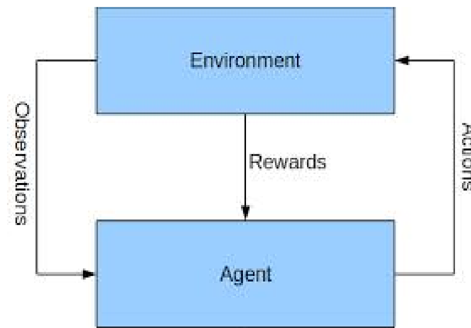
Reinforcement Learning Agents
an approach to AGI



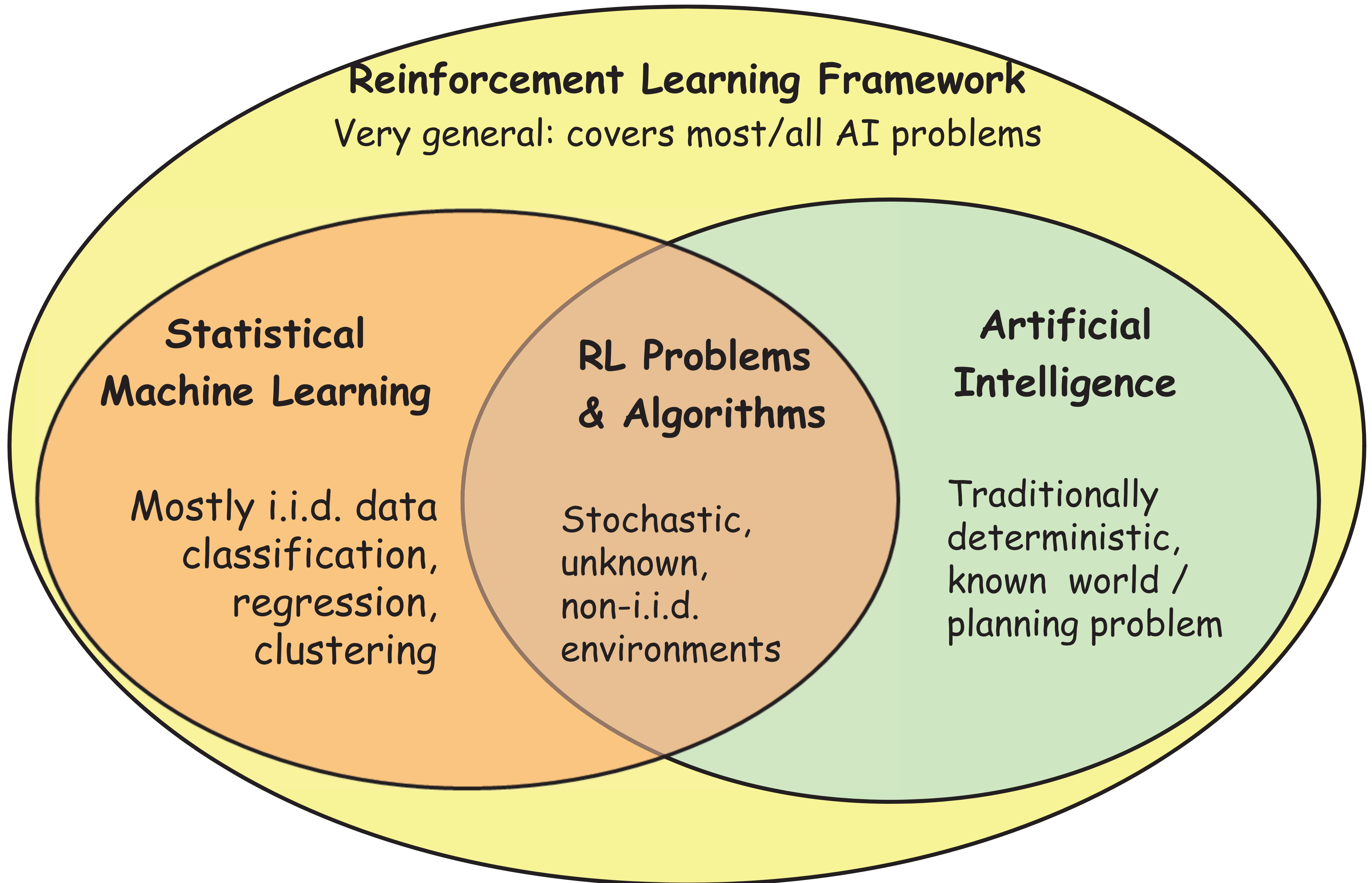
Peter Sunehag

Tutorial at the seventh conference on AGI

Quebec City August 2014

Many slides borrowed from e.g. Sutton&Barto

Relation between RL and AI



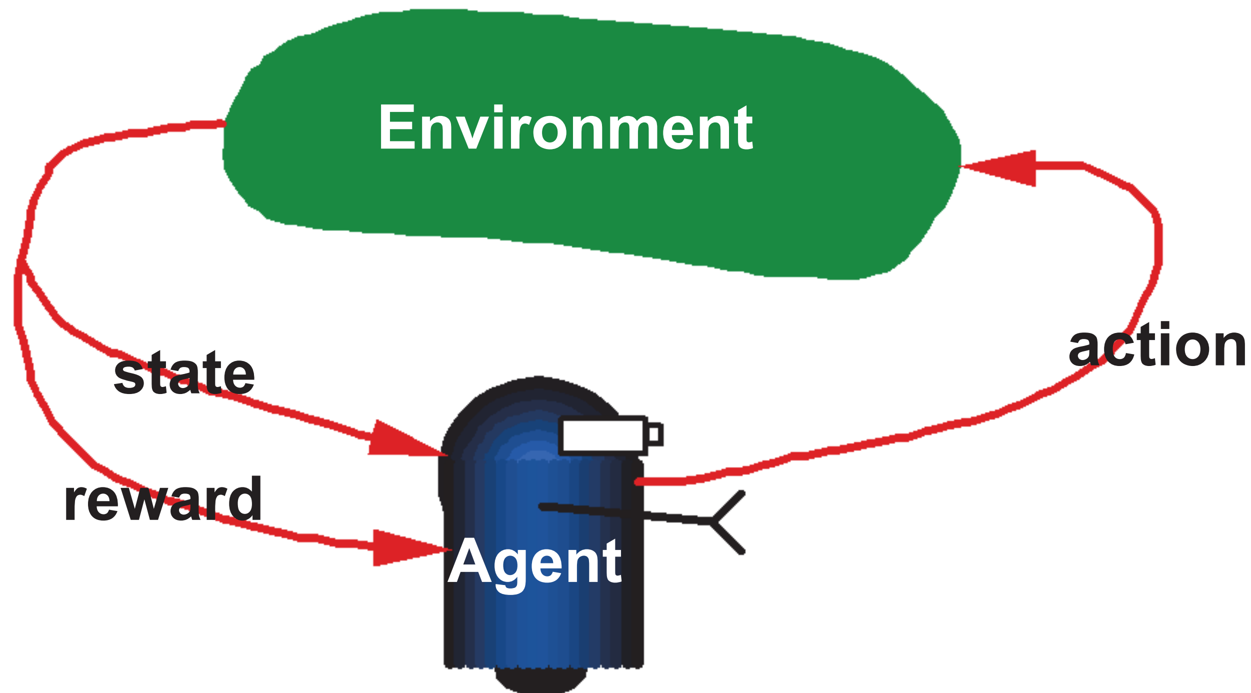
Some Applications of RL

- Checkers (Samuel, 1959)
  - first use of RL in an interesting real game
- (Inverted) helicopter flight (Ng et al. 2004)
  - better than any human
- Computer Go (UCT, 2006)
  - finally some programs with "reasonable" play
- Robocup Soccer Teams (Stone & Veloso, Reidmiller et al.)
  - World's best player of simulated soccer, 1999; Runner-up 2000
- Inventory Management (Van Roy, Bertsekas, Lee & Tsitsiklis)
  - 10-15% improvement over industry standard methods
- Dynamic Channel Assignment (Singh & Bertsekas, Nie & Haykin)
  - World's best assigner of radio channels to mobile telephone calls
- Elevator Control (Crites & Barto)
  - (Probably) world's best down-peak elevator controller
- Many Robots
  - navigation, bi-pedal walking, grasping, switching between skills...
- TD-Gammon and Jellyfish (Tesauro, Dahl)
  - World's best backgammon player. Grandmaster level

Arcade Learning Enivornment
Nadaf 2010, Bellamare et al. 2012, ... (on-going challenge)

