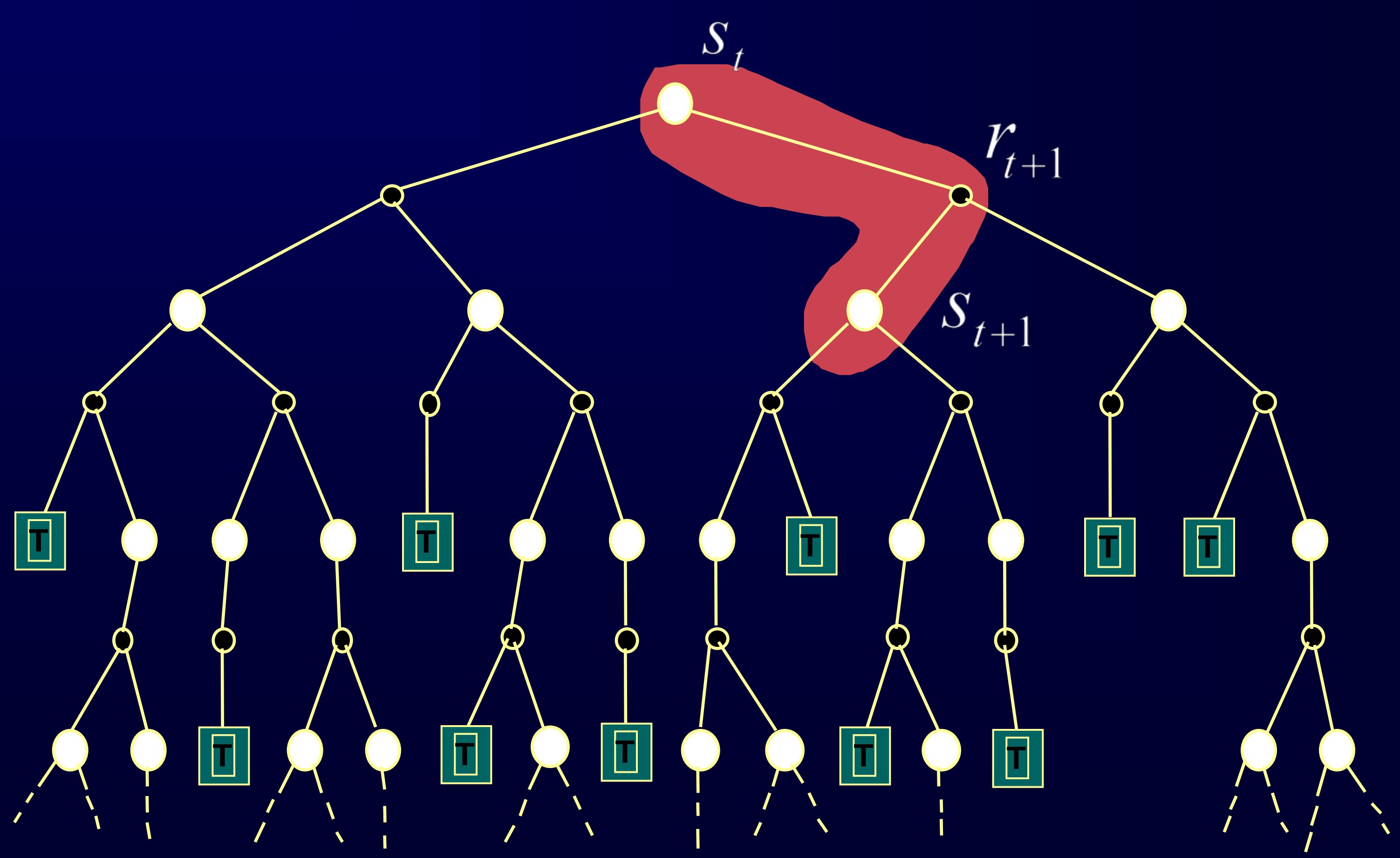
Dept. LSI

Monte Carlo

- Only for trial based learning
- Values for each state or pair state-action are updated only based on final reward, not on estimations of neighbor states

Temporal Difference backup

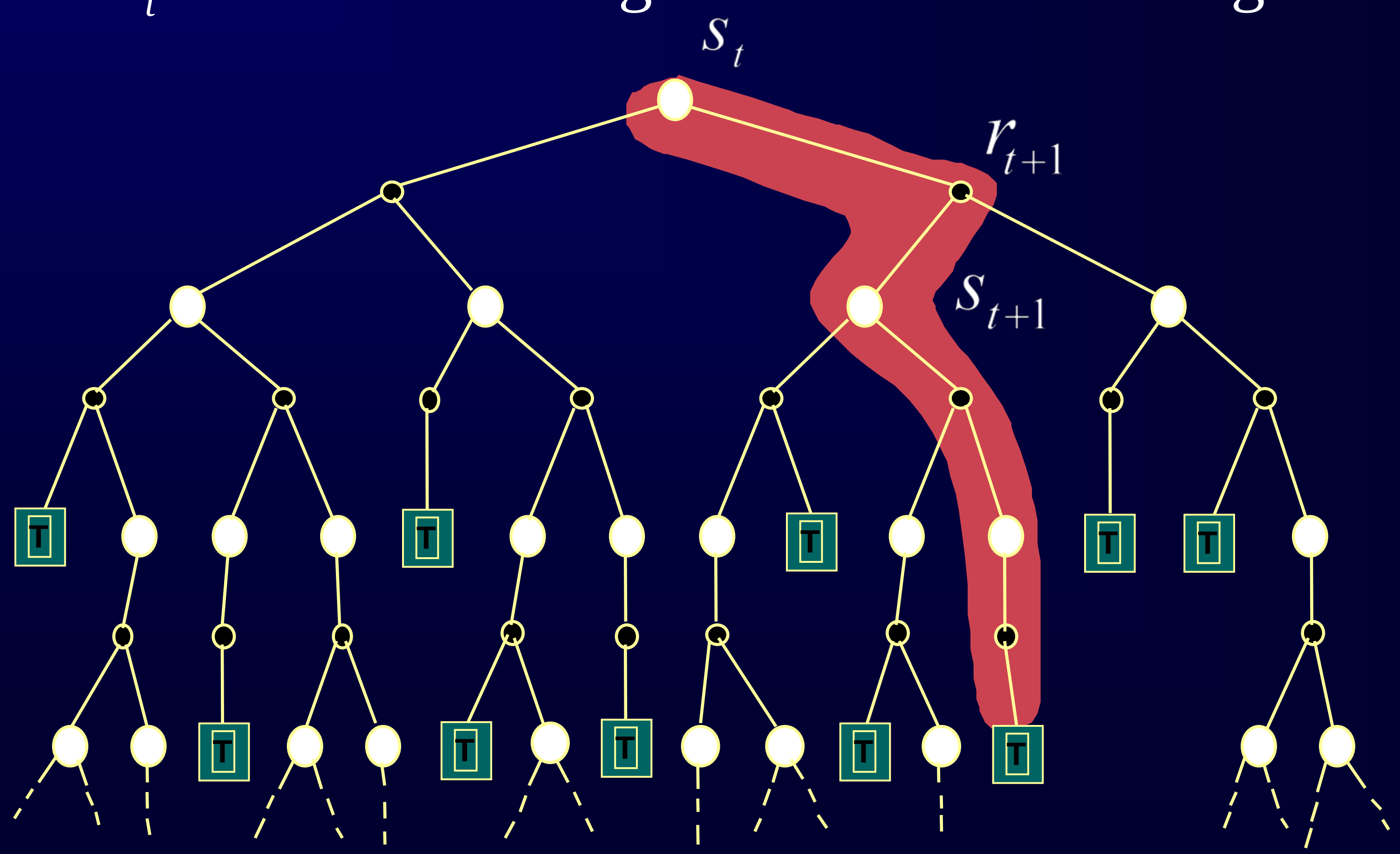
$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$



