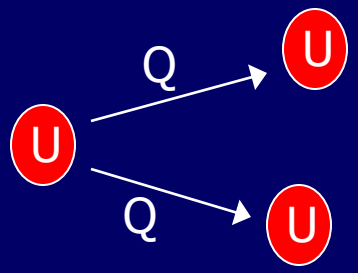
Today's Topics

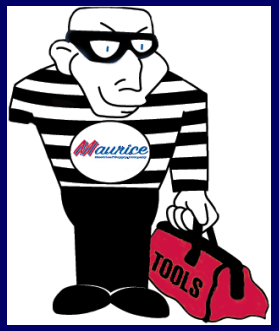
- Exploration vs. Exploitation
- Generalizing Across State
- SARSA
- TD Backups
- HW3 Due FRIDAY (4pm)
- HW4 out this week ('easy' paper-and-pencil HW)
- 'Unofficial' class this Friday (the Agent World)

Q's vs. U's



- Assume we're in state S
  Which action do we choose?
- U's (Model-based)
  - Need to have a 'next state' function to generate all possible states (ie, chess)
  - Choose next state with highest U value.
- Q's (Model-free, though can also do model-based Q learning)
  - Need only know which actions are legal (ie, web)
  - Choose next state with highest Q value.

Exploration vs. Exploitation



In order to learn about better alternatives, we can't always follow the current policy ('exploitation')

Sometimes, need to try random moves ('exploration')

Exploration vs. Exploitation (cont)

Approaches

1) *p* percent of the time, make a random move; could let

$$p = \frac{1}{\sqrt{\#moves_made}}$$

2) Prob(picking action *A* in state *S*)

$$= \frac{const^{Q(S,A)}}{\sum_{i \in actions} const^{Q(S,i)}}$$

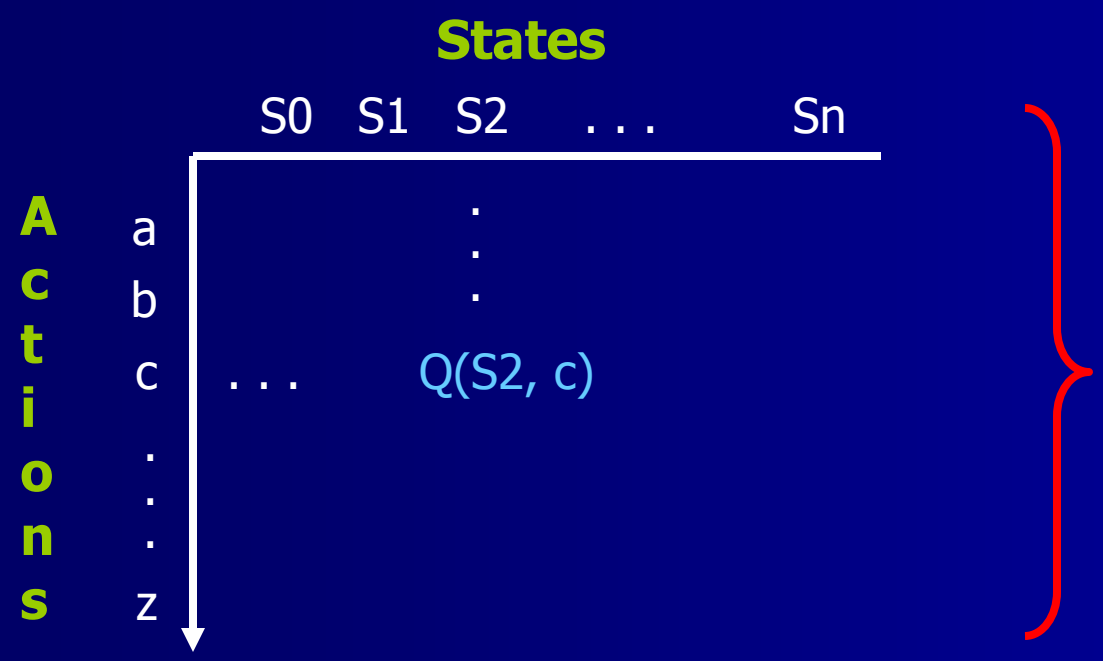
Exponentia-ting gets rid of negative values

One-Step Q-Learning Algo

0. $S \leftarrow$ initial state
1. If random # $\leq P$
   then a = random choice
   Else $a = \Pi_t(S)$
2. $S_{new} \leftarrow W(S, a)$
   $R_{immed} \leftarrow R(S_{new})$
   } Act on world and get reward
3. <u>Error</u> $\leftarrow R_{immed} + \gamma\, U(S_{new}) - Q(S, a)$
4. $Q(S, a) \leftarrow Q(S, a) + \alpha$ <u>Error</u> // Should also decay α
5. $S \leftarrow S_{new}$
6. Go to 1