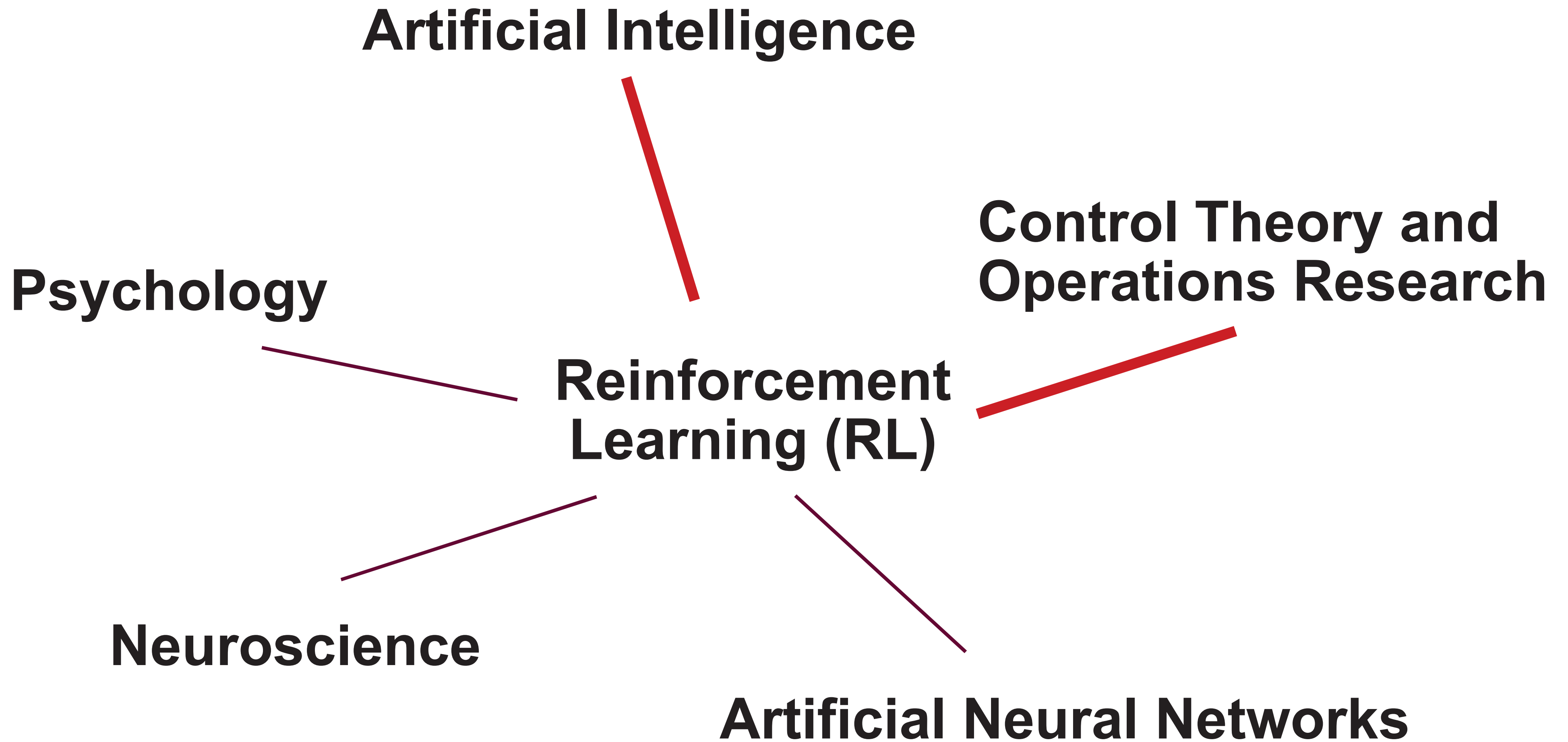
Complete Agent

- Temporally situated
- Continual learning and planning
- Object is to ***affect*** the environment
- Environment is stochastic and uncertain



What is Reinforcement Learning?

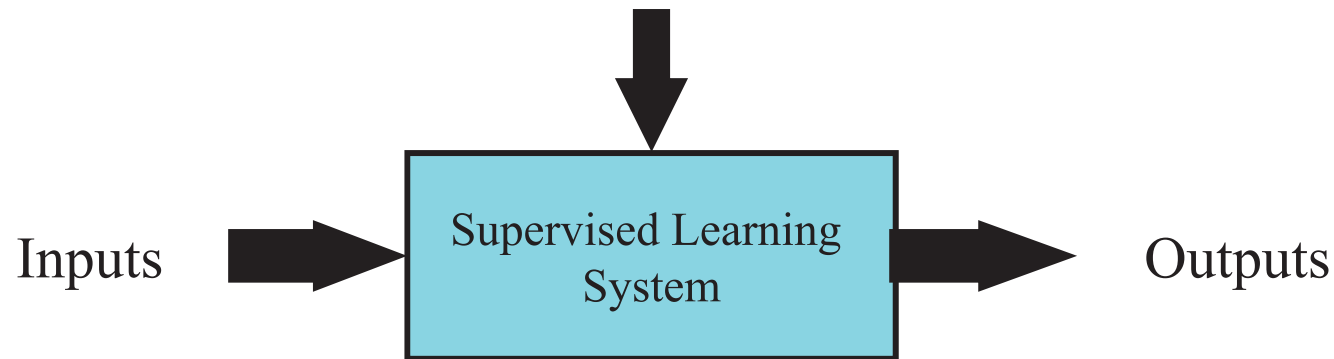
- An approach to Artificial Intelligence
- Learning from interaction
- Goal-oriented learning
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal


Artificial Intelligence
Control Theory and
Operations Research
Psychology
Reinforcement
Learning (RL)
Neuroscience
Artificial Neural Networks

Key Features of RL

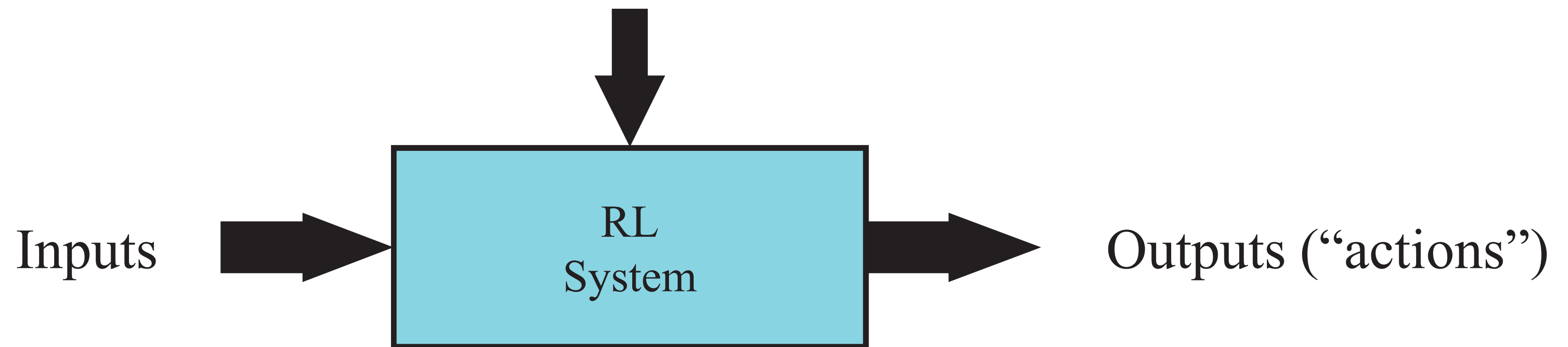
- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains
- The need to ***explore*** and ***exploit***
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

Supervised Learning



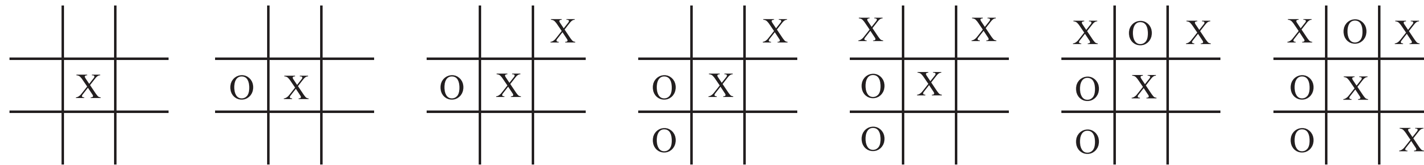
Error = (target output – actual output)

Reinforcement Learning



Objective: get as much reward as possible

An Extended Example: Tic-Tac-Toe



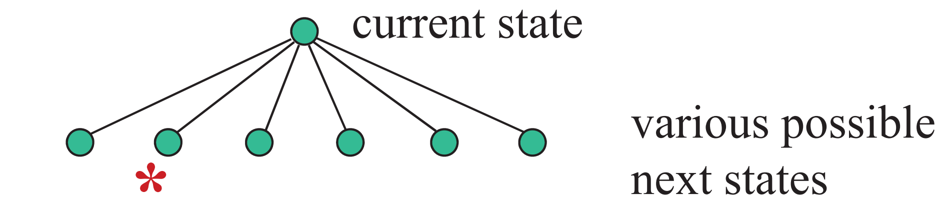
Assume an imperfect opponent:
—he/she sometimes makes mistakes

An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

State	$V(s)$ – estimated probability of winning	
(empty board)	.5	?
x at top left	.5	?
⋮	⋮	
x x x / o · · / · · o	1	win
⋮	⋮	
· x o / x · o / · · o	0	loss
⋮	⋮	
o x o / o x x / x o o	0	draw

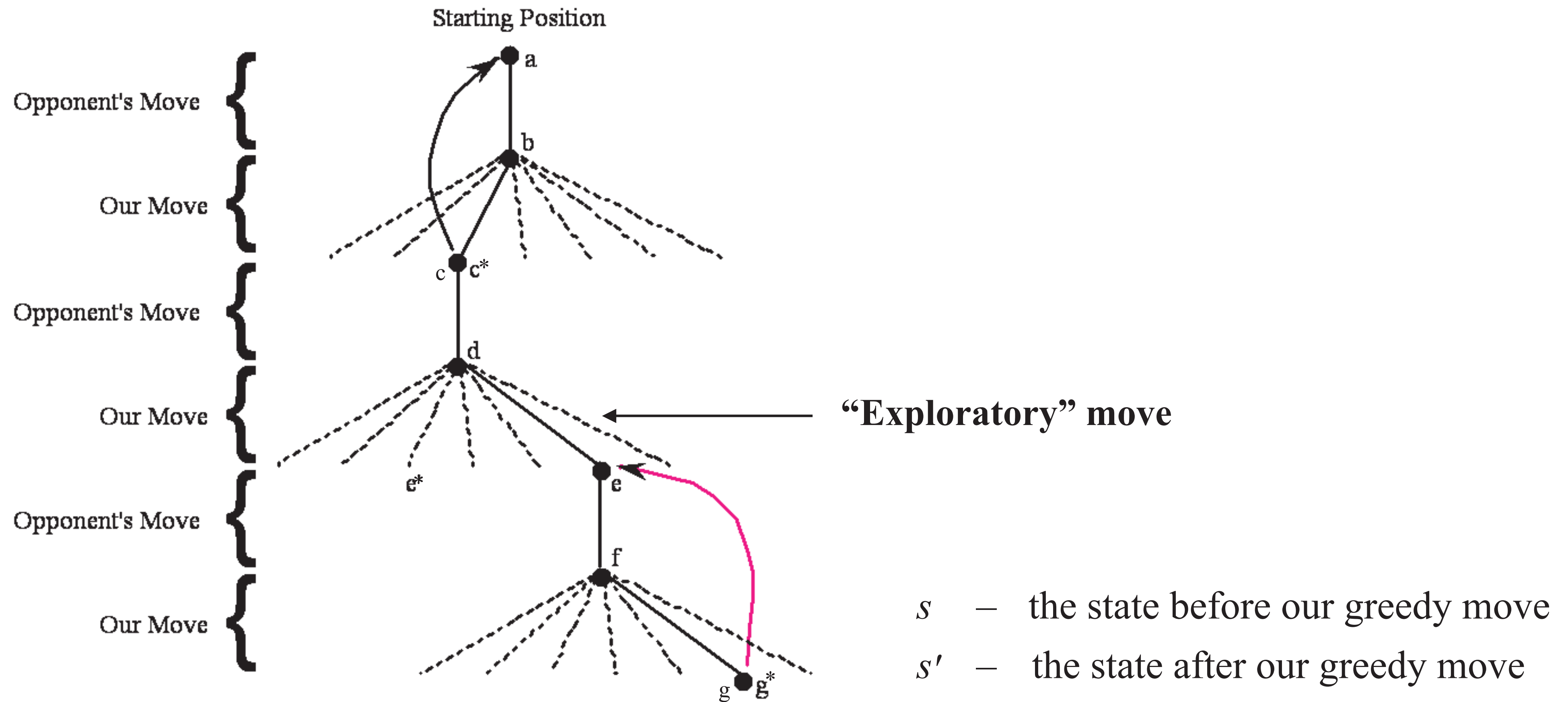
2. Now play lots of games.
To pick our moves,
look ahead one step:



Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a ***greedy*** move.

