
8: Introduction to Reinforcement Learning

Tim Kovacs

University of Bristol

This slides are modified versions of slides by
Andrew Barto

Where we are in the unit

- We've seen how evolution adapts a species over generations
 - Now we'll see how an agent can adapt during its lifetime
 - This part of the unit will follow Sutton & Barto's book closely
 - Slides are based on theirs, but with many changes and additions

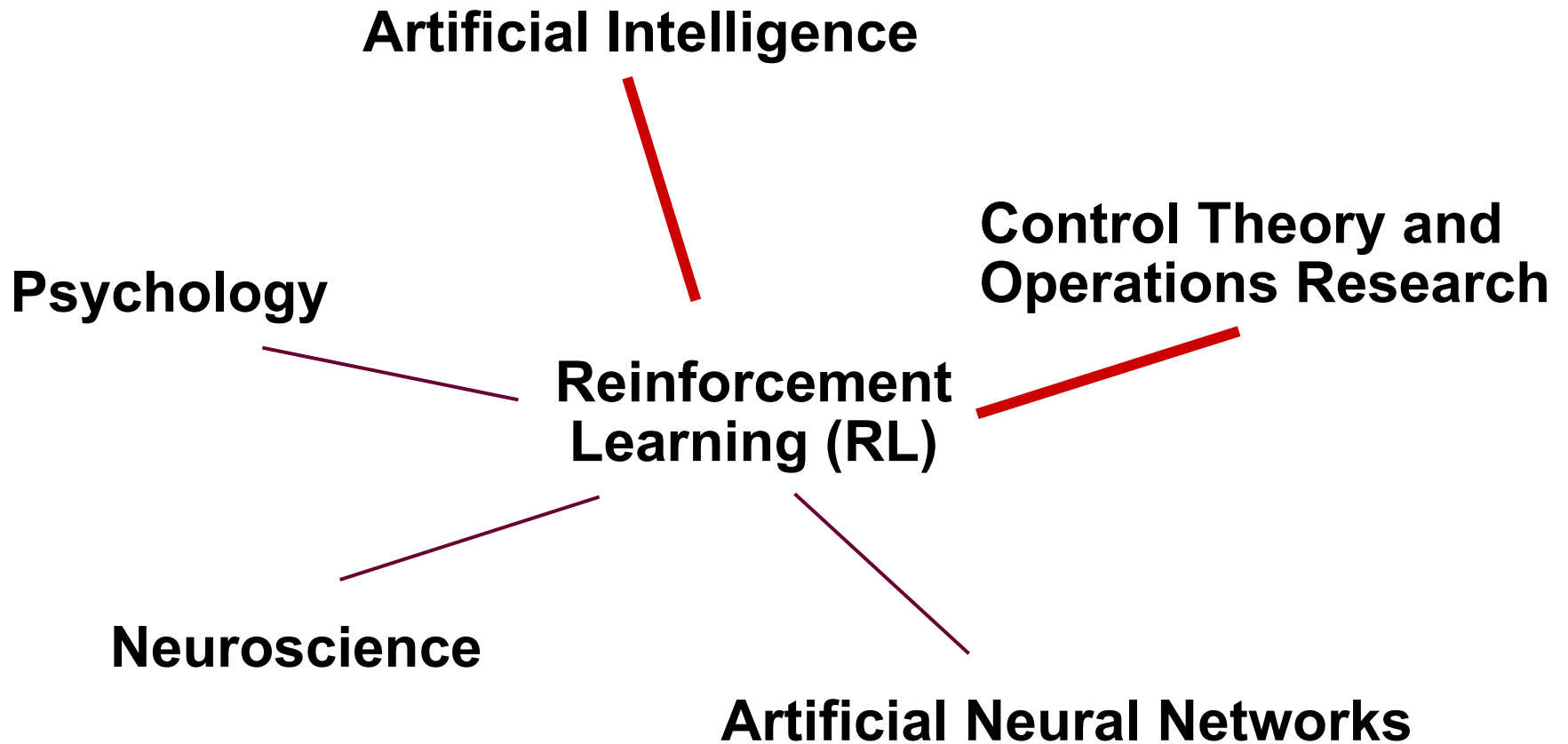
EC & RL

- What they have in common: autonomous learning
 - No teacher (unlike supervised learning)
- How they differ: feedback
 - In evolution a chromosome gets 1 fitness evaluation
 - In RL an agent takes many actions and gets many rewards

What's new about RL?