Monte Carlo

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

where R_t is the actual long-term return following state s_t.



Comparison of TD and MC Learning

- Both TD and MC methods do not require a model of the environment, only experience (not with DP)
- TD, but not MC, methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences
- Both MC and TD converge (under certain assumptions to be detailed later), but which is faster? (we will see that later)

N-step TD Prediction

- Instead of calculating the error in terms of the estimation of the next state, use n-steps future state estimation:

$$Q(s_t, a_t) = r_{t+1} + \underbrace{\gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{n+1} + \ldots}$$

$$Q^{1)}(s_t, a_t) = r_{t+1} + \qquad \gamma V(s_{t+1})$$

One-step predictor

N-step TD Prediction

- Instead of calculating the error in terms of the estimation of the next state, use n-steps future state estimation:

$$Q(s_t, a_t) = r_{t+1} + \gamma r_{t+2} + \underbrace{\gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{n+1} + \ldots}_{}$$

$$Q^{2)}(s_t, a_t) = r_{t+1} + \gamma r_{t+2} + \qquad \gamma^2 V(s_{t+2})$$

Two-steps predictor

N-step TD Prediction

- Instead of calculating the error in terms of the estimation of the next state, use n-steps future state estimation:

$$Q(s_t,a_t)=r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+\underbrace{\ldots+\gamma^{n-1}r_{n+1}+\ldots}$$

$$Q^{3)}(s_t,a_t)=r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+\qquad \gamma^3 V(s_{t+3})$$

Three-steps predictor

N-step TD Prediction

- Instead of calculating the error in terms of the estimation of the next state, use n-steps future state estimation:

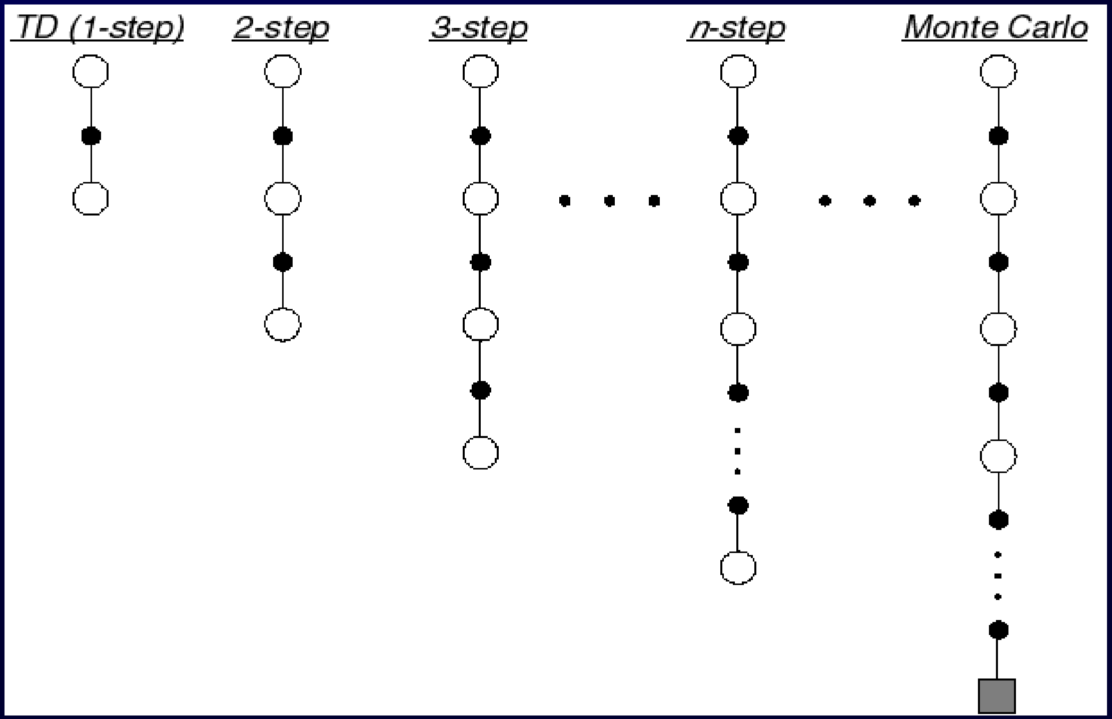
$$Q(s_t, a_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{n+1} + \ldots$$

$$Q^{n)}(s_t, a_t) = \left(\sum_{i=1}^{n} \gamma^{i-1} r_{t+i} \right) + \gamma^n V(s_{t+n})$$

n-steps predictor

N-step TD Prediction

- Idea: Look farther into the future when you do TD backup (1, 2, 3, …, n steps)



Lots of TD predictors for long term reward

- TD 1-step: $R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$

- 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: $R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$

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

N-steps Estimations

- N-steps back-up

$$Q^{n)}(s_t,a_t)=Q^{n)}(s_t,a_t)+\alpha\left(\left(\sum_{i=1}^{n}\gamma^{i\text{-}1}r_{t+1}\right)+\gamma^{n}V(s_{t+n})-Q^{n)}(s_t,a_t)\right)$$