But 10% of the time pick a move at random; an ***exploratory move***.

RL Learning Rule for Tic-Tac-Toe



We increment each $V(s)$ toward $V(s')$ – a ***backup***:

$$V(s) \leftarrow V(s) + \alpha\left[V(s') - V(s)\right]$$

a small positive fraction, e.g., $\alpha = .1$

the ***step-size parameter***

The *n*-Armed Bandit Problem

- Choose repeatedly from one of *n* actions; each choice is called a **play**
- After each play a_t, you get a reward r_t, where

$$E\{r_t \mid a_t\} = Q^*(a_t)$$

These are unknown ***action values***
Distribution of r_t depends only on a_t

- Objective is to maximize the reward in the long term, e.g., over 1000 plays

To solve the *n*-armed bandit problem,
you must **explore** a variety of actions
and **exploit** the best of them

The Exploration/Exploitation Dilemma

- Suppose you form estimates

$$Q_t(a) \approx Q^*(a)$$ **action value estimates**

- The **greedy** action at t is a_t

$$a_t^* = \operatorname*{argmax}_a Q_t(a)$$

$$a_t = a_t^* \Rightarrow \text{exploitation}$$

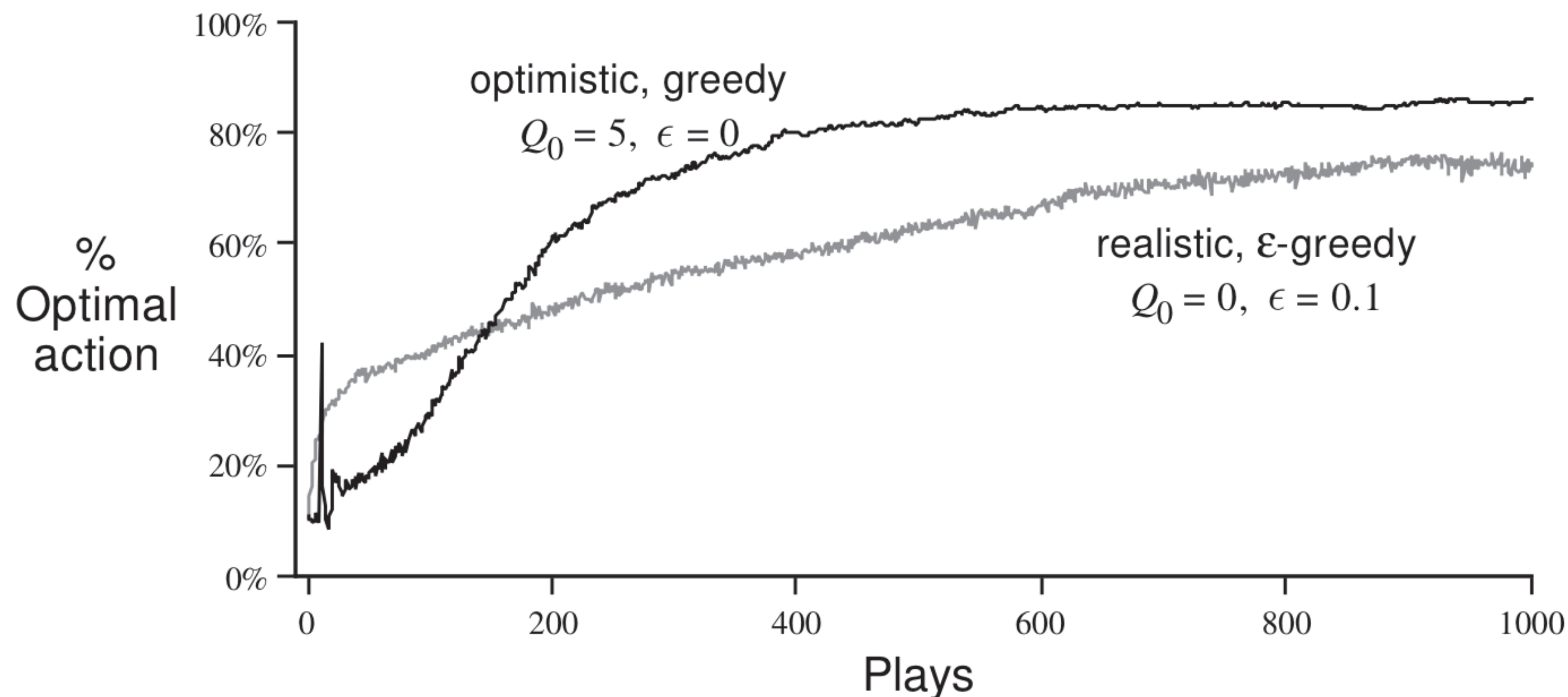
$$a_t \neq a_t^* \Rightarrow \text{exploration}$$

- You can't exploit all the time; you can't explore all the time
- You can never stop exploring; but you should always reduce exploring. Maybe.

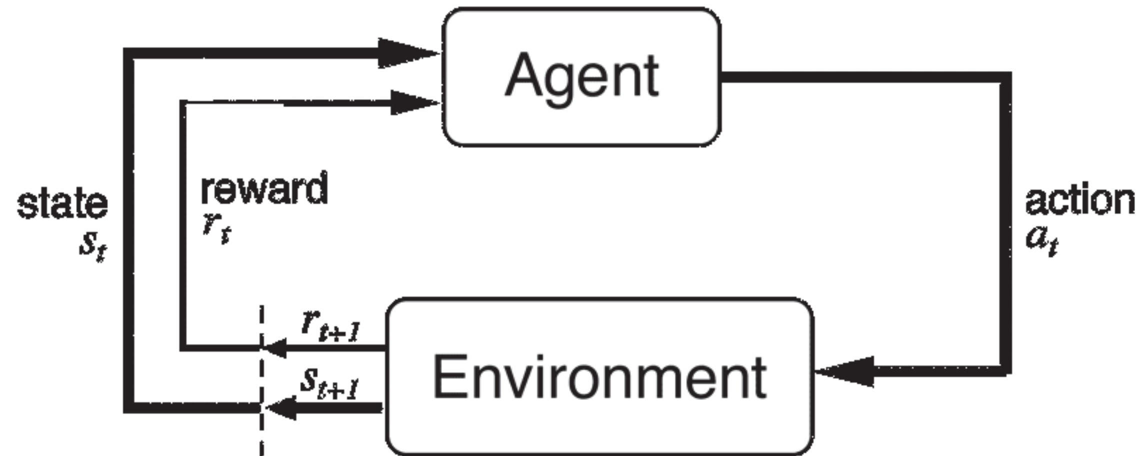
Optimistic Initial Values

- All methods so far depend on $Q_0(a)$, i.e., they are **biased.**
- Suppose instead we initialize the action values optimistically,

  i.e., on the 10-armed testbed, use $Q_0(a) = 5$ for all a



The Agent-Environment Interface



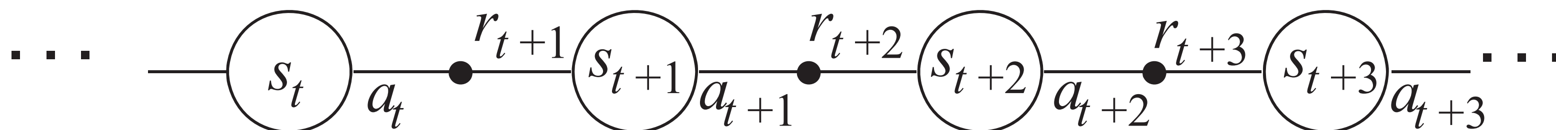
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: s_{t+1}



Bellman Equation for a Policy π

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

So:

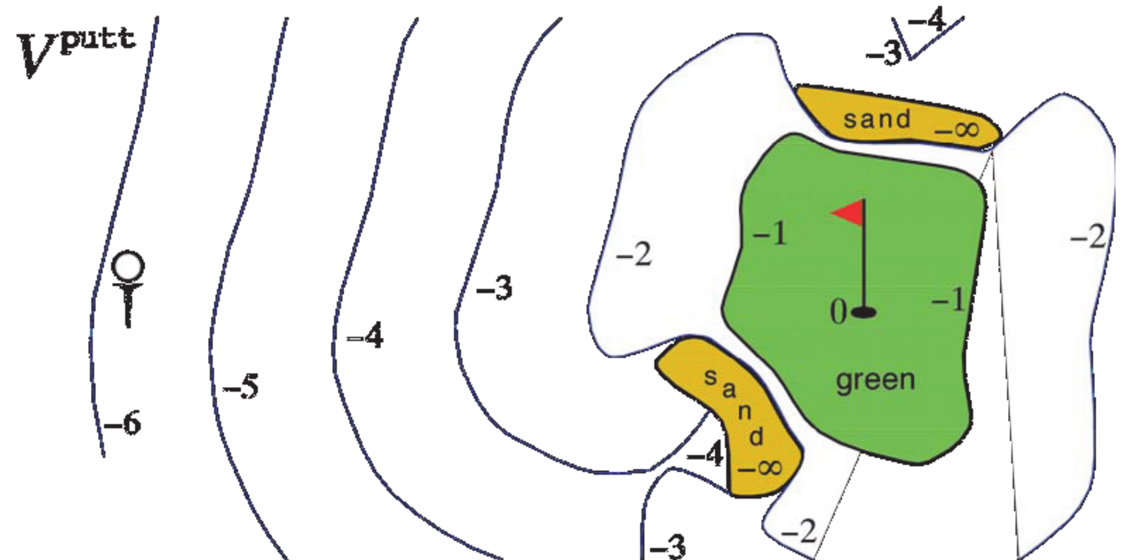
$$\begin{aligned} V^\pi(s) &= E_\pi \left\{ R_t \middle| s_t = s \right\} \\ &= E_\pi \left\{ r_{t+1} + \gamma V\left(s_{t+1}\right) \middle| s_t = s \right\} \end{aligned}$$