Q-Learning: Implementation Details

Remember, conceptually we are filling in a <u>huge table</u>



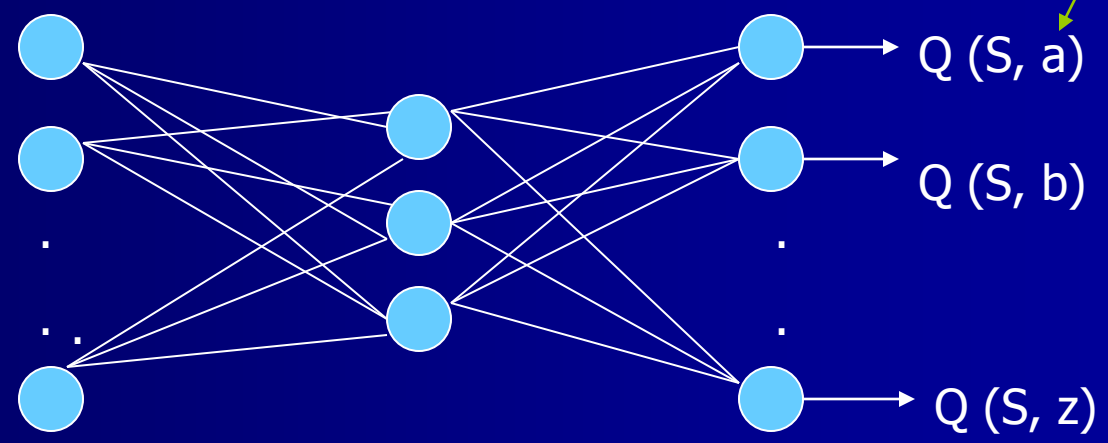
Tables are a very <u>verbose</u> representation of a function

Representing Q Functions More Compactly

We can use some other function representation (eg, neural net) to <u>compactly</u> encode this big table

Second argument is a constant

An encoding of the state (S)



Each input unit encodes a property of the state (eg, a sensor value)

Or could have <u>one net</u> for <u>each</u> possible action

Q Tables vs Q Nets

Given: 100 Boolean-valued features

10 possible actions

Size of Q table

10×2^{100} ← # of possible states

Size of Q net (100 HU's)

$$\underbrace{100 \times 100}_{\text{Weights between inputs and HU's}} + \underbrace{100 \times 10}_{\text{Weights between HU's and outputs}} = 11{,}000$$

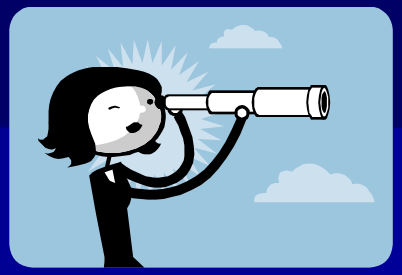
Why Use a Compact Q-Function?

1. Full Q table may not fit in memory for realistic problems
2. Can generalize across states,
   thereby speeding up convergence
   ie, one example 'fills' many cells in the Q table

Notes

1. When generalizing across states, cannot use $\alpha=1$
2. Convergence proofs only apply to Q tables
3. Some work on bounding errors caused by using compact representations (eg, Singh & Lee, MLj)

Connectionist Q-Learning (1-step Lookahead)



<u>Algorithm for training the 'Q-net'</u>

1. Measure sensors (S_0)
2. Predict $Q(S_0, \alpha)$ for each action α

   Let action A be

   i. The best action

   [ie, $Q(S_0,A)$ is highest $Q(S_0,\alpha)$]

   ii. A randomly chosen action

   With probability p, choose between *i* and *ii*

3. Apply action A in the world,

   Re-measure sensors (S_1) and immediate reward (R)

Connectionist Q-Learning (cont)