- Of course a phenotype can have a long lifetime
 - Its fitness evaluation may not happen all at once
 - It can be an agent which takes many actions and gets many rewards
 - E.g. a neural net which is evaluated on a dataset
 - We just sum up the rewards to get one fitness
- The key difference between EC and RL:
 - In RL we learn from individual rewards
 - not just the sum of rewards over a lifetime

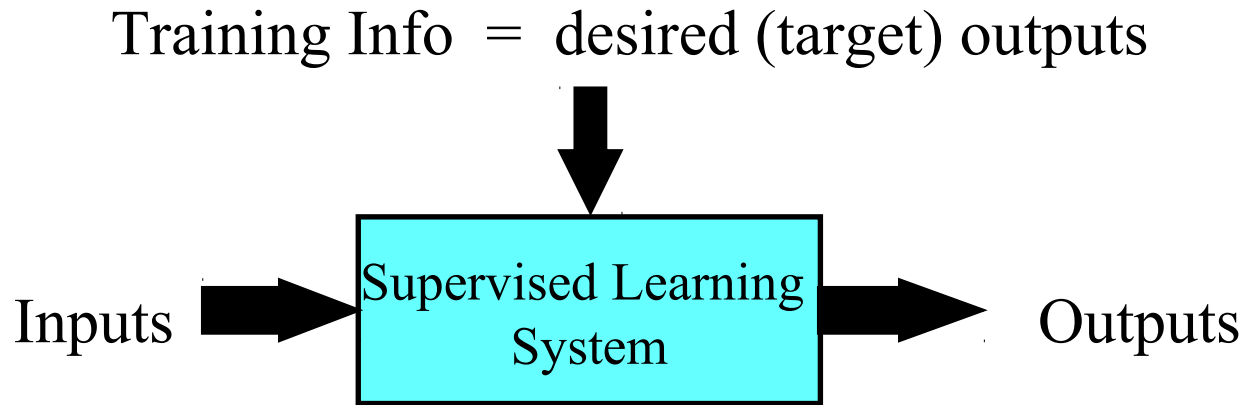
Chapter 1: Introduction



What is Reinforcement Learning?

- 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

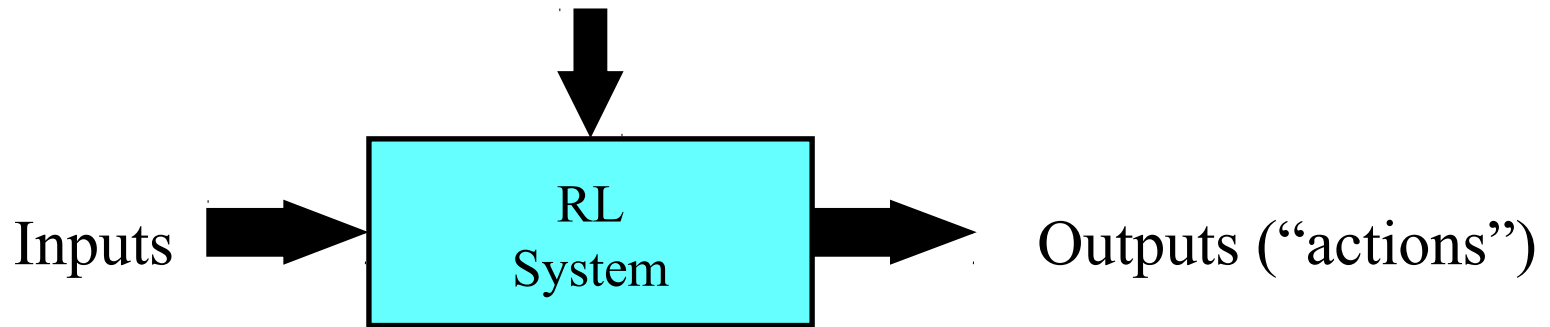
Supervised Learning



$$\text{Error} = (\text{target output} - \text{actual output})$$

Reinforcement Learning

Training Info = evaluations (“rewards” / “penalties”)



Objective: get as much reward as possible

Supervised Learning

Step: 1

Teacher: You are in state 9. What is its value?

Learner: 90.

Teacher: Its value is 95.

Step: 2

Teacher: You are in state 32. What is its value?

Learner: -10.

Teacher: Its value is 20.

Step: 3 ...

Reinforcement Learning

Step: 1

World: You are in state 9. Choose action A or C.

Learner: Action A.

World: Your reward is 100.

Step: 2

World: You are in state 32. Choose action B or E.

Learner: Action B.

World: Your reward is 50.

Step: 3 ...

EC is actually a form of RL

- In an RL problem you learn from rewards
 - Not from a labelled training set
 - EC's fitness is a reward signal
 - Therefore, EC is an RL problem

Step: 1

Species: What's the fitness of chromosome 0011?

World: 2

Step: 2

Species: What's the fitness of chromosome 1011?