Or, without the expectation operator:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathbf{P}_{ss'}^a \left[\mathbf{R}_{ss'}^a + \gamma V^\pi(s') \right]$$

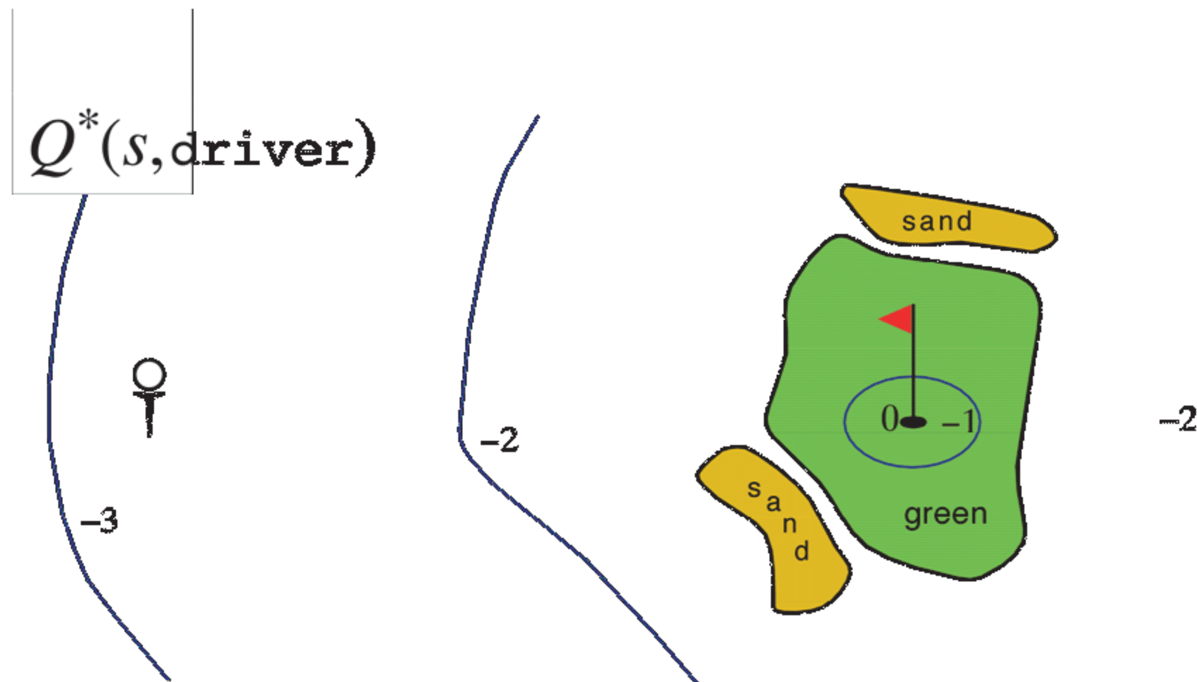
Golf

- State is ball location
- Reward of –1 for each stroke until the ball is in the hole
- Value of a state?
- Actions:
  - `putt` (use putter)
  - `driver` (use driver)
- `putt` succeeds anywhere on the green



Optimal Value Function for Golf

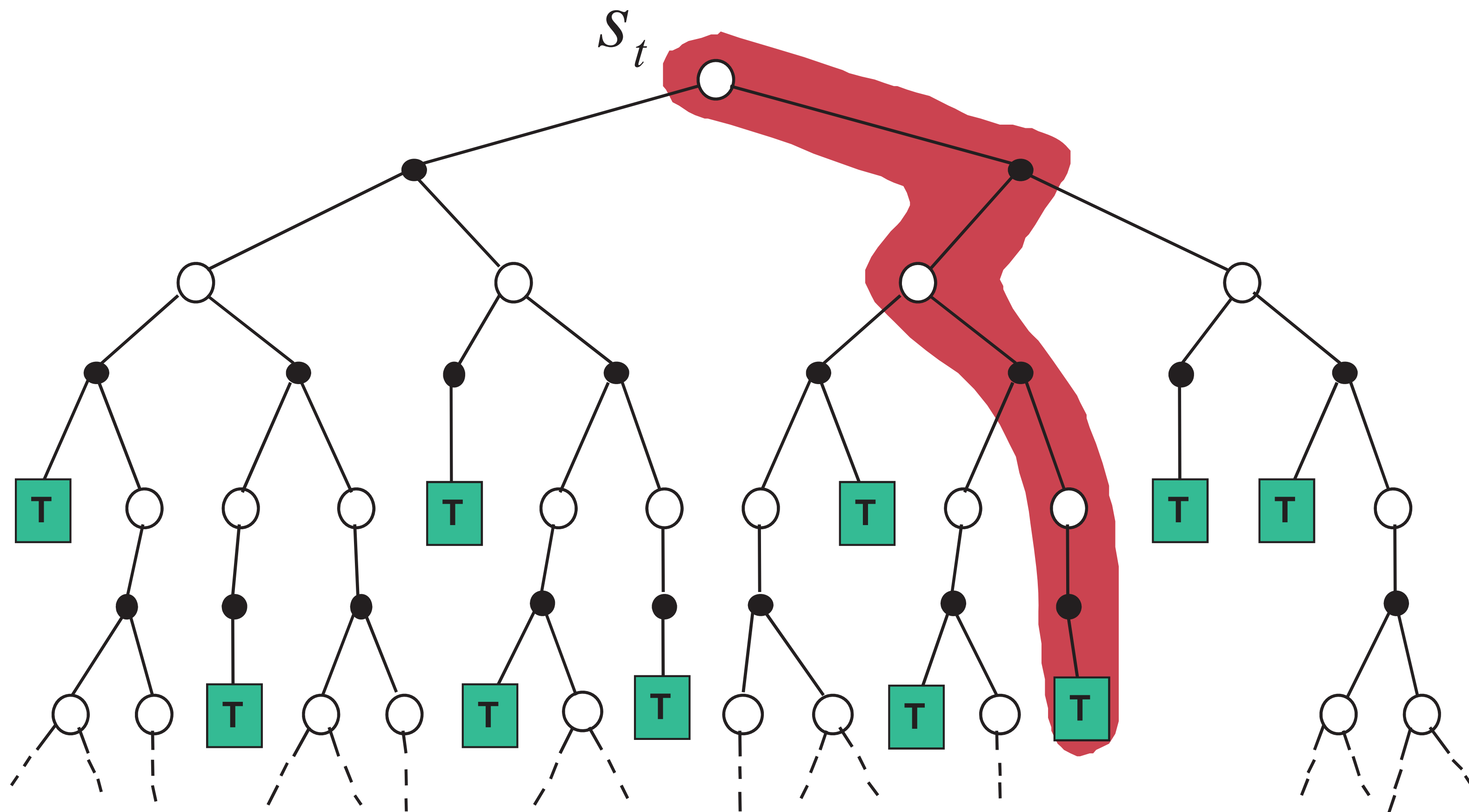
- We can hit the ball farther with `driver` than with `putter`, but with less accuracy
- $Q^*(s,\texttt{driver})$ gives the value of using `driver` first, then using whichever actions are best



Simple Monte Carlo

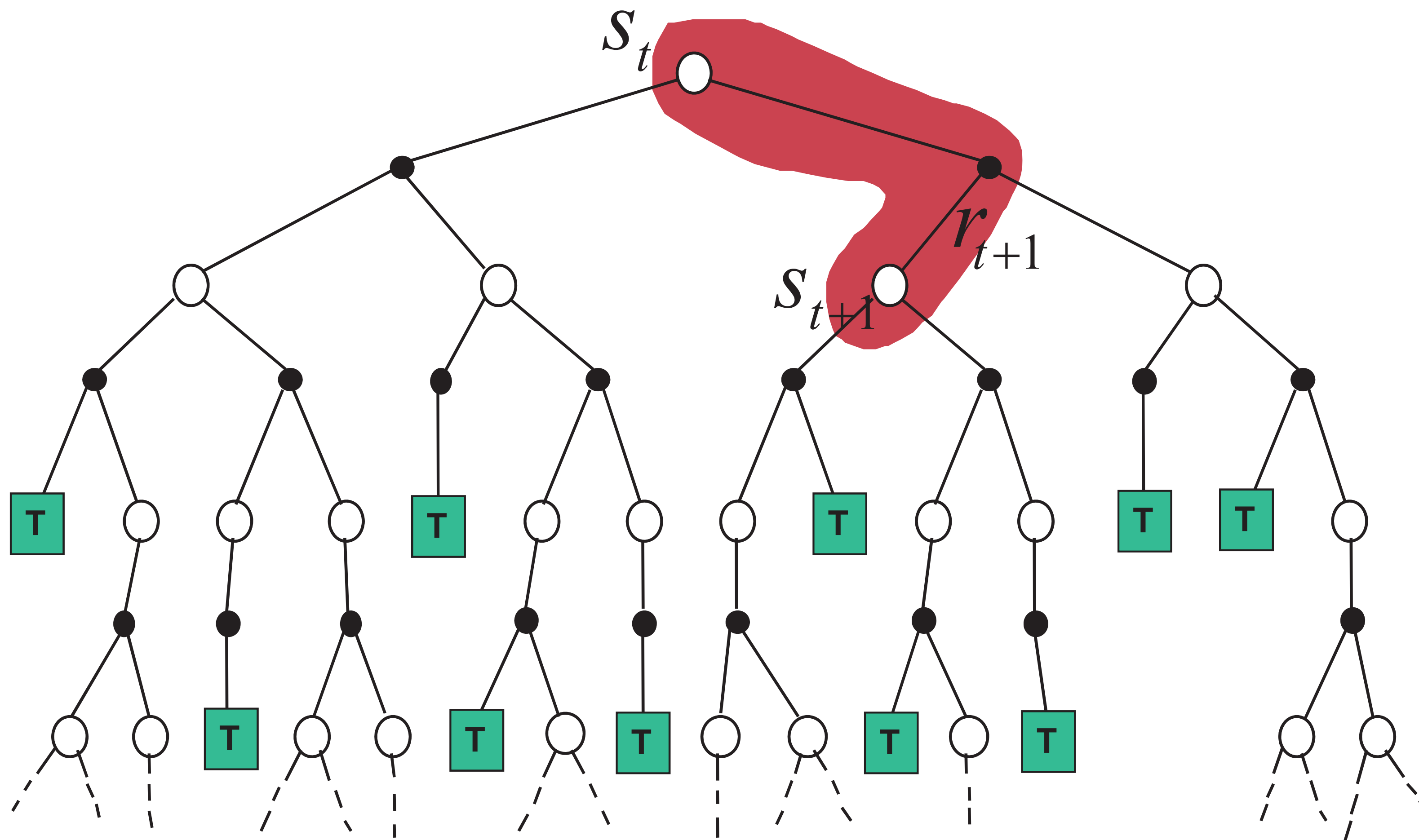
$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

where R_t is the actual return following state s_t.