4. Find action β that maximizes $Q(S_1, \beta)$
   ie compute $U(S_1)$ but <u>don't apply</u> β
5. Train with

   input = S_0

   output = $Q(S_0,A) + \alpha[R + \gamma\, Q(S_1, \beta) - Q(S_0,A)]$

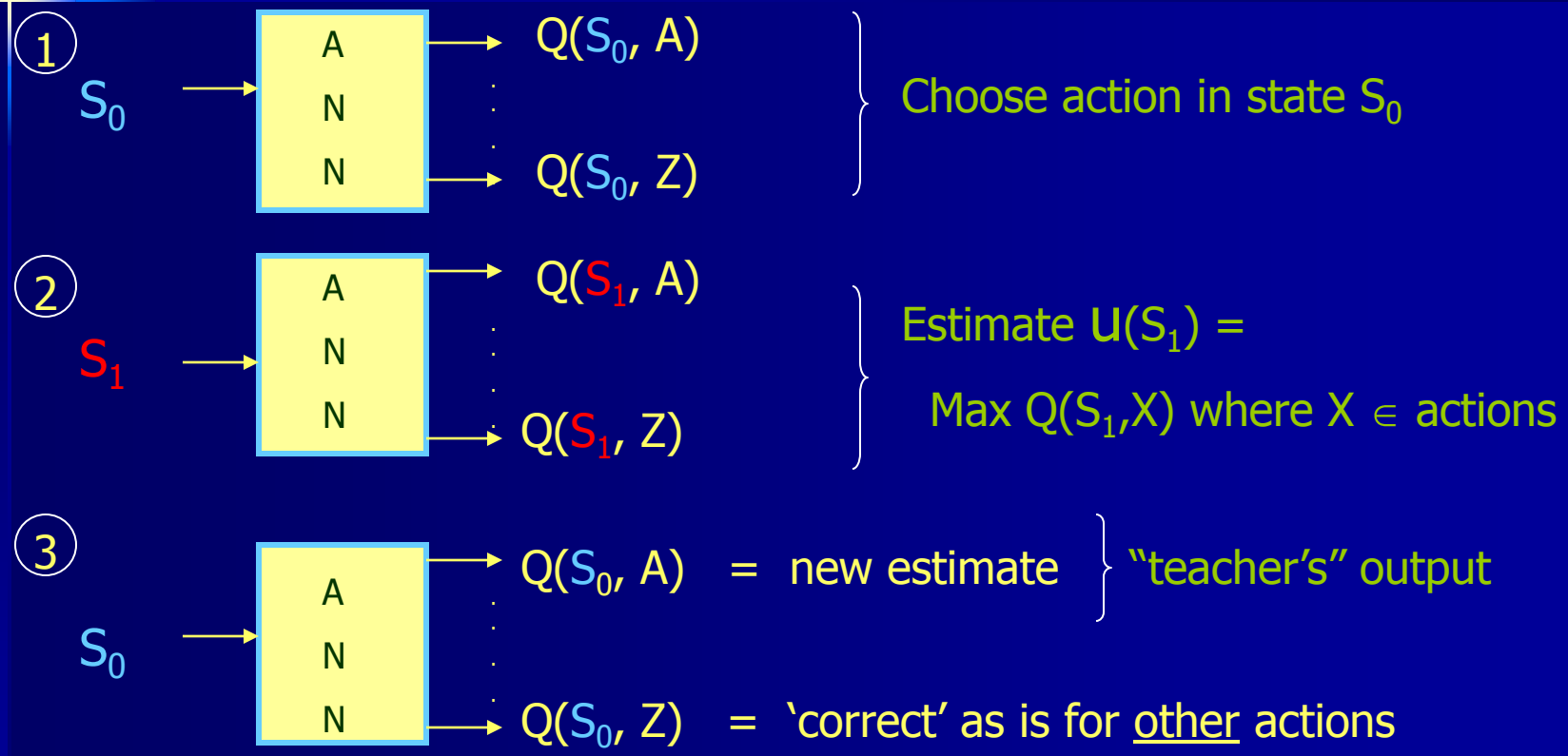
<u>We create a standard I/O pair!</u>

* <u>Only</u> train output corresponding to action *A*

Typically, <u>discard</u> I/O pair after <u>one</u> backprop

Two Forward Props and a BackProp

3?



Arguably this is THREE forward props, but could cache result of 1 for use in 3

k-NN for Q-Learning

Could also use (relational) regression trees, support-vector regression, etc

Represent *Q(s, a)* using *k*-NN

One Possible Design

For *each* action, keep set of I/O pairs

Data Point

input = S_0

output = $Q(S_0, A) + \alpha [R + \gamma Q(S_1, B) - Q(S_0, A)]$

just like for the ANN

Measure similarity to previous states, average the outputs of neighbors to predict ***Q(s, a)***

(Might want to discard old examples since Q's out of date)

SARSA vs. Q-Learning

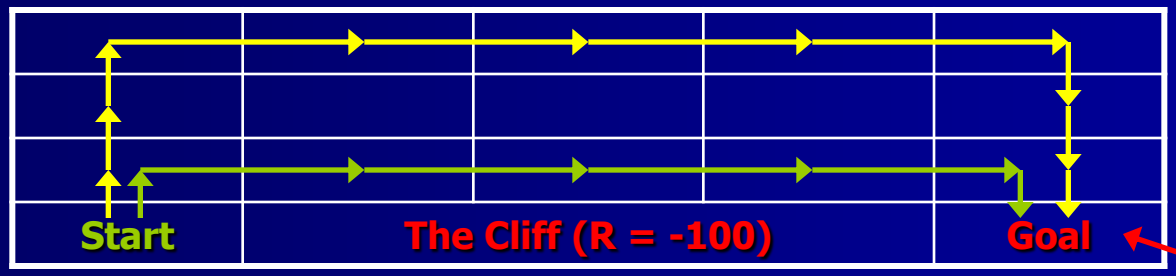
(1994, 1996) (1989)

Exploring can be hazardous!

Should we learn to consider its impact?

The Cliff-Walking Task (pg 150 of Sutton + Barto RL Text)

Safe route



R=-1 for all of these

R=0

Optimal path
-if no exploration!

What would Q-Learning learn?