- A lot of predictors. Which one to use? Trust in only one can lead to bad results because it can be severely biased.
- But, it is so biased to use the 1-step back-up as to use the n-steps back-up

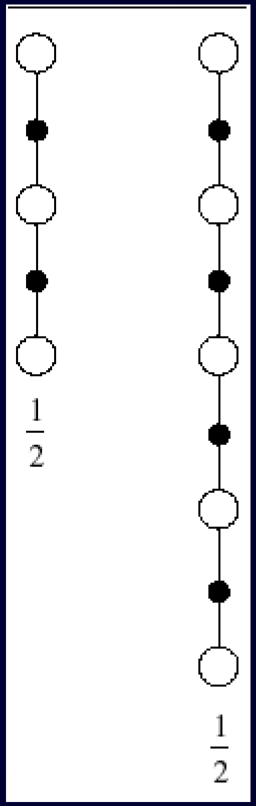
Averaging N-step Returns

- 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
  - Choose each component
  - Label with the weights for that component

One backup



TD(λ)

- Better, use all the next estimations. Pass credit not only based in the next state but in the whole set of next states

- Value estimation for the n-steps

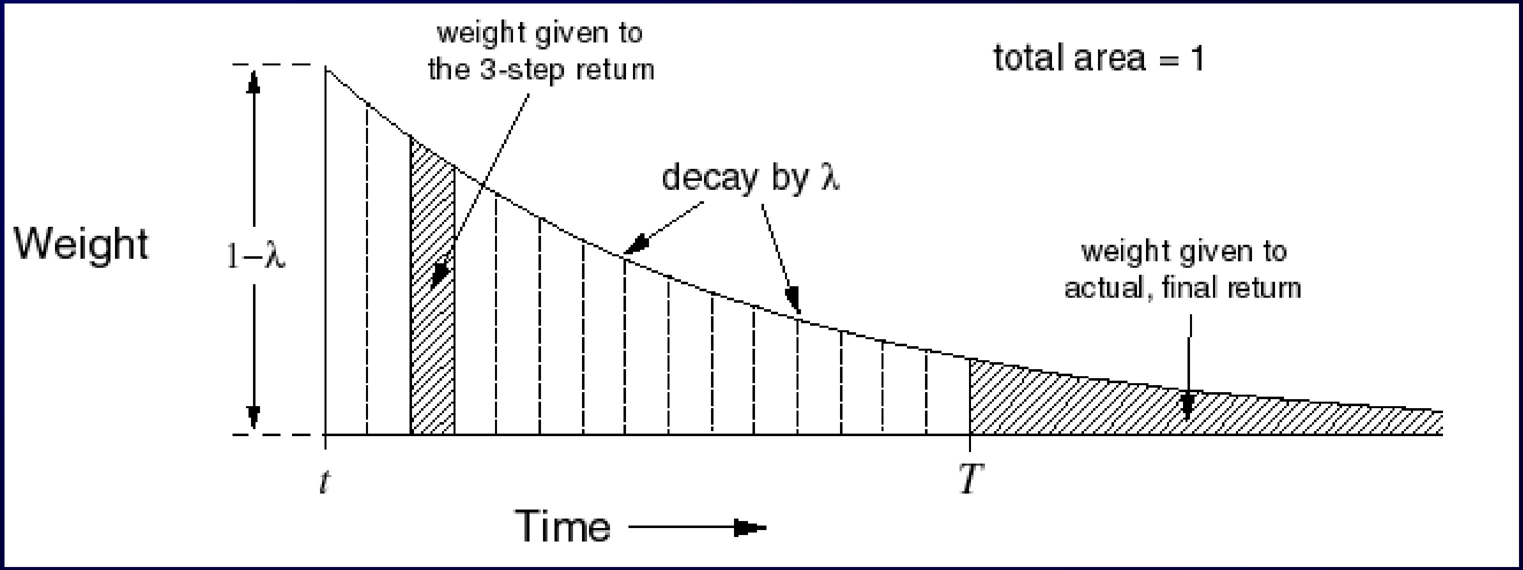
$$V^{n)}(s_t) = \left(\sum_{i=1}^{n} \gamma^{i-1} r_{t+1} \right) + \gamma^n V(s_{t+n})$$

TD(λ)

- Define the new estimation $V^{\lambda}(s)$ as the geometrical average with parameter (0≤λ≤1)

$$V^{\lambda}(s_t) = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} \, V^{n)}(s_t)$$

λ-return Weighting Function



$$R_t^\lambda = \underbrace{(1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}R_t^{(n)}}_{\text{Until termination}} + \underbrace{\lambda^{T-t-1}R_t}_{\text{After termination}}$$

TD(λ)

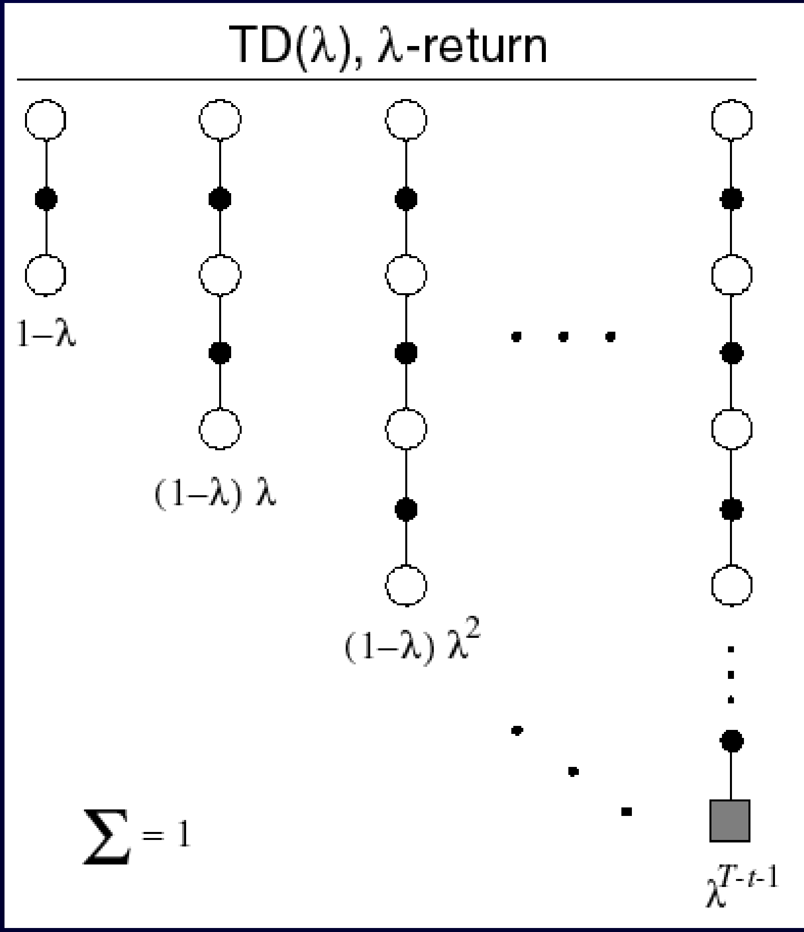
- TD(λ) is a method for averaging all n-step backups
  - weight by λ^{n-1} (time since visitation)