World: 3

...

Terminology

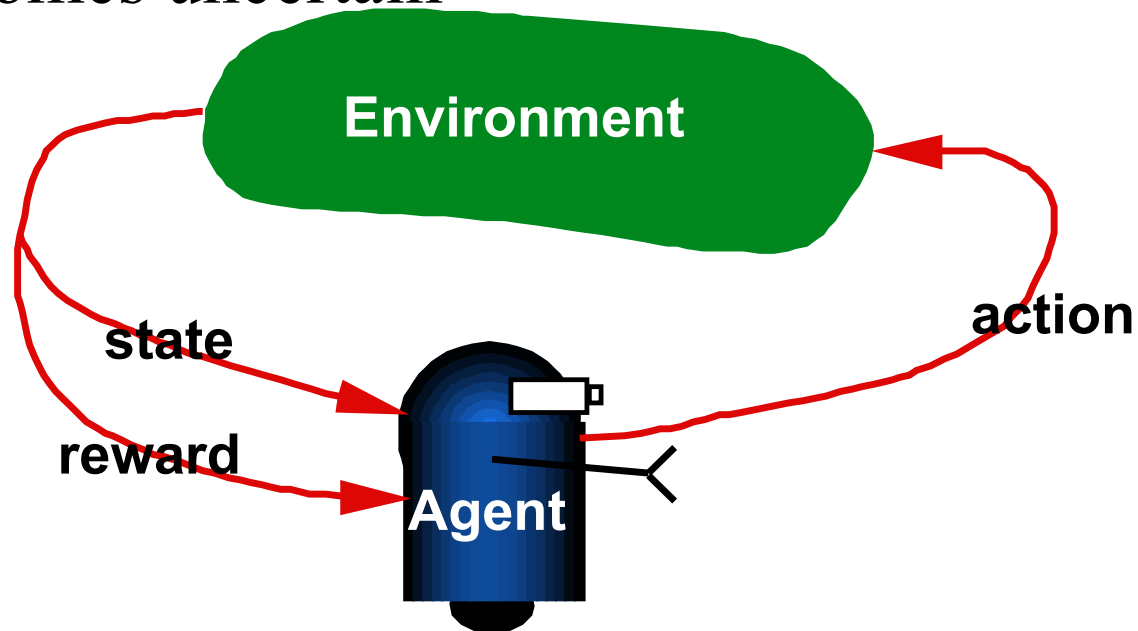
- Any problem where you learn from rewards is an RL problem
- Any algorithm which solves RL problems is an RL algorithm
- BUT: when people talk about RL algorithms, they usually mean algorithms which learn *value functions*
 - All the methods in Sutton and Barto learn value functions
- Using the broader definition of RL problems above, EC methods are also RL algorithms
- From now on I won't talk about EC much

Key Features of RL

- Learner is not told which actions to take
 - Trial-and-Error search
 - The need to *explore* and *exploit*
- Sequences of actions
 - Sacrifice short-term gains for greater long-term gains
- Can be applied to wide range of problems
 - Learner needs only very weak feedback
 - Not much needs to be known about task
- Possibility of delayed reward
- Non-deterministic environments
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

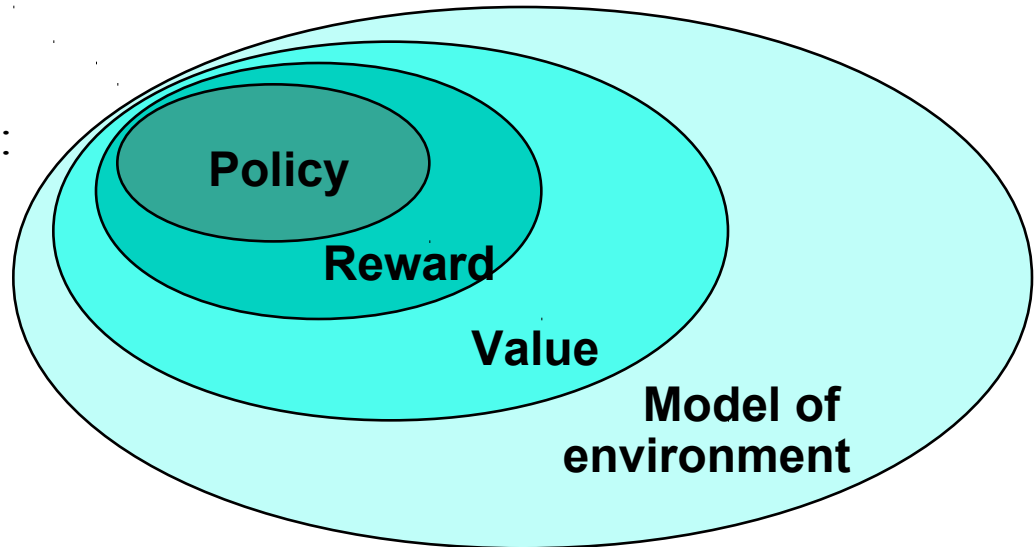
Complete Agent

- Temporally situated
 - Continual learning and planning
 - Object is to *affect* the environment
 - Environment is stochastic (i.e. non-deterministic) so outcomes uncertain