SARSA = State Action Reward State Action

SARSA

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\,R + \gamma\, \mathbf{Q(s', a')} - Q(s,a)\,]$$

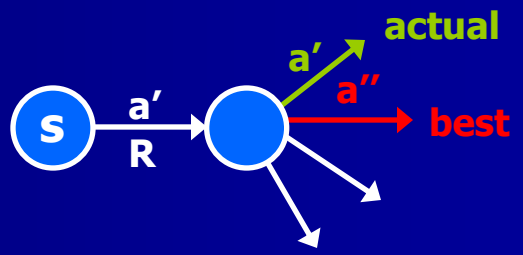
Standard Q-Learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\,R + \gamma\, \mathbf{\underline{max}\ Q(s', a'')} - Q(s,a)\,]$$

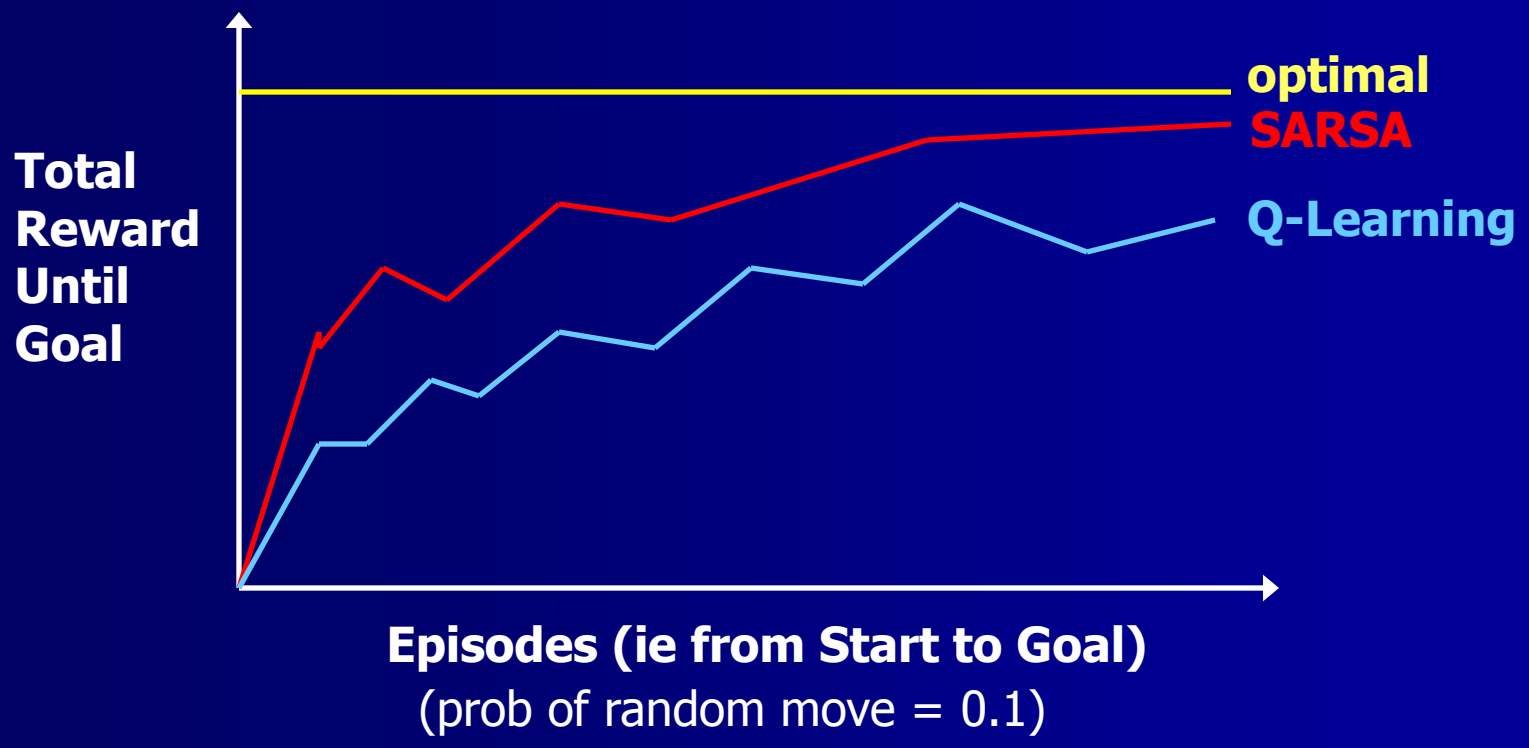
SARSA uses actual next action (still chosen via explore-exploit strategy, e.g. soft-max or "coin-flip")

- SARSA also converges

Notice that in Q learning, (currently) non-optimal moves do not impact the Q function



Sample Results: Cliff-Walking Task



SARSA learns to avoid 'pitfalls' while in <u>exploration</u> phase (<u>always</u> need to explore if in a real-world situation?)