    λ-return:

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

- Backup using λ-return:

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



TD(λ)

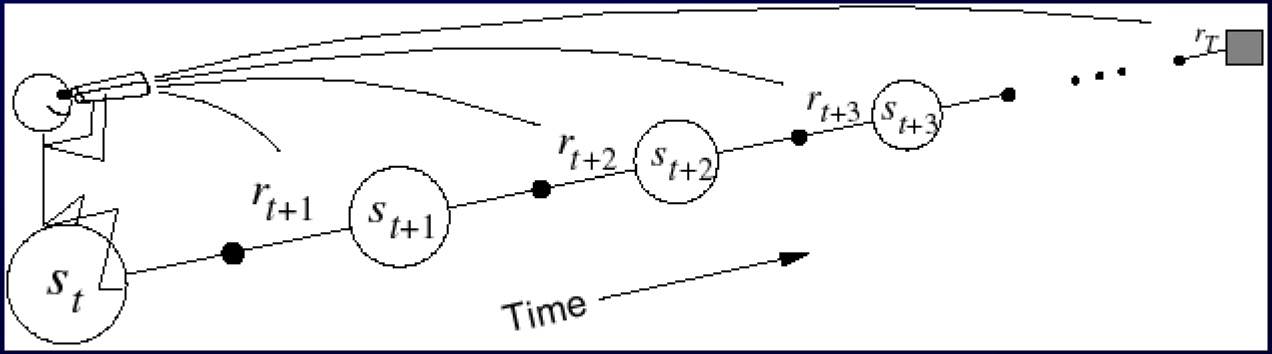
- Back-up

$$V(s_t) = V(s_t) + \alpha\left(V^{\lambda}(s_t) - V(s_t)\right)$$

- Problem: In order to perform the back-up it is necessary to end the trial

Forward View of TD(λ) II

- Look forward from each state to determine update from future states and rewards:



$$V^{\lambda}(s_t) = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\, V^{n)}(s_t)$$

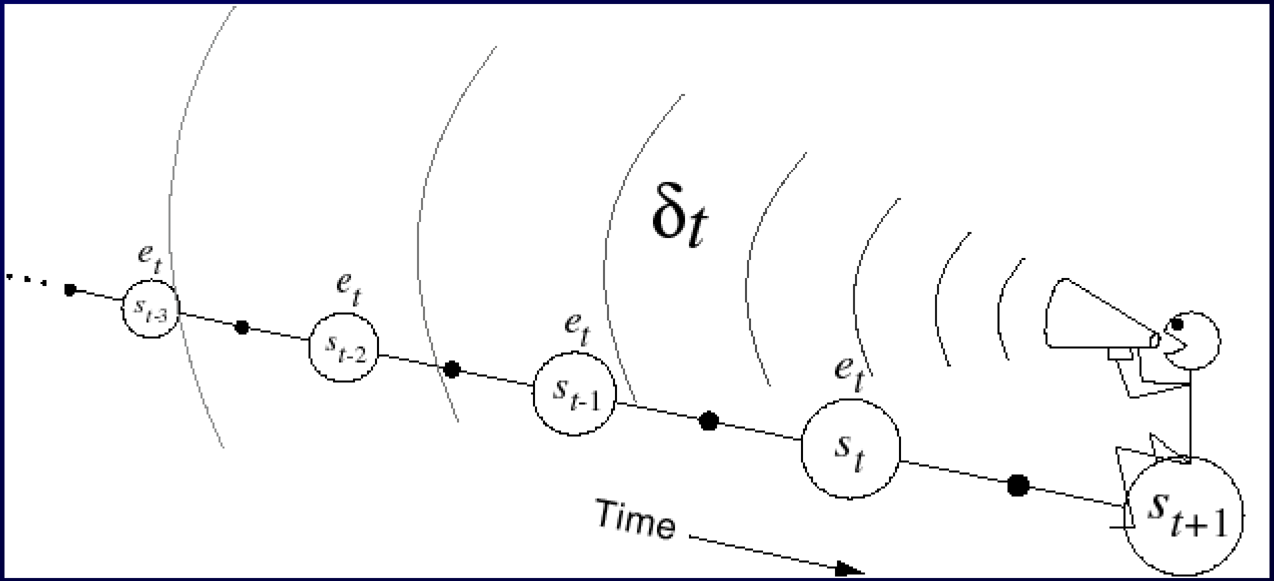
TD(λ)

- Back-up

$$V(s_t) = V(s_t) + \alpha\left(V^{\lambda}(s_t) - V(s_t)\right)$$

- Problem: In order to perform the back-up it is necessary to end the trial
- Solution: Fortunately, this forward view of TD(λ) is equivalent to the following backward view

Backward View

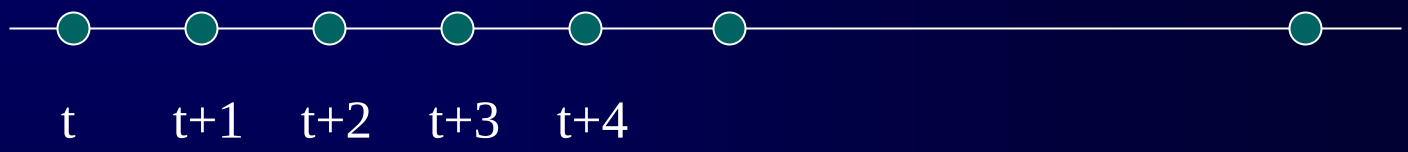


$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

- Shout δ_t backwards over time
- The strength of your voice decreases with temporal distance by $\gamma\lambda$

$$V^{\lambda}(s_t) = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\, V^{n)}(s_t)$$

Back-up not only when visiting the current state but also when visiting the following states