Elements of RL

How central element is:



Essential:

- **Policy:** what to do
- **Reward:** the pleasure/pain agent feels right now

Optional:

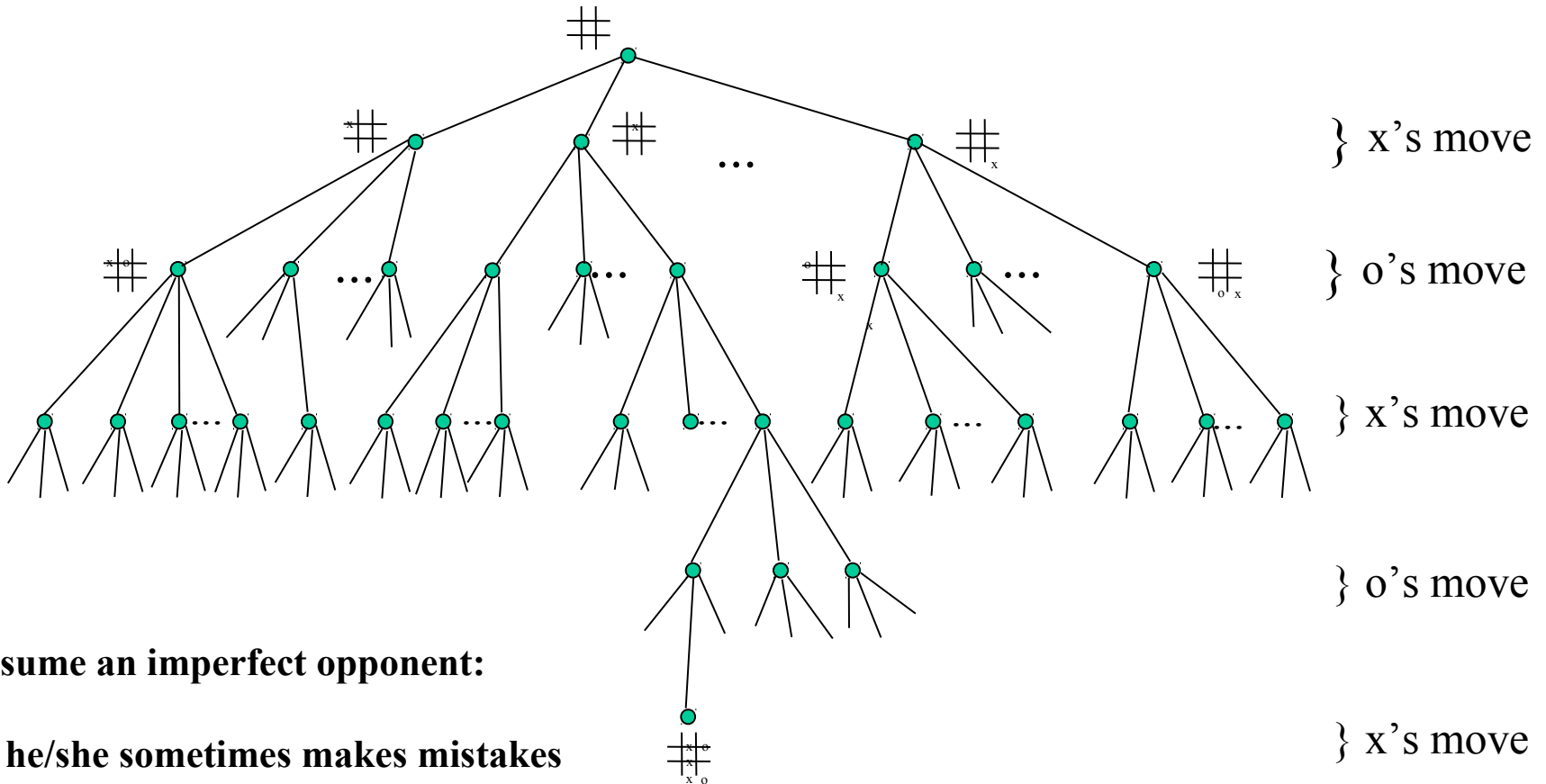
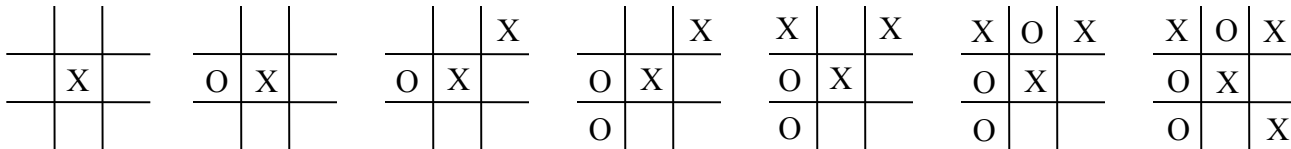
- **Value:** measure of reward now and in the future
- **Model:** what follows what in the world