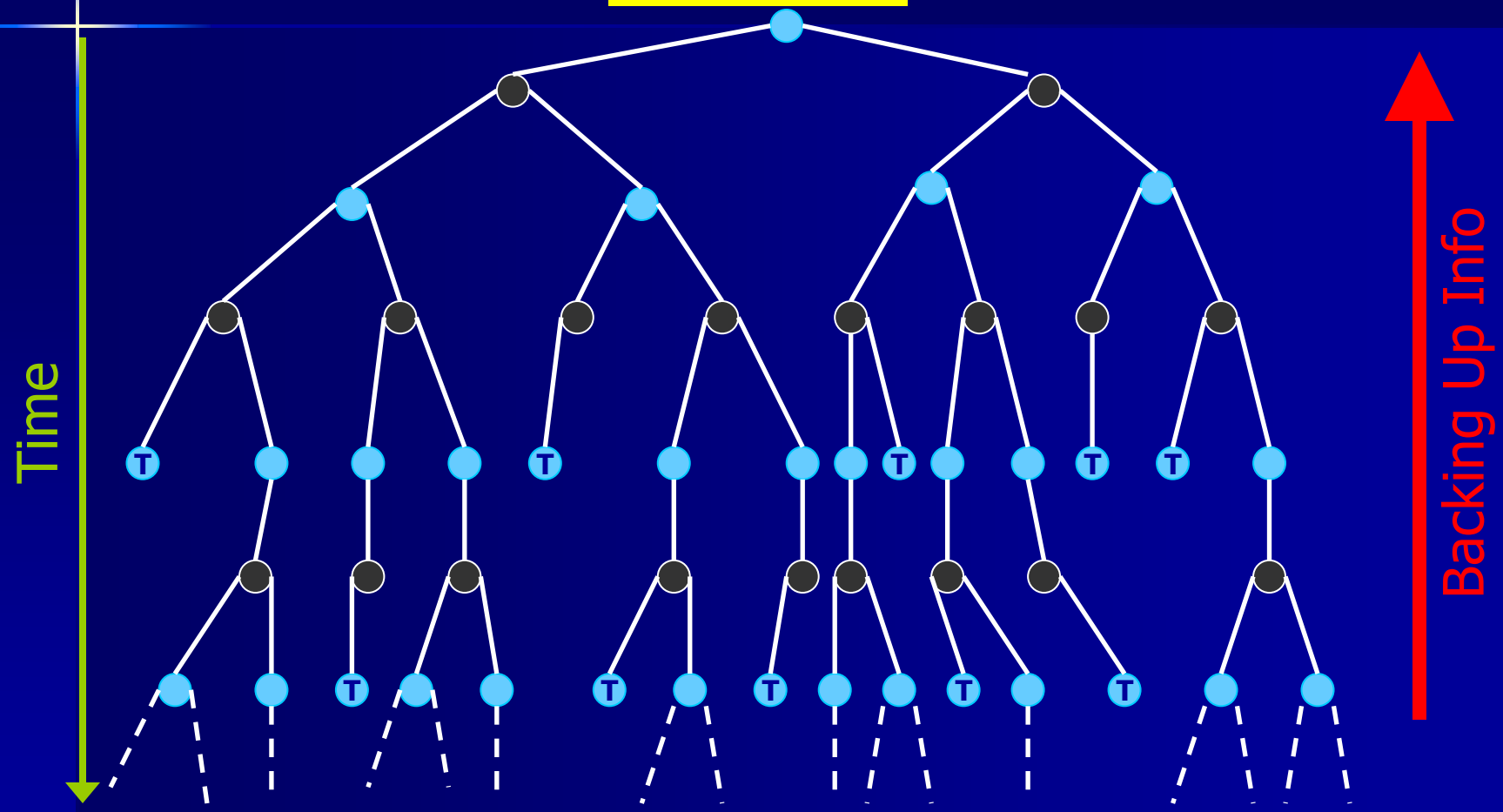
'Backups'

(from Sutton & Barto)