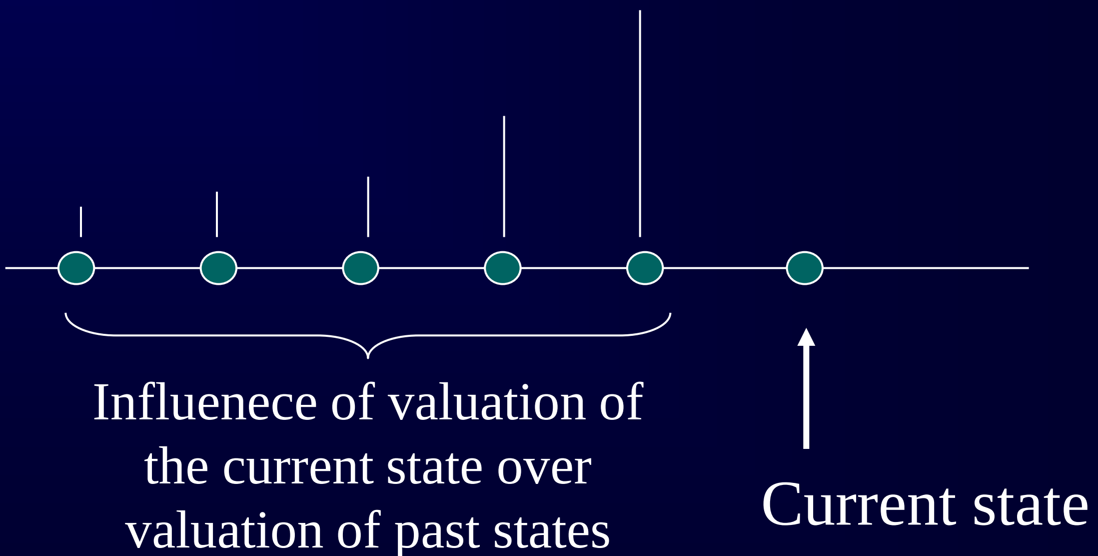


t: $(1-\lambda)\ V(s_t)$

t+1: $(1-\lambda)\ \lambda\ V(s_{t+1})$

t+2: $(1-\lambda)\ \lambda^2\ V(s_{t+2})$

t+3: $(1-\lambda)\ \lambda^3\ V(s_{t+3})$



Backward View of TD(λ)

- New variable called *eligibility trace*
  - 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(λ)

Initialize $V(s)$ arbitrarily

Repeat (for each episode):

  $e(s) = 0$, for all $s \in S$

  Initialize s

  Repeat (for each step of episode):

    $a \leftarrow$ action given by 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 all s in the trace:

      $V(s) \leftarrow V(s) + \alpha \delta e(s)$

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

    $s \leftarrow s'$

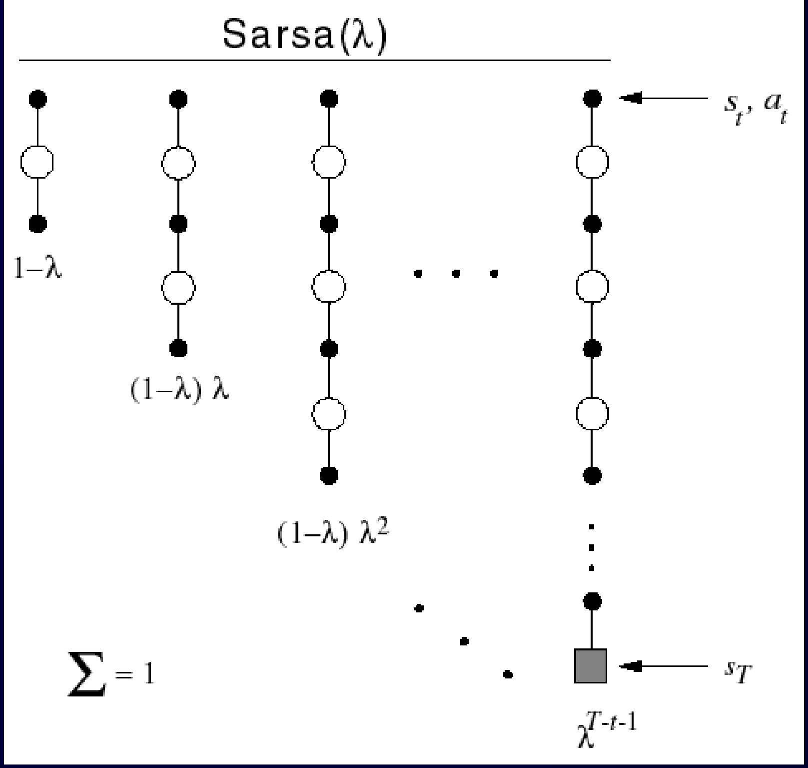
  Until s is terminal

Extensible to Q-functions : Sarsa(λ)

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

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

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



Sarsa(λ) Algorithm

Initialize $Q(s,a)$ arbitrarily

Repeat (for each episode):

  $e(s,a) = 0$, for all s, a

  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. ε-greedy)

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

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

    For all s,a:

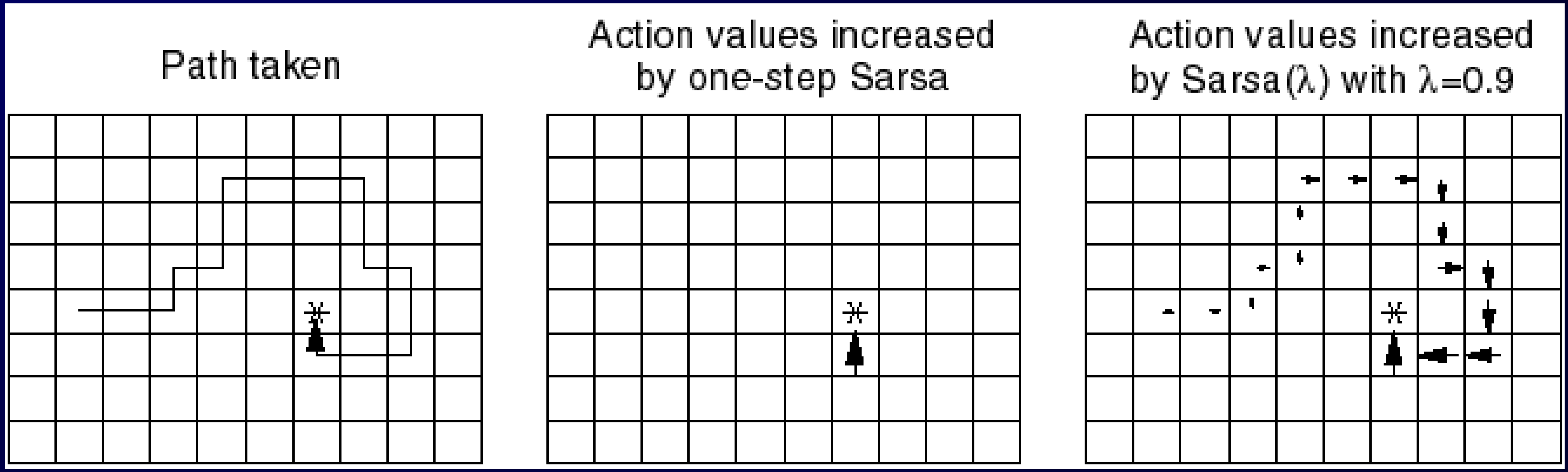
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$