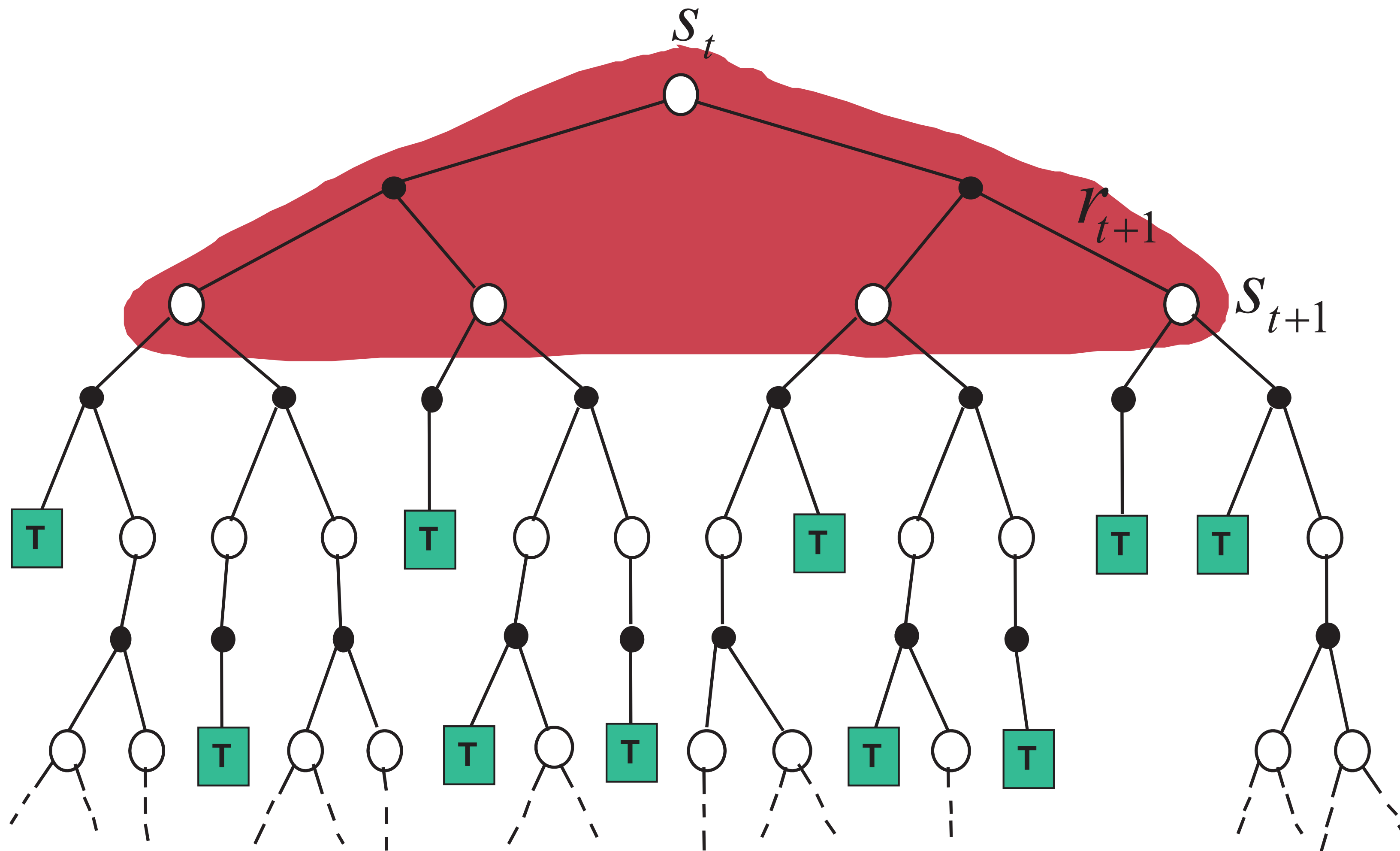
Simplest TD Method

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



Dynamic Programming

$$V(s_t) \leftarrow E_\pi \left\{ r_{t+1} + \gamma V(s_{t+1}) \mid s_t \right\}$$



TD methods bootstrap and sample

- Bootstrapping: update involves an *estimate*
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update does not involve an *expected value*
  - MC samples
  - DP does not sample
  - TD samples

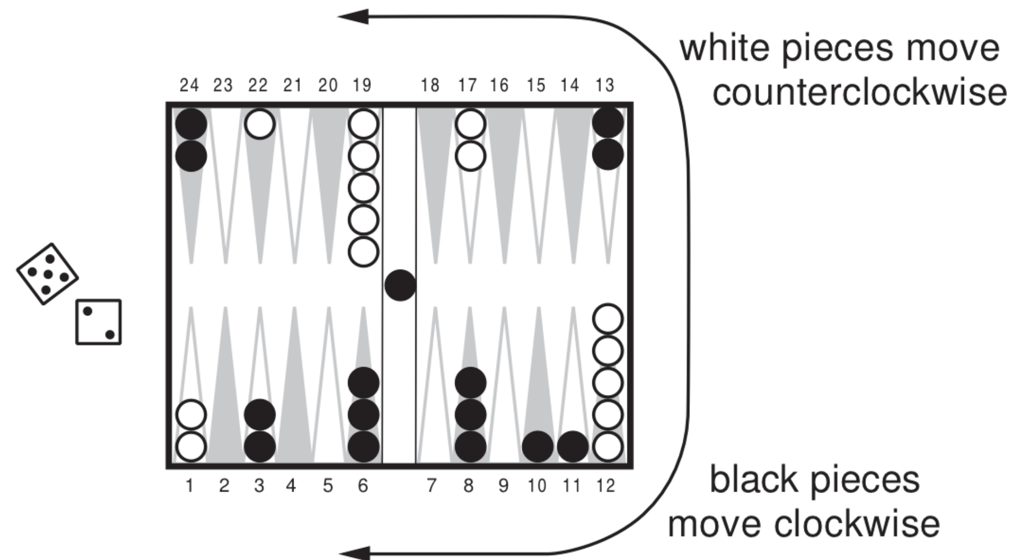
Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- 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?

TD Gammon

Tesauro 1992, 1994, 1995, ...

- White has just rolled a 5 and a 2 so can move one of his pieces 5 and one (possibly the same) 2 steps
- Objective is to advance all pieces to points 19-24
- Hitting
- Doubling
- 30 pieces, 24 locations implies enormous number of configurations
- Effective branching factor of 400



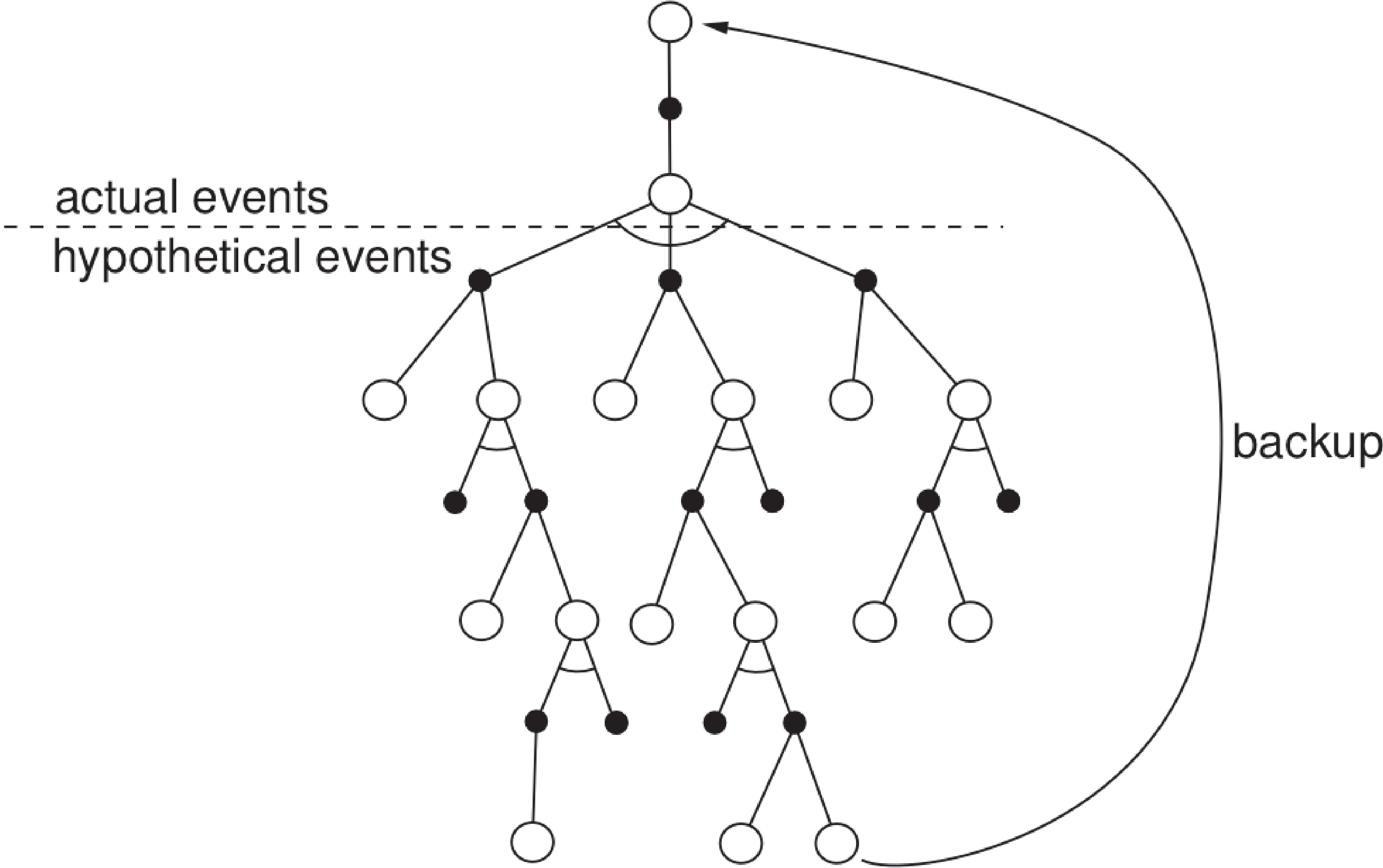
TD Gammon combined TD(λ) with neural network as value function approximator

Samuel's Checkers Player

Arthur Samuel 1959, 1967

- Score board configurations by a "scoring polynomial" (after Shannon, 1950)
- Minimax to determine "backed-up score" of a position
- Alpha-beta cutoffs
- Rote learning: save each board config encountered together with backed-up score
  - needed a "sense of direction": like discounting
- Learning by generalization: similar to TD algorithm with linear value function approximation.