$U(s) \leftarrow ?$

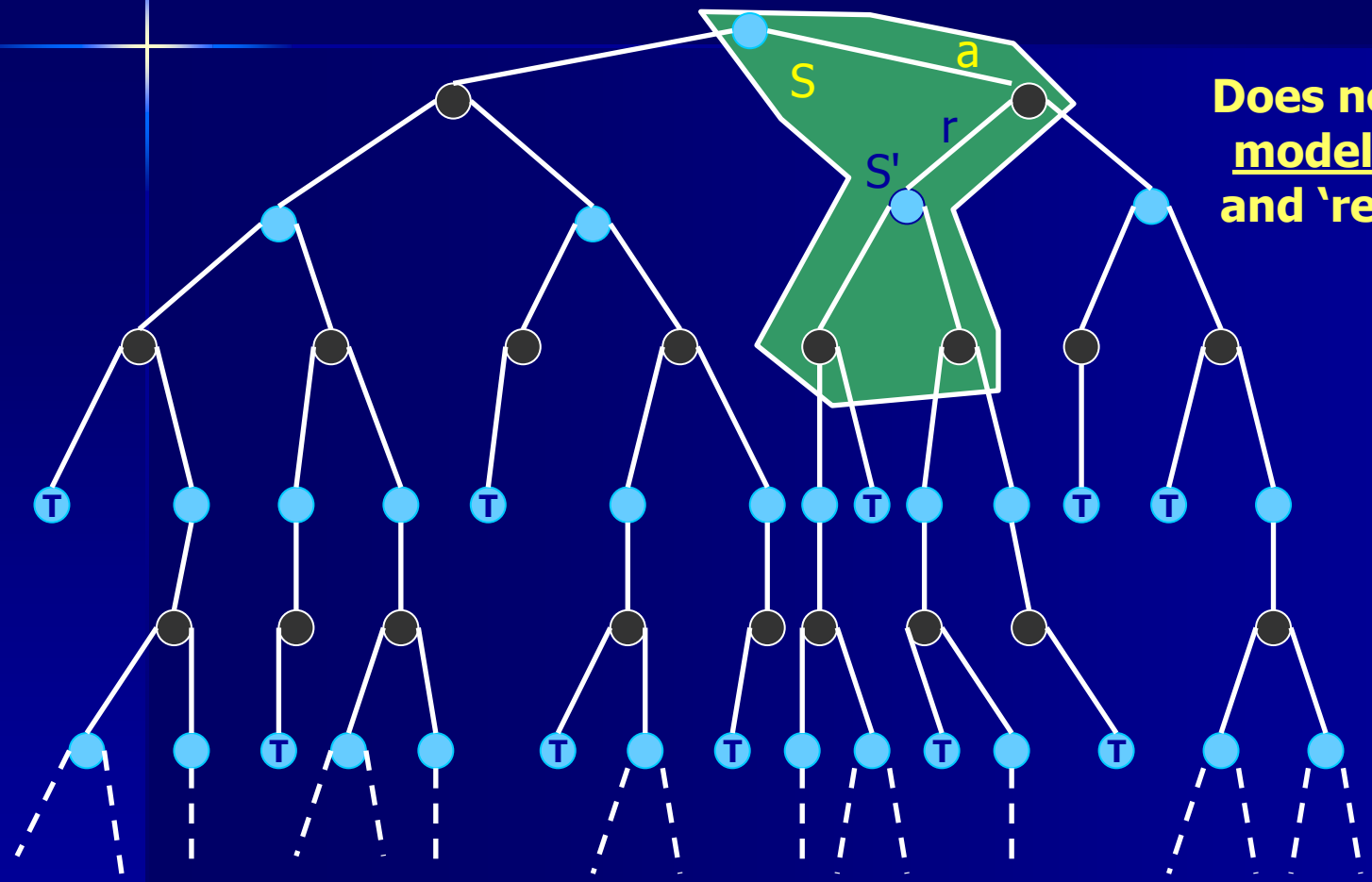
Key

- Non-deterministic action
- Possible next states
- T 'Terminal state' (end of game)



Q-Learning Backup

$$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + \alpha\, [r + \gamma \max_b Q(s',b)]$$