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

    $s \leftarrow s'; a \leftarrow a'$

  Until s is terminal

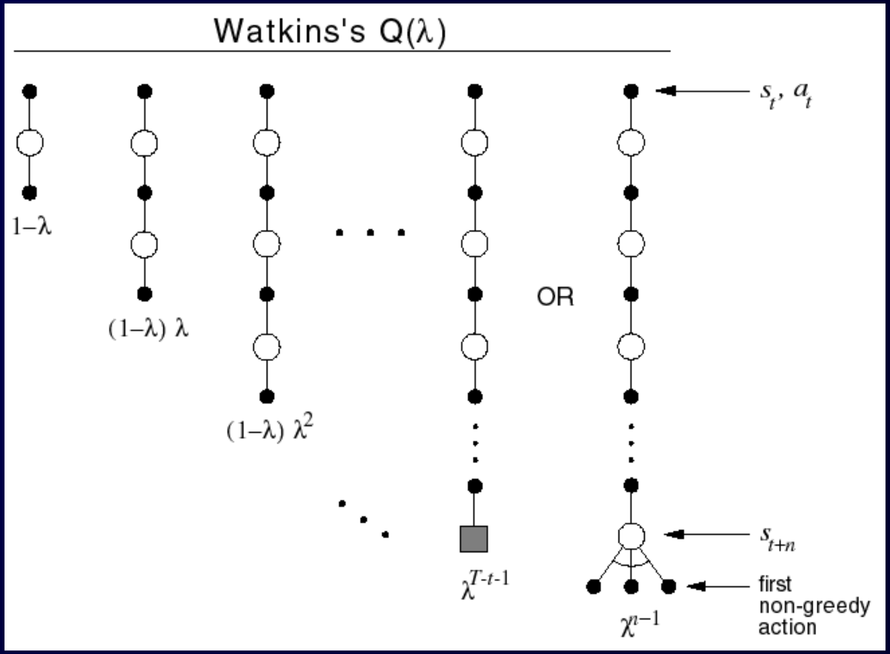
Sarsa(λ) Gridworld Example



- 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

Q-learning(λ)?

- 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(λ)

Initialize $Q(s,a)$ arbitrarily

Repeat (for each episode):

  $e(s,a) = 0$, for all s, a

  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. ε-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 $e(s,a) \leftarrow \gamma\lambda e(s,a)$

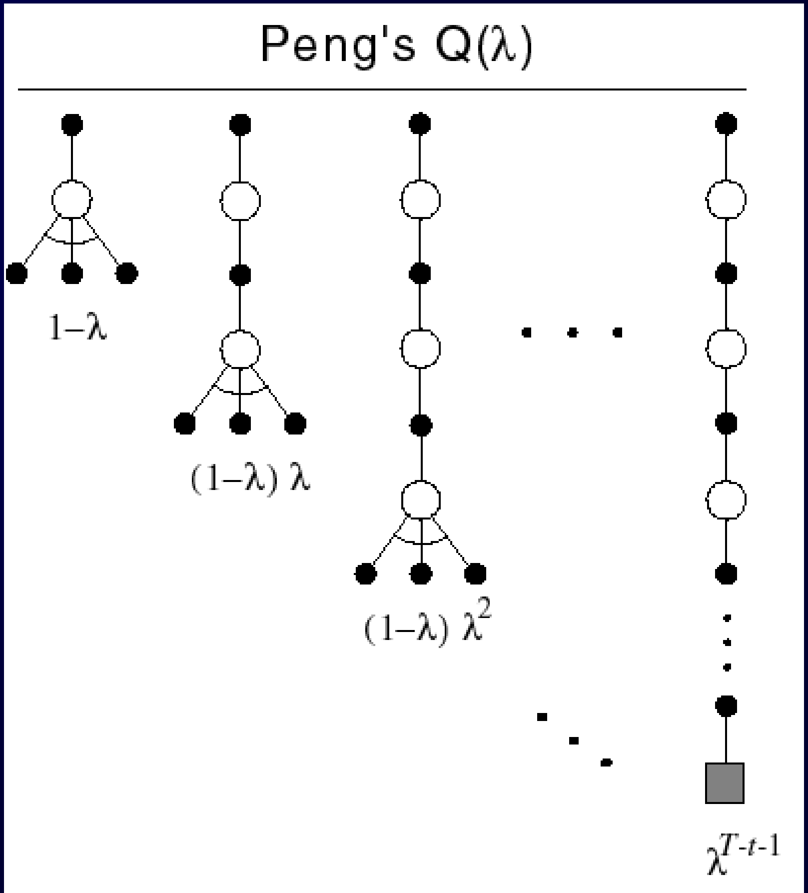
        else $e(s,a) \leftarrow 0$

    $s \leftarrow s'; a \leftarrow a'$

  Until s is terminal

Peng's Q(λ)