Reward and Value

- **Reward** measures how good what I'm doing right now is
- **Value** measures how good what I'm doing now will turn out to be in the long run

- Rewards are part of the problem definition
- The agent must work out values itself
 - This is *the main problem* an RL agent faces
 - Once correct values are known, solution is trivial
 - We don't need to know solution to specify problem

An Extended Example: Tic-Tac-Toe



The Reward Function

- There are 2 types of states
 - Terminal states: the game is over
 - Win: $R = 1$
 - Lose or Draw: $R = 0$
 - Non-terminal states: the game continues
 - $R = 0$
- This does not give the learner a lot of clues!
- We write the reward for state s as $R(s)$
- We can implement with
 - a look-up table listing each possible state
 - rules which define different kinds of states

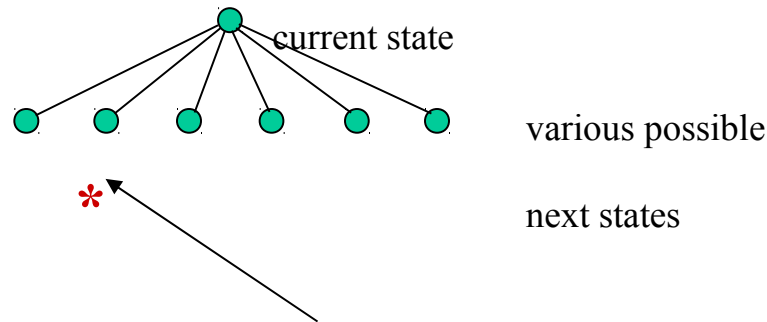
The value function

- Make a table with one entry per state
- Write the value of state s as $V(s)$
- For terminal states $V(s) = R(s)$
 - We never change their values
- For non-terminal states initially $V(s) = 0.5$
 - We will update their values with experience
 - We choose 0.5 because it's halfway between 0 and 1
 - In other words
 - we assume they're equally likely to lead to wins and losses
 - we're not biasing the values either way

Selecting actions

Now play lots of games.

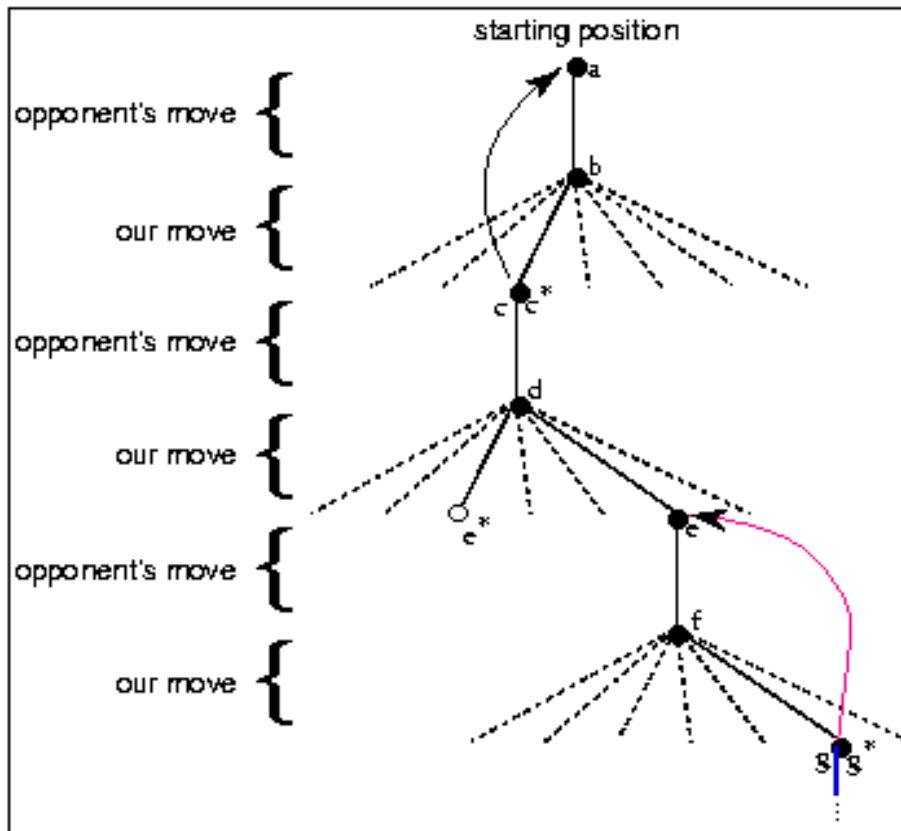
To pick moves look ahead one step:



90% of the time pick state with highest estimated prob. of winning — the largest $V(s)$; a *greedy* move (shown with *)

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

RL Learning Rule for TTT



b,d,f are our moves

Possible moves are dotted lines

Greedy moves are labeled with *

Actual move just has letter

After each greedy move do a 'backup': move $V(s)$ closer to $V(s')$

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

where s is state before greedy move and s' is state after

An RL Learning Rule

- Each backup moves $V(s)$ closer to $V(s')$

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

- How much closer is controlled by α
 - called the *step size* or *learning rate*
 - between 0 and 1
- Eventually the values $V(s)$ will converge to the probability of winning if we always play greedily
 - But only given this formulation
 - For other set-ups $V(s)$ is not a probability
 - E.g. giving reward of -1 for losing will make it prefer draws to losing, but $V(s)$ is no longer a probability

Where's the reward?

- We did not include reward because *in this reward function* it is always 0 for non-terminals so has no effect
- Equivalent more general backup:
$$V(s) \leftarrow V(s) + \alpha [V(s') - V(s) + R(s)]$$
- We'll see versions of this backup again many times

What $V(s)$ means

If we update only on greedy moves:

- $V(s)$ converges to prob. of winning when we always take greedy moves

If we update on all moves:

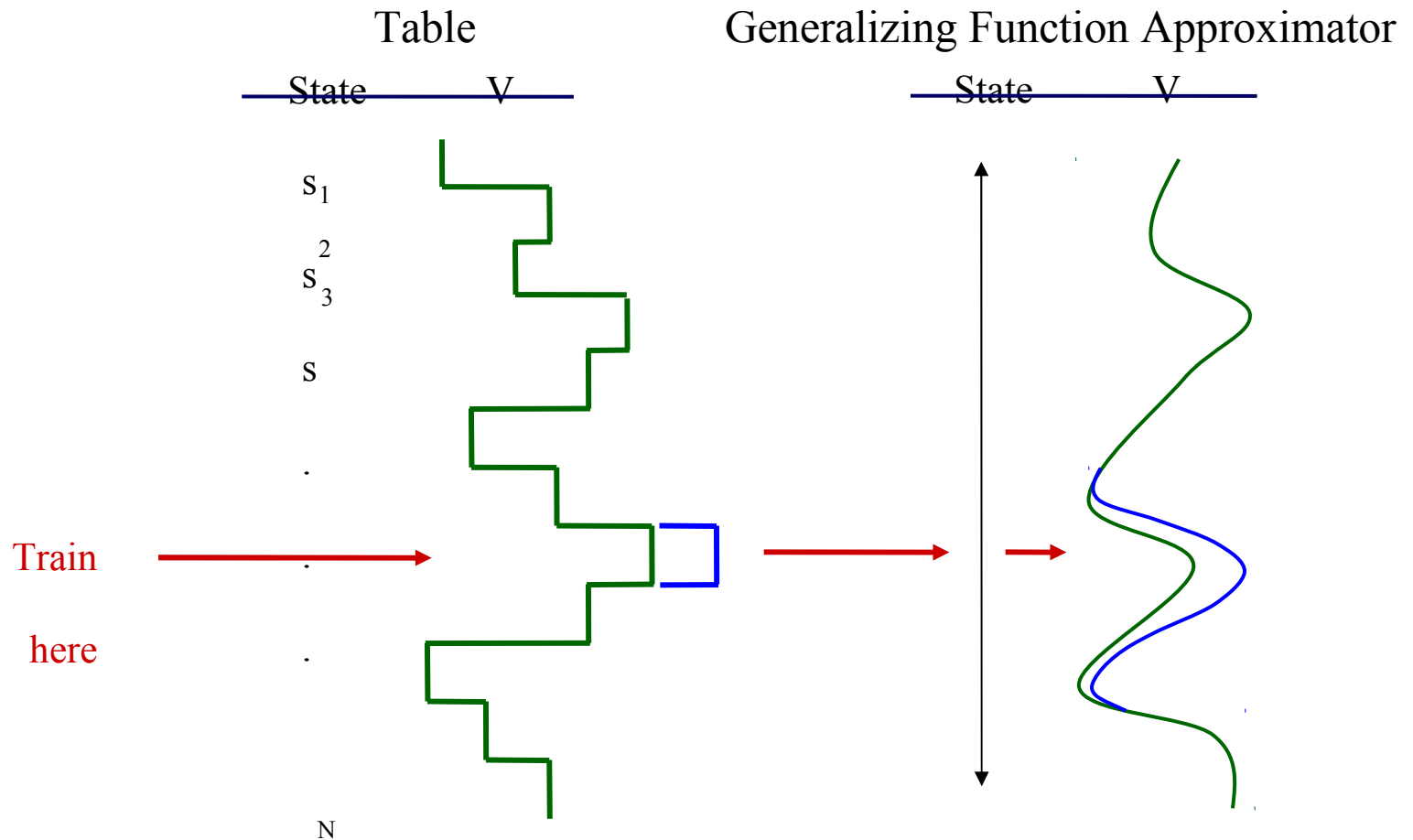
- $V(s)$ converges to prob. of winning when we explore 10% of the time