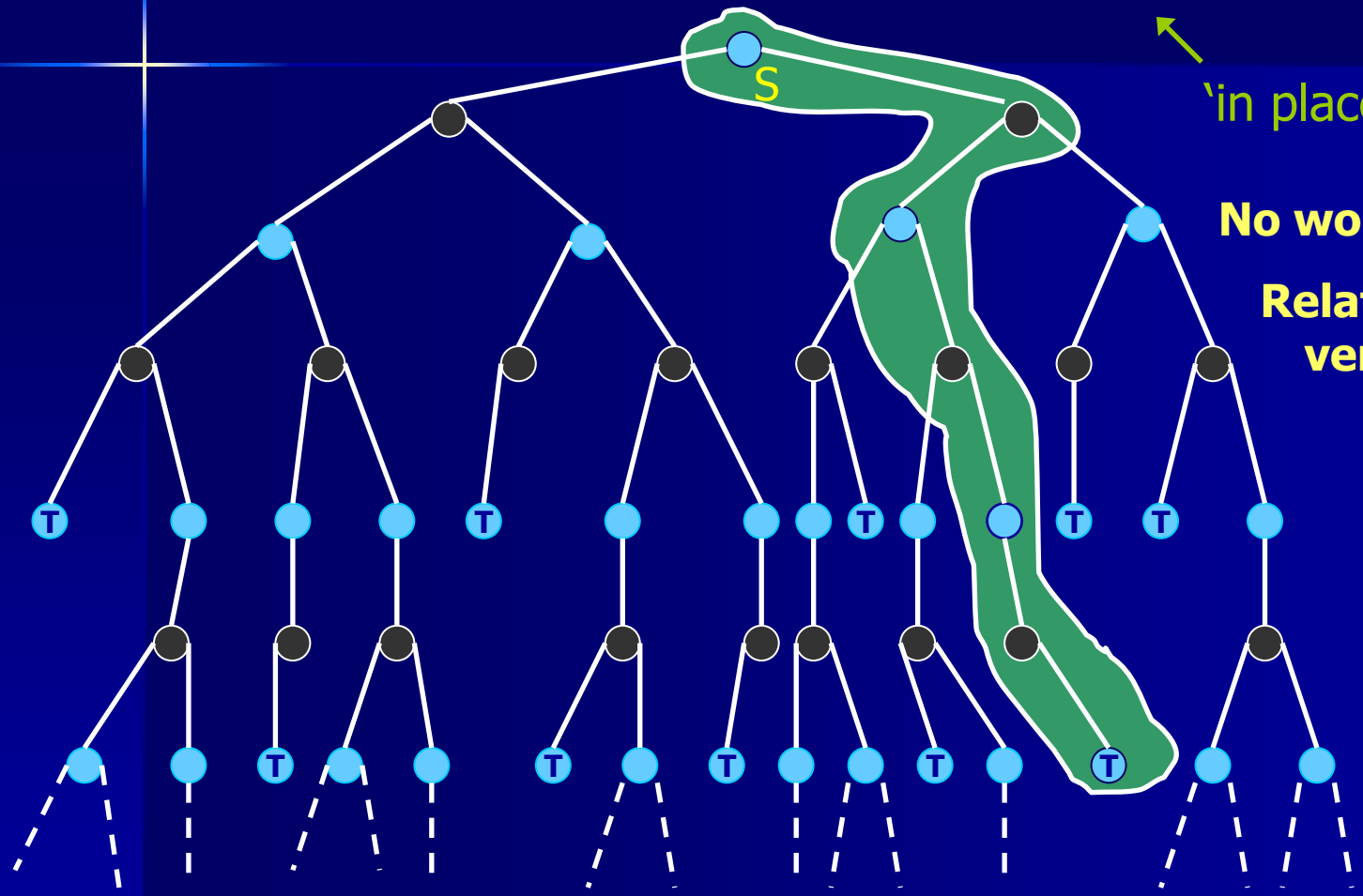
Does not need an <u>world model</u> (the 'next state' and 'reward' functions)