- 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



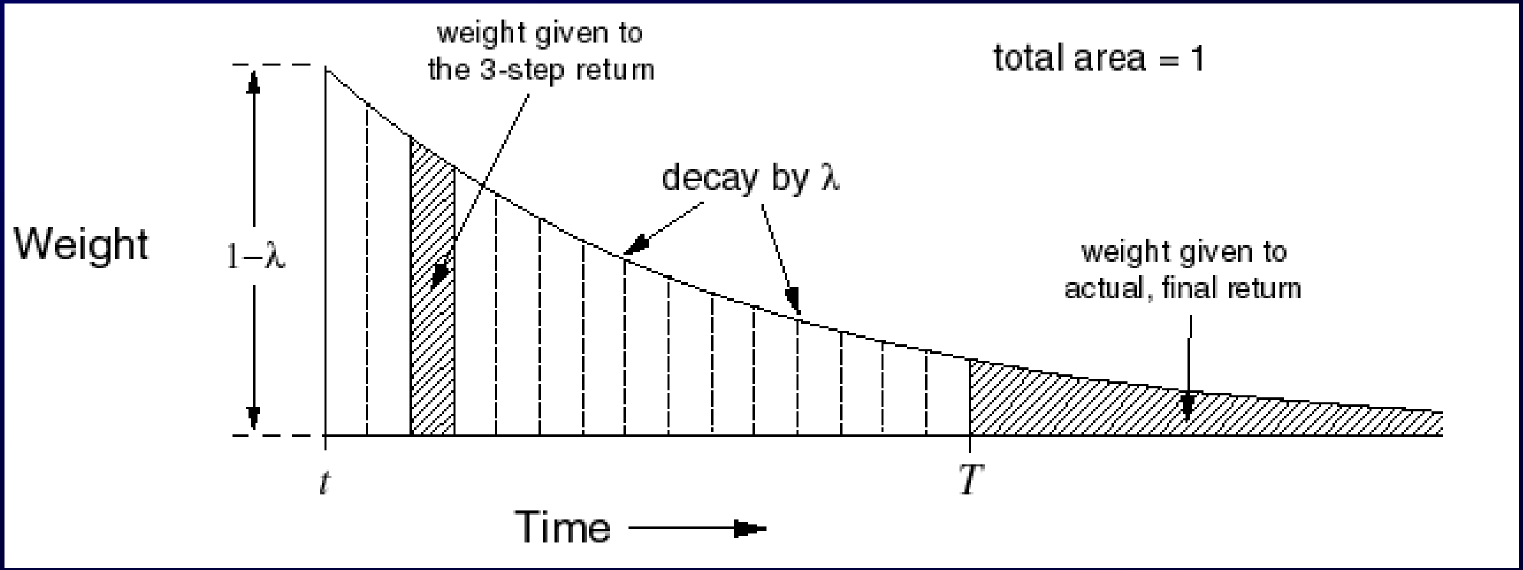
Naïve Q(λ)

- 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 in preliminary empirical studies

Convergence of the Q(λ)'s

- None of the methods are proven to converge.
  - Watkins's is thought to converge to Q^*
  - Peng's is thought to converge to a mixture of Q^π and Q^*
  - Naïve - Q^*?

Relationship between TD(λ), Q-learning and Monte Carlo

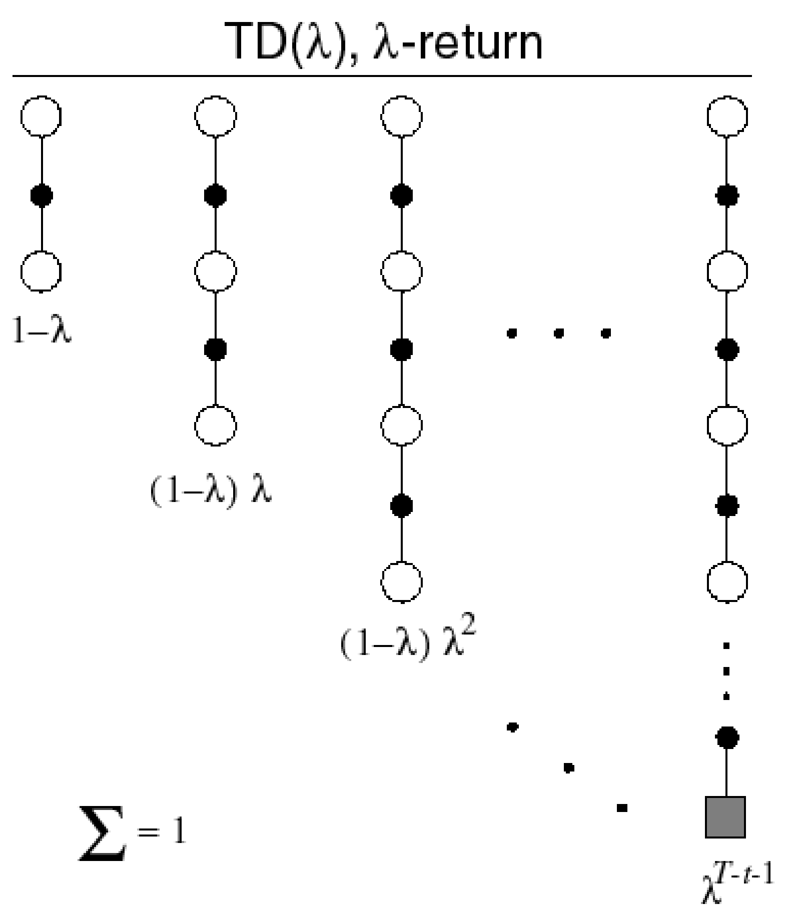


$$R_t^{\lambda} = \underbrace{(1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}R_t^{(n)}}_{\text{Until termination}} + \underbrace{\lambda^{T-t-1}R_t}_{\text{After termination}}$$

Relationship between TD(λ), Q-learning and Monte Carlo

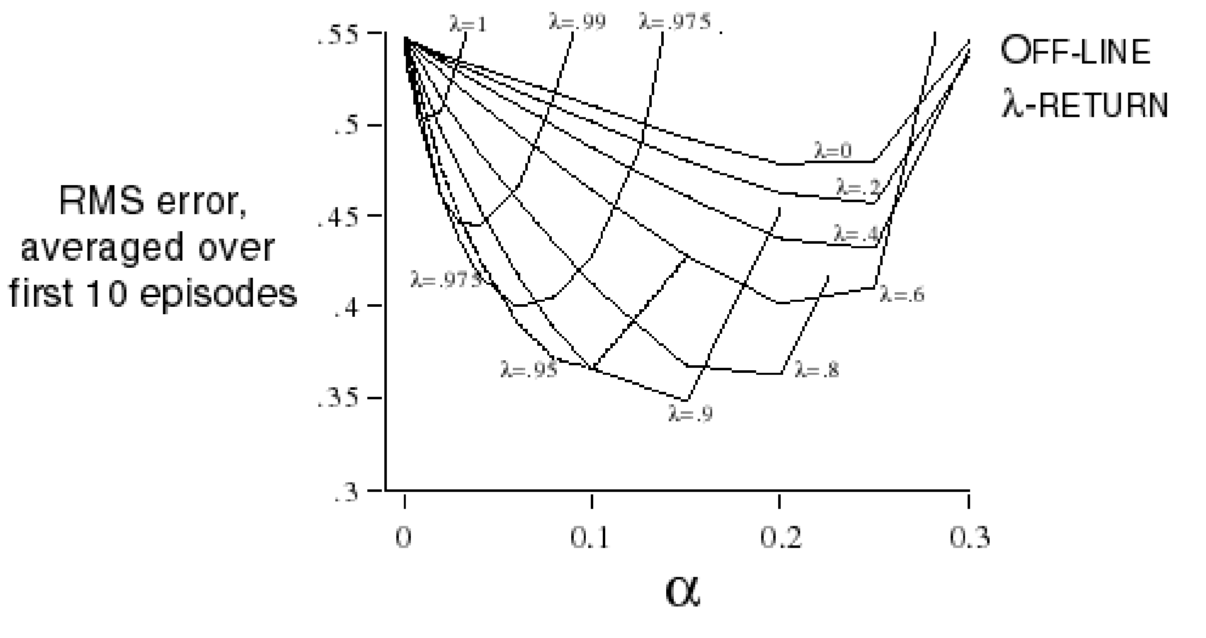
$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t$$