But again, in RL $V(s)$ is not usually a prob. It's just some measure of long-term value

How can we improve this TTT player?

- Take advantage of symmetries
 - Representation/generalization
 - How might this backfire?
- Do we need “random” moves? Why?
 - Do we always need a full 10%?
- Can we learn from “random” moves?
- Can we learn offline?
 - Pre-training from self play?
 - Using learned models of opponent?
-

Generalisation



Generalisation

- When we visit 1 state, we can generalise to similar states
 - Update $V(s)$ for many similar states
- How?
 - E.g. use a neural network to approximate value function
- Advantages
 - Speeds up learning (visit 1 state, update many)
 - Tells how to act in states we haven't visited yet
 - Can save memory
 - Store a few NN weights instead of a huge table
- Issue: how to define “similar”?
 - Different for each learning problem...

Other points on TTT example

In some ways simpler than most RL problems:

- Finite, small number of states
- One-step look-ahead is always possible
- State completely observable

In some ways harder:

- We need to know how to look ahead to successor states
- Is there an easy way to avoid this?

How could we use EC for TTT?

- We could use the chromosome as a look-up table
 - Each gene specifies the action for 1 state
 - Simple approach, but there are hundreds of states
 - Needs a big chromosome
 - No generalisation
 - Won't scale up to bigger problems (even 4x4 TTT board is *much* bigger)
- Chromosome could encode IF-THEN rules
 - IF state has certain properties THEN do a certain action
- Chromosome could encode a neural network
 - Optionally adjust the weights with backpropagation based on how well it plays
- There are other ways...

Notable RL Applications

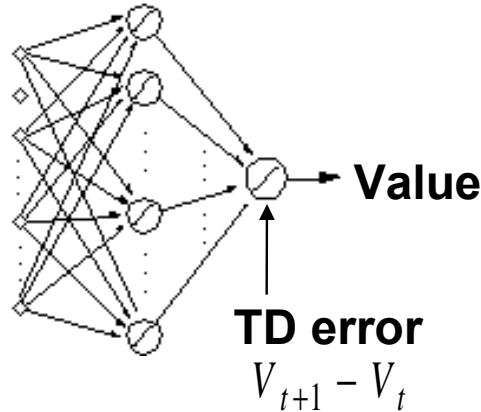
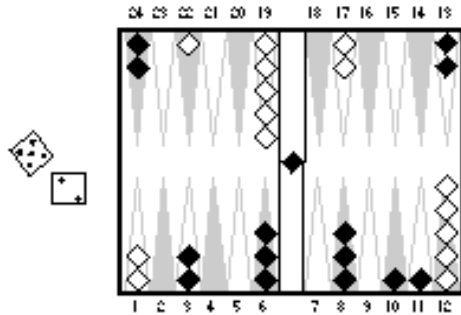
- Go through them in your own time
- Things to note
 - None of them is an “off-the-shelf” application of RL
 - They're all customised
 - Finding the right representation can take a lot of time
 - See the backgammon example
 - As with EC, you need the right tool for the job

Some Notable RL Applications

- **TD-Gammon:** Tesauro
 - world's best backgammon program
- **Elevator Control:** Crites & Barto
 - high performance down-peak elevator controller
- **Inventory Management:** Van Roy, Bertsekas, Lee & Tsitsiklis
 - 10–15% improvement over industry standard methods
- **Dynamic Channel Assignment:** Singh & Bertsekas, Nie & Haykin
 - high performance assignment of radio channels to mobile telephone calls