Simple Monte Carlo

Stochastically run to end of 'game' many times

$$U(s) \leftarrow (1-\alpha)\, U(s) + \alpha\, \text{Reward(path)}$$

'in place' averaging



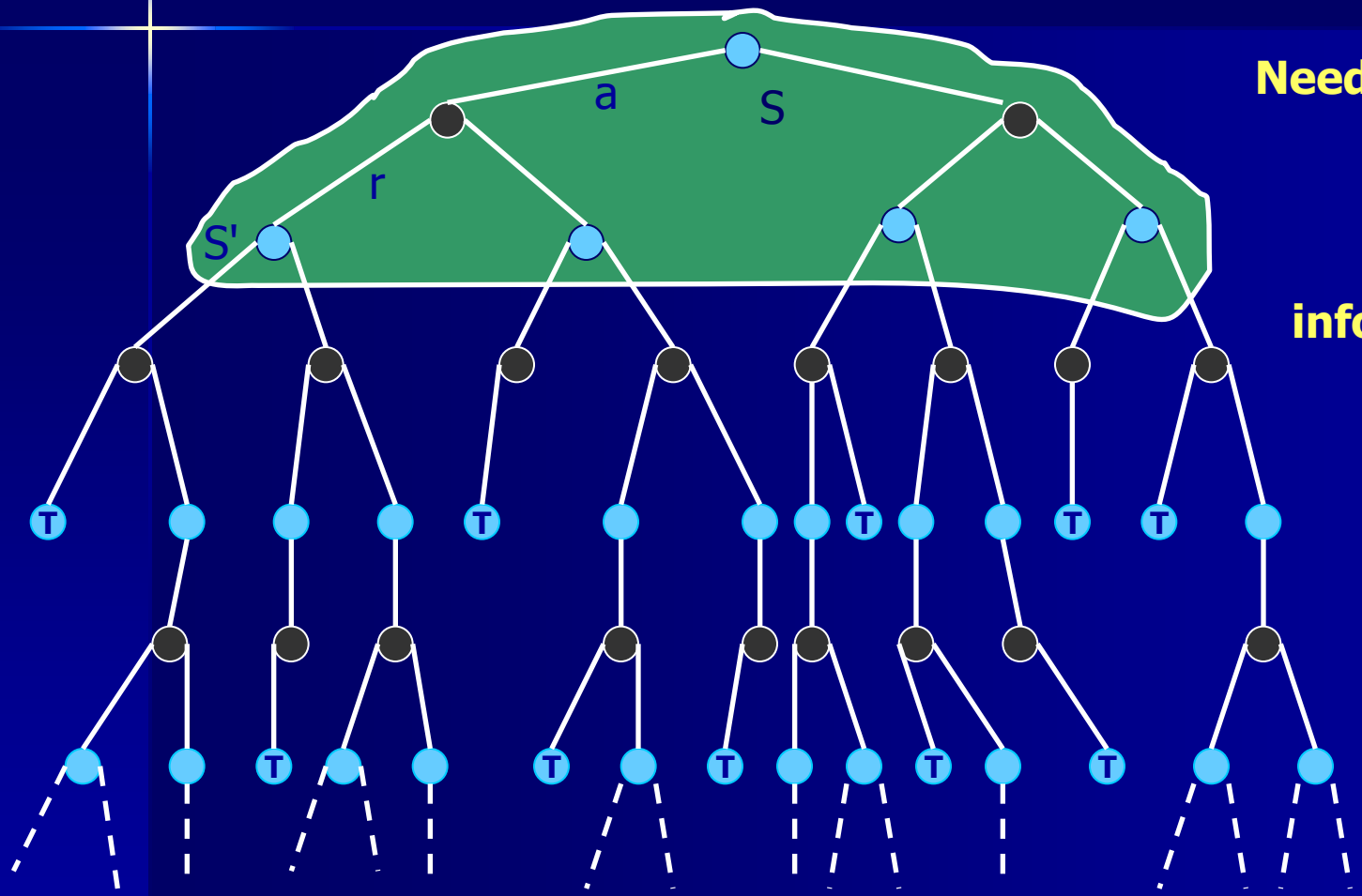
No world model needed

Relatively efficient on very large problems

Stochastic Dynamic Programming

Expected value

$$U(s) \leftarrow \max_a \mathbf{E}\,[r + \gamma\, U(s' \text{ given action } a)]$$



Needs a <u>world model</u> to compute true expected value

A very informative backup

Complex TD Backups

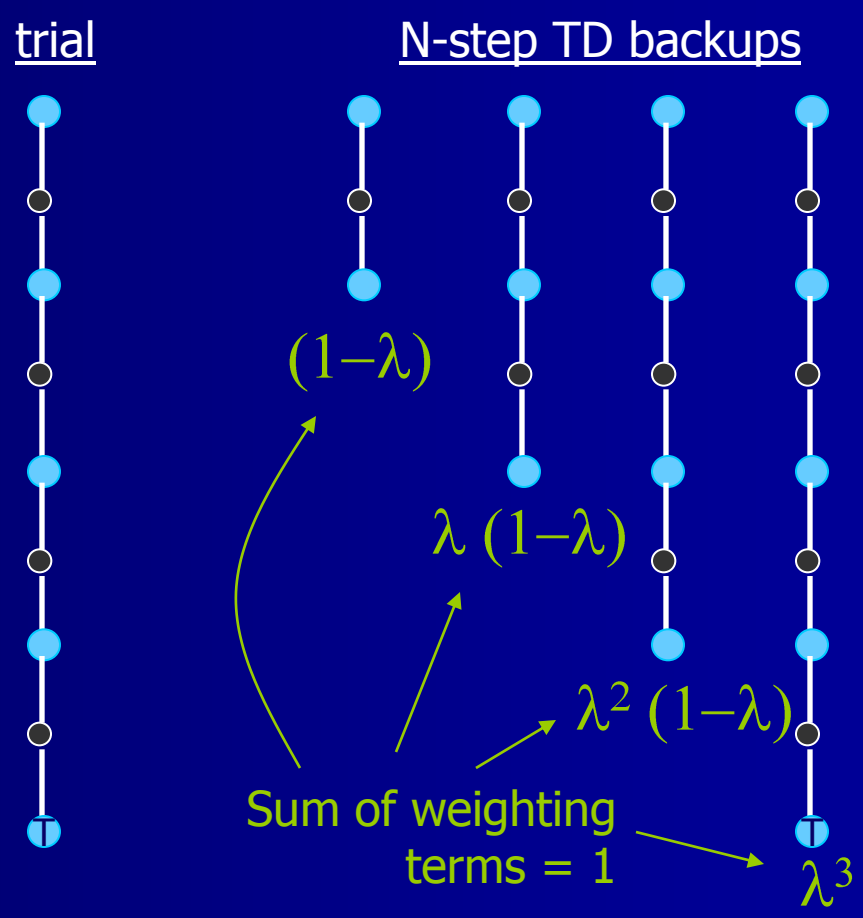
(TD = Temporal Difference; Sutton, MLj, '88)

TD(λ)

Incrementally computes a weighted mixture of these backups as states are visited

λ=1 → Simple Monte Carlo

λ=0 → 1-Step Q-Learning



The Sum of the TD(λ) Weighting Terms

$$\left[\sum_{i=0 \text{ to } N-1} (1-\lambda) \times \lambda^{i}\right] + \lambda^{N}$$

$$= (1-\lambda) \times \left[\sum_{i=0 \text{ to } N-1} \lambda^{i}\right] + \lambda^{N}$$

$$= (1-\lambda) \times \left[(1-\lambda^{N}) / (1-\lambda)\right] + \lambda^{N}$$

see http://www.math.com/tables/expansion/geom.htm

$$= (1-\lambda^{N}) + \lambda^{N}$$

$$= 1$$