- If λ = 0, you get TD(0)
  - Q-learning
  - Sarsa
- If λ = 1, you get MC



Best λ?

- Not know before-hand
- Empirical example: Prediction of return in a simple Random Walk experiment



Some remarks about TD(λ)

- Extensible to Q-values [Q(λ) and Sarsa(λ)]
- Extensible to continuous learning: Eligibility traces are set to 0 when they are enough small
- Q-learning is a variant of TD(0)
- Monte-Carlo is a variant of TD(1)
- Usually TD(λ) with $\lambda <> 0$ and 1 show better results than TD(0) or TD(1)

Reinforcement Learning

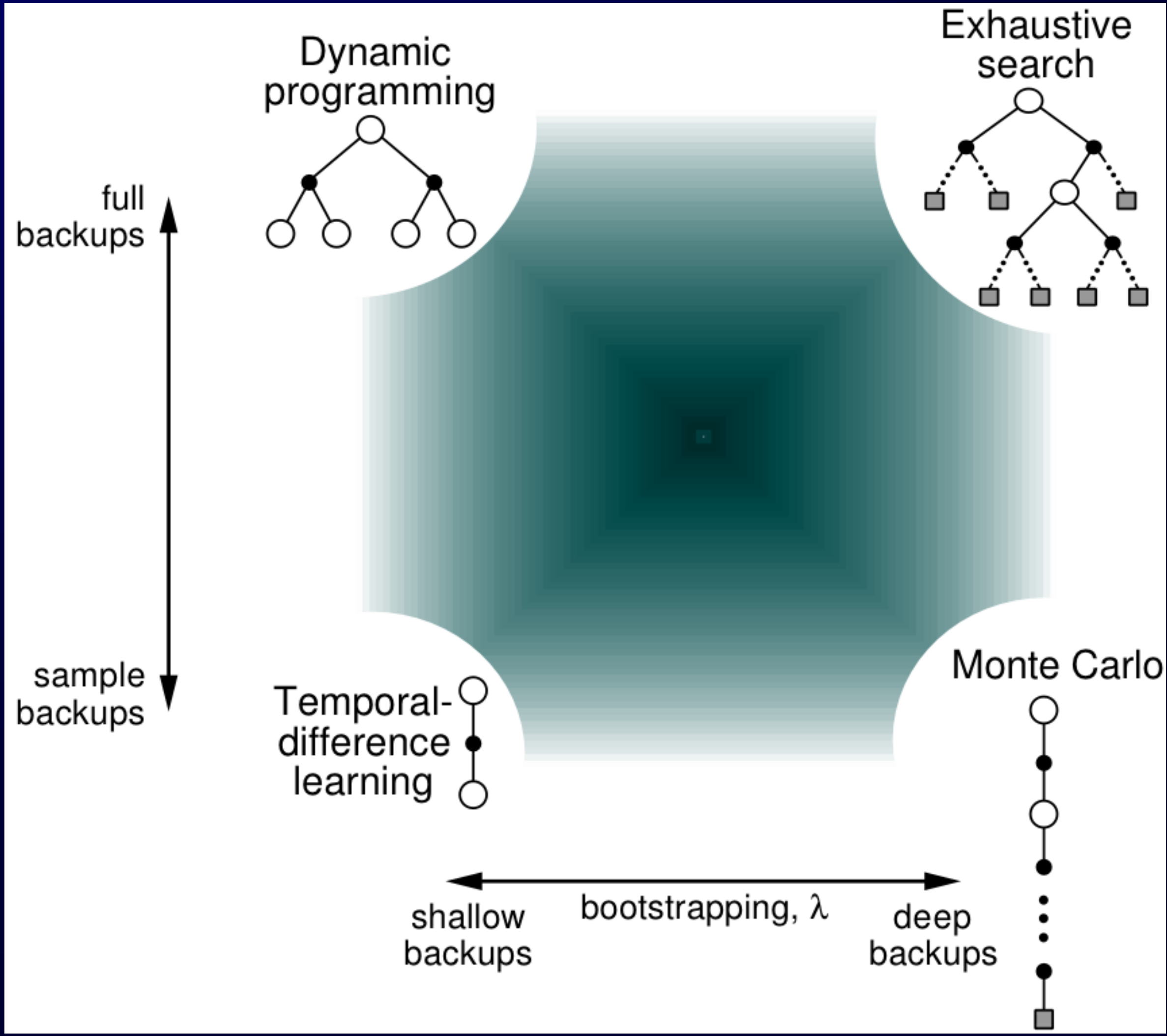
Summary

Mario Martin
Universitat politècnica de Catalunya
Dept. LSI

Three Common Ideas

- Estimation of **value functions**
- **Backing up values** along real or simulated trajectories
- **Generalized Policy Iteration**: maintain an approximate optimal value function and approximate optimal policy, use each to improve the other

Backup Dimensions



Other Dimensions

- On-policy/Off-policy
  - On-policy: learn the value function of the policy being followed
  - Off-policy: try learn the value function for the best policy, irrespective of what policy is being followed

Still More Dimensions

- Definition of return: episodic, continuing, discounted, averaged, etc.
- Action selection/exploration: ε-greed, softmax, more sophisticated methods
- Synchronous vs. asynchronous
- Replacing vs. accumulating traces
- Real vs. simulated experience
- Memory for backups: how long should backed up values be retained?

Some open problems

- Function approximation: methods and convergence
- Incomplete state information
  - Partially Observable MDPs (POMDPs)
  - Try to do the best you can with non-Markov states
- Modularity and/or hierarchies of actions and states
- Exploration procedures
- Using teachers
- Incorporating prior knowledge