Samuel's Backups



The Basic Idea

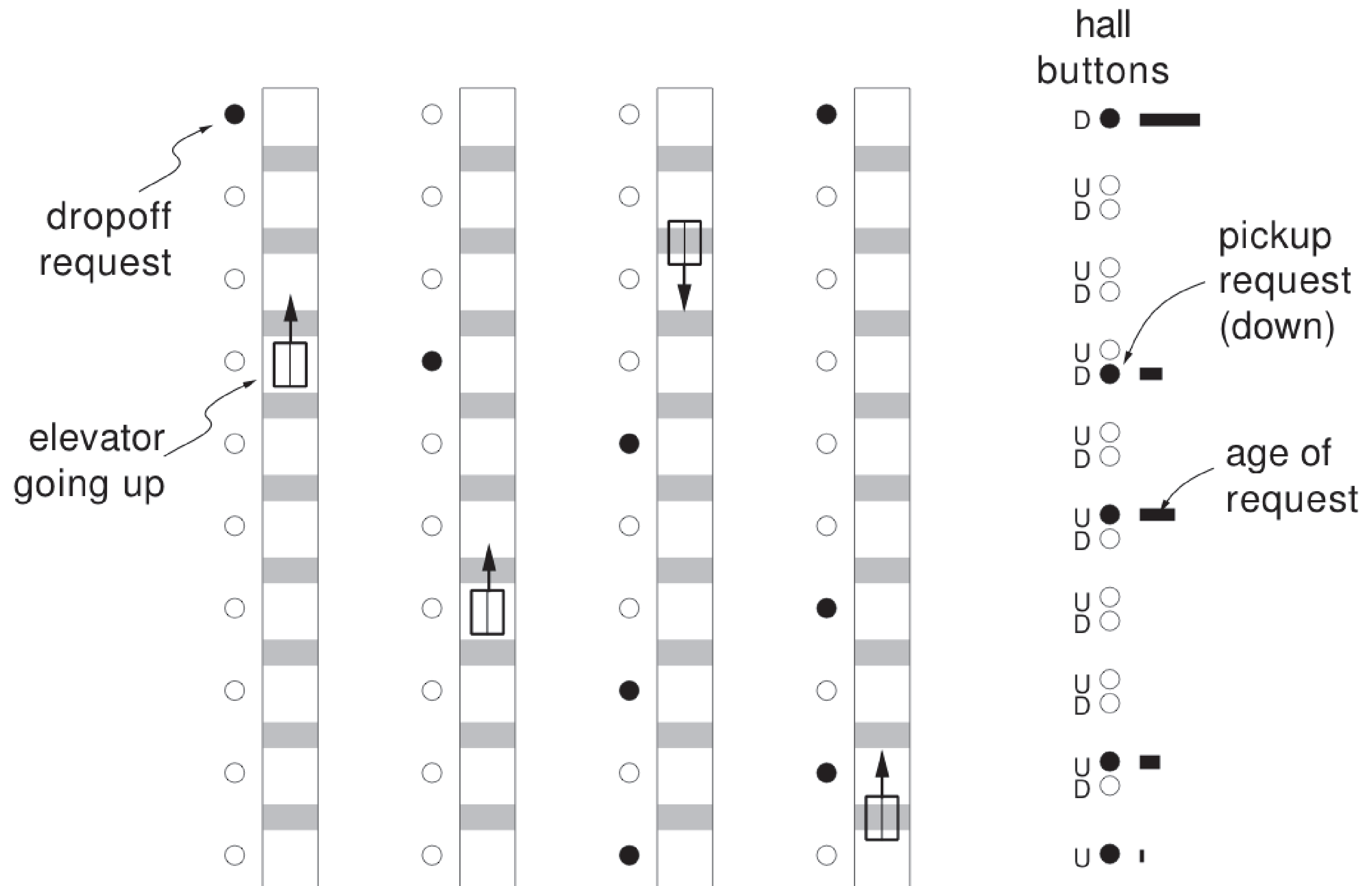
". . . we are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board positions of the chain of moves which most probably occur during actual play."

A. L. Samuel

***Some Studies in Machine Learning Using the Game of Checkers*, 1959**

Elevator Dispatching

Crites and Barto 1996



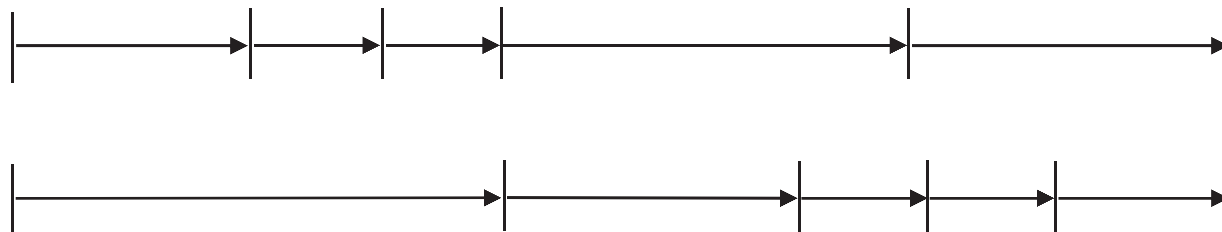
State Space

- 18 hall call buttons: 2^{18} combinations
- positions and directions of cars: 18^4 (rounding to nearest floor)
- motion states of cars (accelerating, moving, decelerating, stopped, loading, turning): 6
- 40 car buttons: 2^{40}
- Set of passengers waiting at each floor, each passenger's arrival time and destination: unobservable. However, 18 real numbers are available giving elapsed time since hall buttons pushed; we discretize these.
- Set of passengers riding each car and their destinations: observable only through the car buttons

Conservatively about 10^{22} states

Actions

- When moving (halfway between floors):
  - stop at next floor
  - continue past next floor
- When stopped at a floor:
  - go up
  - go down
- Asynchronous



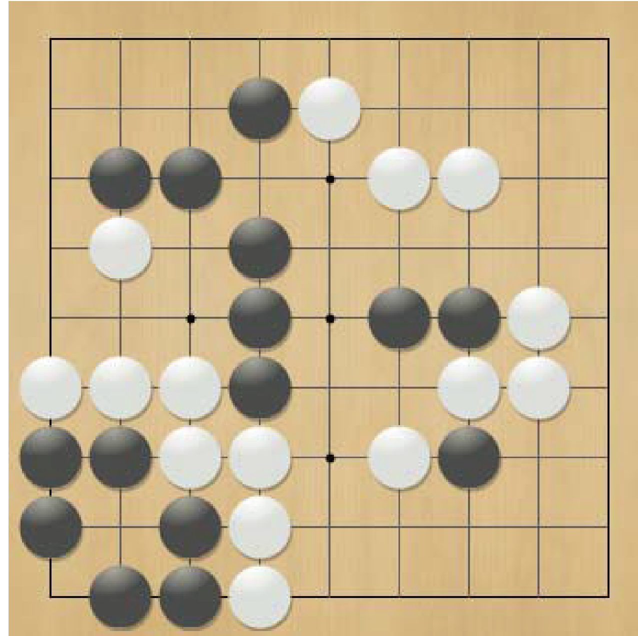
Performance Criteria

Minimize:

- Average wait time
- Average system time (wait + travel time)
- % waiting > T seconds (e.g., T = 60)

- Average squared wait time (to encourage fast and fair service)

Computer Go

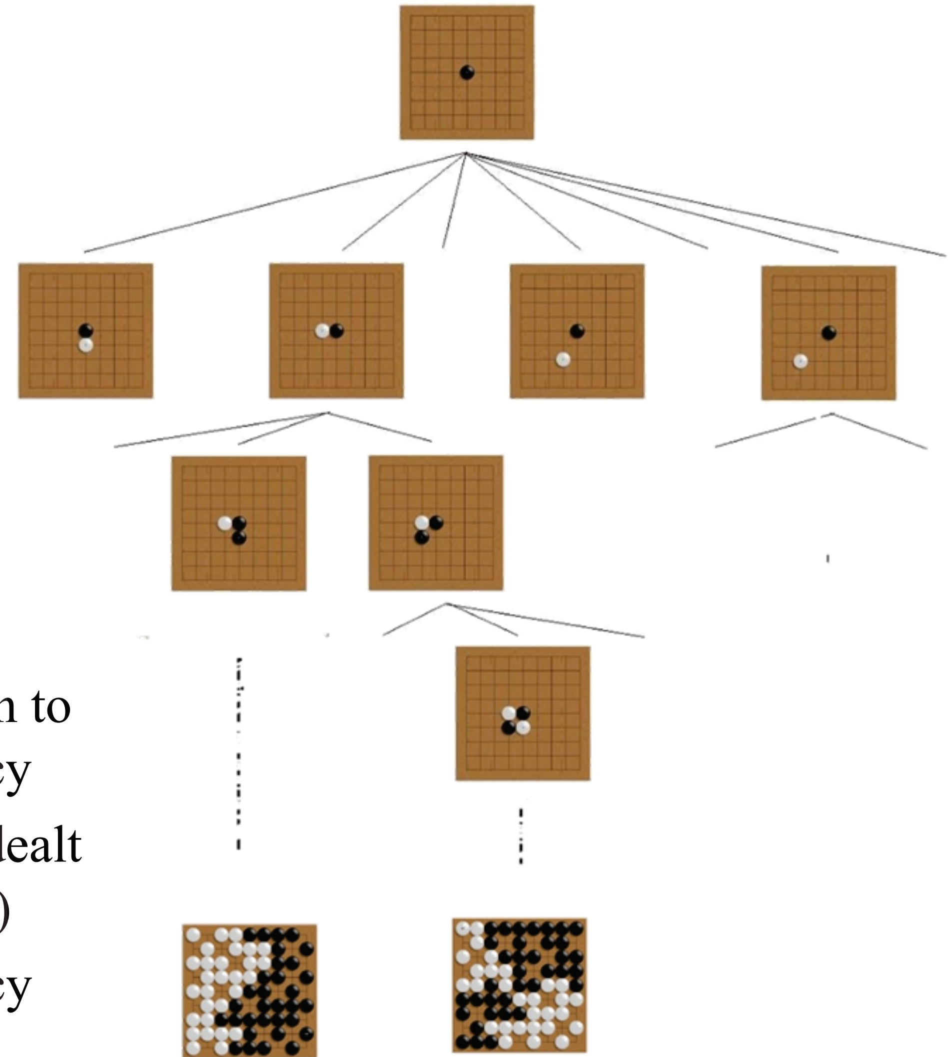


- “Task Par Excellence for AI” (Hans Berliner)
- “New Drosophila of AI” (John McCarthy)
- “Grand Challenge Task” (David Mechner)

Sylvain Gelly (2004)

Monte Carlo Tree Search + Playout Policy

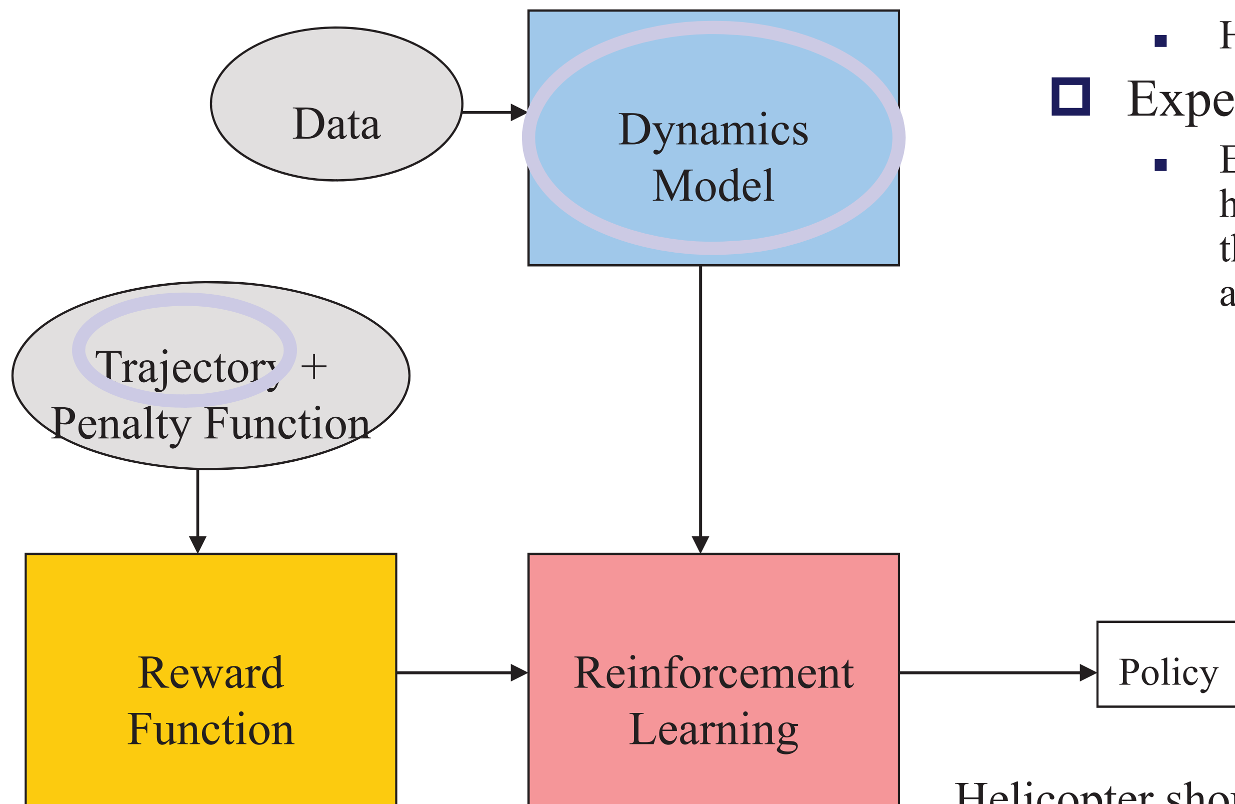
- DP & TD do not apply:
  - State space too large
- MC & ES do not apply:
  - Action space too large
- New Approach
  - In the current state sample and learn to construct a locally specialized policy
  - Exploration/exploitation dilemma dealt with Upper Confidence Tree (UCT)
  - Evaluate leaves using Playout Policy



Inverted Helicopter Flight

Andrew Ng et al. (2004-2008)

RL learns policy better than any human



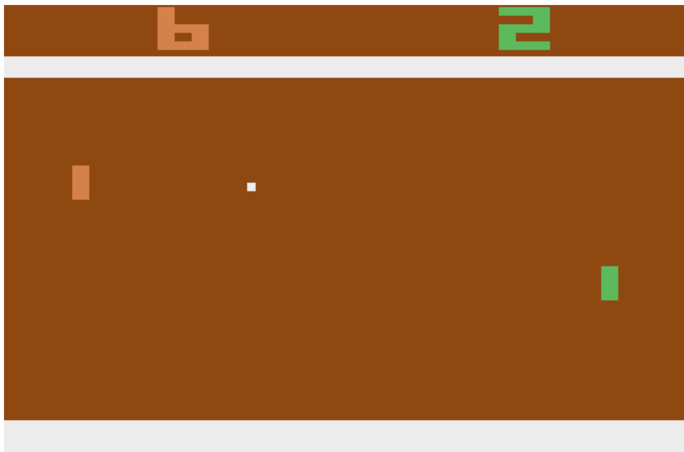
- Generative model for multiple suboptimal demonstrations.
- Learning algorithm that extracts:
  - Intended trajectory
  - High-accuracy dynamics model
- Experimental results:
  - Enabled them to fly autonomous helicopter aerobatics well beyond the capabilities of any other autonomous helicopter.

Helicopter should follow a desired trajectory.

http://heli.stanford.edu/

The Arcade Learning Environment (ALE)

ALE (Nadaf 2010, Bellamare et. al. 2012) is an interface built upon the open-source Atari 2600 emulator Stella. It provides a convenient interface to ATARI 2600 games.



Features for ALE

- Basic Abstraction of Screen Shots (BASS, from Nadaf 2010) first stores a background of the game it's playing. Then for every frame it subtracts away the background and divides the screen into 16x14 tiles. For each colour (8-bit SECAM) it creates a feature. It then takes the pairwise interaction of all these resulting features resulting in 1,606,528 features.
- Color provides object recognition.
- We study linear function approximation with BASS. We want to see how well one can do with that if one finds the right parameters
- Improved results has been achieved with non-linear neural/deep approaches.

The gap

Table : The gap between score (more is better) achieved by (linear) learning and (uct) planning

Game	UCT	BestLearner
Beam Rider	6,624.6	929.4
Seaquest	5,132.4	288
Space Invaders	2,718	250.1
Pong	21	-19

Out of 55 games, UCT has the best performance on 45 of them.
The remaining games require a terribly long horizon.

Learning from an oracle

- Reinforcement learning is made much more difficult than supervised learning due to the need to explore.
- Therefore, many authors has in recent years been developing ways of teaching an rl agent through e.g. demonstration or advice with reduction to supervised learning.
- I will here discuss this idea in the context of Atari games through the Arcade Learning Environment (ALE) framework

Learning from UCT

A common scenario when applying reinforcement learning algorithms in real-world situations, learn in a simulator, apply in the real-world.

- UCT in the "real-world" still requires the simulator.
- UCT does not provide a policy representation, merely a trajectory.
- How do you extract a complete explicit policy from UCT?
- We will treat the value estimates from UCT as advice provided to the agent and we can then learn to play Pong with just a few episodes of data.
- Learning the value function is now a regression problem we solve using LibLinear (also exploring kernels, brings us back to feature selection/sparsification)
- Similar to the Dataset Aggregation algorithm for imitation learning (Ross and Bagnell 2010)

DAgger for reinforcement learning with advice Initialise $D \leftarrow \emptyset$
Initialise $\pi_1 (= \pi^*)$ t = 0 **for** *i = 1 to N* **do**

```
    while not end of episode do
        for each action a do
            Obtain feature φ(s_t, a) and oracle's Q*(s_t, a)
            Add training sample {(φ(s_t, a), Q*(s_t, a))} to D_a.
        end
        Act according to π_i
    end
    for each action a do
        Learn new model Q̂_i^a := w_i^a⊤ φ from D_a using regression
    end
    π_i(φ) = arg max_a Q̂_i^a(φ)
end
```

Preliminary results

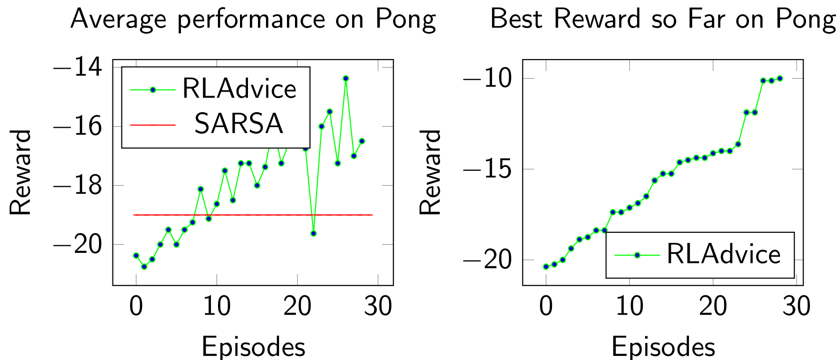
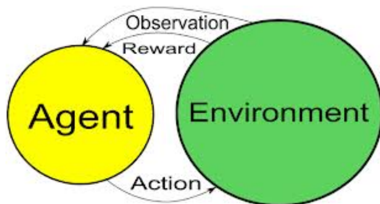


Figure : Pong Results: RLadvice with different amount of aggregated data (1-30 games) vs SARSA (linear function approximation) after 5000 games played. Results averaged over 8 runs

Partially Observable Markov Decision Processes

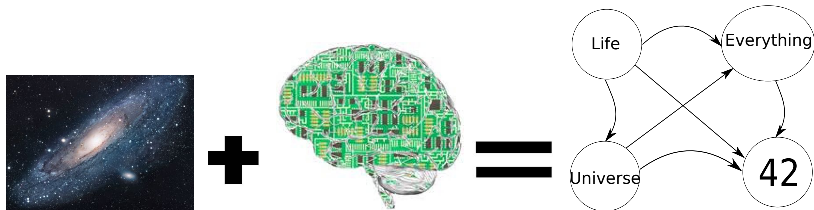
- Exacerbated Exploration issues due to not knowing the state.
- Not knowing the underlying state space makes things worse
- Explicit or implicit restrictions to subclasses
- Recent work (feature rl) on history based methods that learns a map from histories to some finite/compact state space (other alternative PSRs)
- Model-free version becomes feature selection for high-dim rl

General Reinforcement Learning



- Classes of completely general reinforcement learning environments. Beyond hopeless
- No finite underlying state space assumed. Theoretical work exist with some further assumption exist
- Bayesian (computable env.), AIXI (Hutter 2005)
- Finite/compact classes; Optimistic agents (Sunehag&Hutter. 2012), Max. exploration (Lattimore&Hutter&Sunehag 2013)

Feature Reinforcement Learning



Feature RL aims to automatically reduce a complex real-world non-Markovian problem to a useful (computationally tractable) representation (MDP).

Formally we create a map ϕ from an agent's history to a state representation. ϕ is then a function that produces a relevant summary of the history.

$$\phi(h_t) = s_t$$

Feature Markov Decision Process (ΦMDP)

To select the best ϕ, one defines a cost function.

$$\phi_{best} = \arg\min_\phi(Cost(\phi)).$$

- Feature RL is a recent framework.
- Original cost from Hutter 2009 is a model-based criterion.

$$Cost(\phi|h) = CL(s_{1:n}|a_{1:n}) + CL(r_{1:n}|s_{1:n}, a_{1:n}) + CL(\phi)$$

A practically useful modification adds a parameter α to control the balance between reward coding and state coding,

$$Cost_\alpha(\phi|h_n) := \alpha CL(s_{1:n}|a_{1:n}) + (1-\alpha)CL(r_{1:n}|s_{1:n}, a_{1:n}) + CL(\phi).$$

- A global stochastic search (e.g. simulated annealing) is used to find the ϕ with minimal cost.
- For fixed ϕ, MDP methods can be used to find a good policy

Model-free cost criterion

Daswani&Sunehag&Hutter 2013 introduced a fitted-Q cost

$$Cost_{QL}(\phi) = \min_Q \frac{1}{2}\sum_{t=1}^{n}(r_{t+1}+\gamma \max_a Q(\phi(h_{t+1}),a)-Q(\phi(h_t),a_t))^2+Reg(\phi)$$

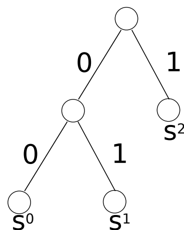
- $Cost_{QL}$ also extends easily to the linear function approximation setting by approximating $Q(h_t,a_t) \leftarrow \xi(h_t,a_t)^T w$ where $\xi : \mathcal{H} \times \mathcal{A} \rightarrow \mathbb{R}^k$ for some $k \in \mathbb{R}$.
- Connects feature rl to feature selection for TD methods, e.g. Lasso-TD or Dantzig-TD using ℓ_1 regularization while above *Reg* tends to be a more aggressive ℓ_0.
- For a fixed policy, a TD cost without $\max_a$ can be defined but one can also reduce the problem to feature selection for supervised learning using pairs (s_t, R_t) where R_t is the return achieved after state s_t.

Input : Environment $Env()$;
Initialise ϕ ;
Initialise history with observations and rewards from
$t = init_history$ random actions;
Initialise M to be the number of timesteps per epoch;
while *true* **do**
  $\phi = SimulAnneal(\phi, h_t)$;
  $s_{1:t} = (\phi(h_1), \phi(h_2), ..., \phi(h_t))$;
  $\pi = FindPolicy(s_{1:t}, r_{1:t}, a_{1:t-1})$;
  for $i = 1, 2, 3, ...M$ **do**
    $a_t \leftarrow \pi(s_t)$;
    $o_{t+1}, r_{t+1} \leftarrow Env(h_t, a_t)$;
    $h_{t+1} \leftarrow h_t a_t o_{t+1} r_{t+1}$;
    $t \leftarrow t + 1$;
  end
end

Algorithm 1: A high-level view of the generic ΦMDP algorithm.

Feature maps

- Tabular : use suffix trees to map histories to states (Nguyen&Sunehag&Hutter 2011,2012). Looping trees for long-term dependences (Daswani&Sunehag&Hutter 2012)
- Function approximation : define a new feature class of event selectors. A feature ξ_i checks the $n-m$ position in the history (h_n) for an observation-action pair (o,a).



If the history is $(0,1),(0,2),(3,4),(1,2)$ then a event-selector checking 3 steps in the past for the observation-action pair $(0,2)$ will be turned on.

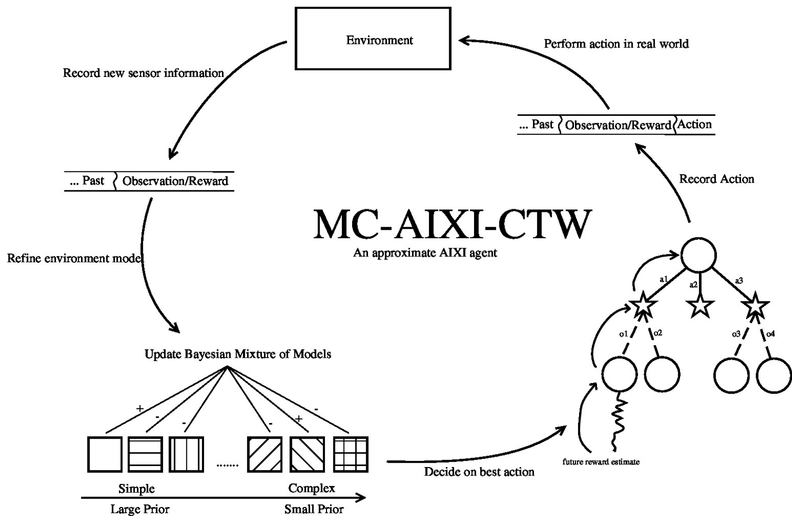
Bayesian general reinforcement learning: MC-AIXI-CTW

Unlike Feature RL, the Bayesian approach does not pick one map but uses a mixture of all instead. The problem is (again) split into two main areas:

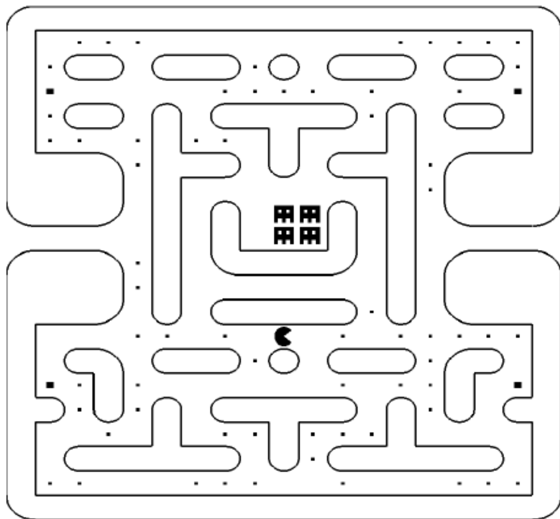
- Learning - online sequence prediction / model building
- Planning/Control - search / sequential decision theory

The hard parts:

- Large model class required for Bayesian mixture predictor to have *general* prediction capabilities.
- Fortunately, an efficient and general class exists: all Prediction Suffix Trees of maximum finite depth D. Class contains over $2^{2^{D-1}}$ models!
- The planning problem can be performed approximately with Monte-Carlo Tree Search (UCT)
- MC-AIXI-CTW (Veness et. al. 2010) combines the above


Environment
Perform action in real world
Record new sensor information
... Past
Observation/Reward
Action
... Past
Observation/Reward
Record Action
MC-AIXI-CTW
An approximate AIXI agent
Refine environment model
a1
a2
a3
o1
o2
o3
o4
Update Bayesian Mixture of Models
+
-
-
-
+
-
....
Decide on best action
future reward estimate
Simple
Complex
Large Prior
Small Prior

Domain : POCMAN



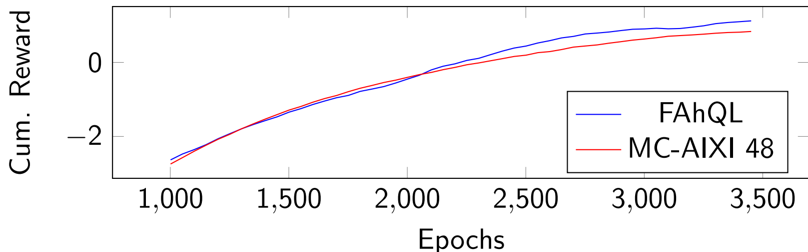


Figure : MC-AIXI vs hQL on Pocman

Agent	Cores	Memory(GB)	Time(hours)	Iterations
MC-AIXI 96 bits	8	32	60	$1 \cdot 10^5$
MC-AIXI 48 bits	8	14.5	49.5	$3.5 \cdot 10^5$
FAhQL	1	0.4	17.5	$3.5 \cdot 10^5$

Conclusions/Outlook

- Reinforcement Learning is a powerful paradigm within which (basically) all AI problems can be formulated
- Many practical successes using MDPs by engineering problem reductions/reprentations
- Practically increasing the versatility of agents by learning reductions automatically.
- Recently introduced arcade gaming environment (from Alberta) for RL containing all ATARI games. Aim, have one generic RL agent solve all!
- Use data on how valuable states are from either simulations or experience to reduce complexity