TD-Gammon

Tesauro, 1992–1995



Action selection
by 2–3 ply search

Start with a random network

Play very many games against self

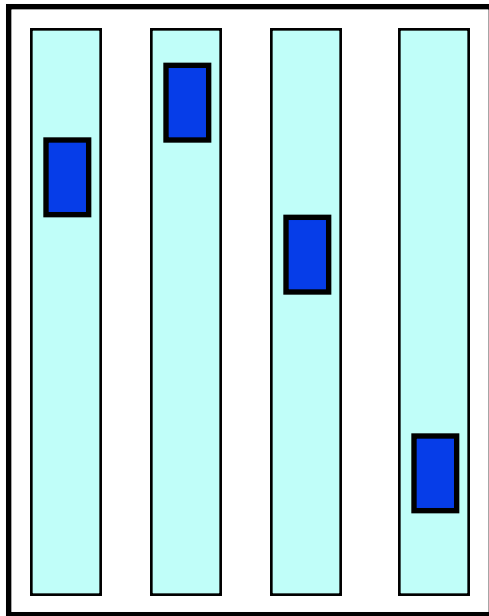
Learn a value function from this simulated experience

This produces arguably the best player in the world

Elevator Dispatching

Crites and Barto, 1996

10 floors, 4 elevator cars



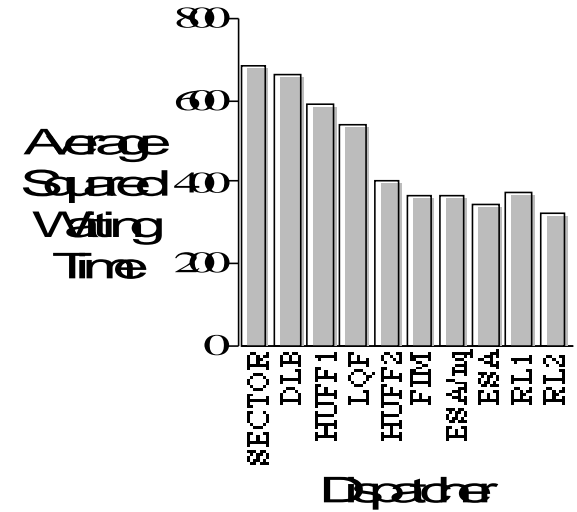
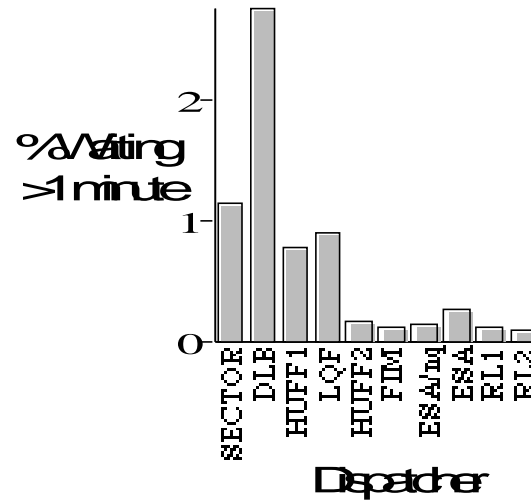
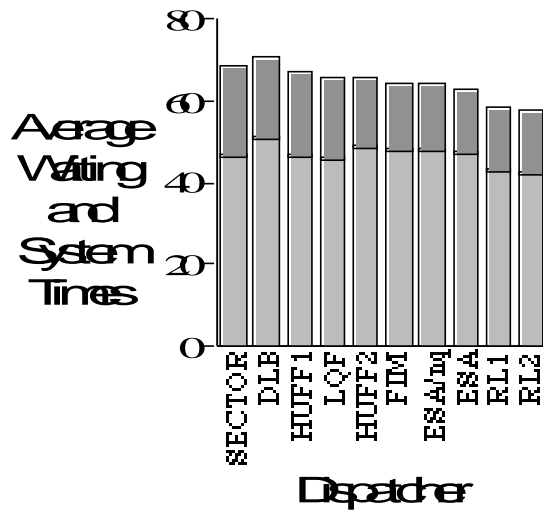
STATES: button states;
positions, directions,
and motion states of
cars; passengers in cars
& in halls

ACTIONS: stop at, or go
by, next floor

REWARDS: roughly, -1
per time step for each
person waiting

Conservatively about 10^{22} states

Performance Comparison



Some RL History

Trial-and-Error learning

Thorndike (Ψ)
1911

Minsky

Klopf

Barto et al.

Temporal-difference learning

Secondary reinforcement (Ψ)

Samuel

Holland

Witten

Sutton

Optimal control, value functions

Hamilton (Physics)
1800s

Shannon

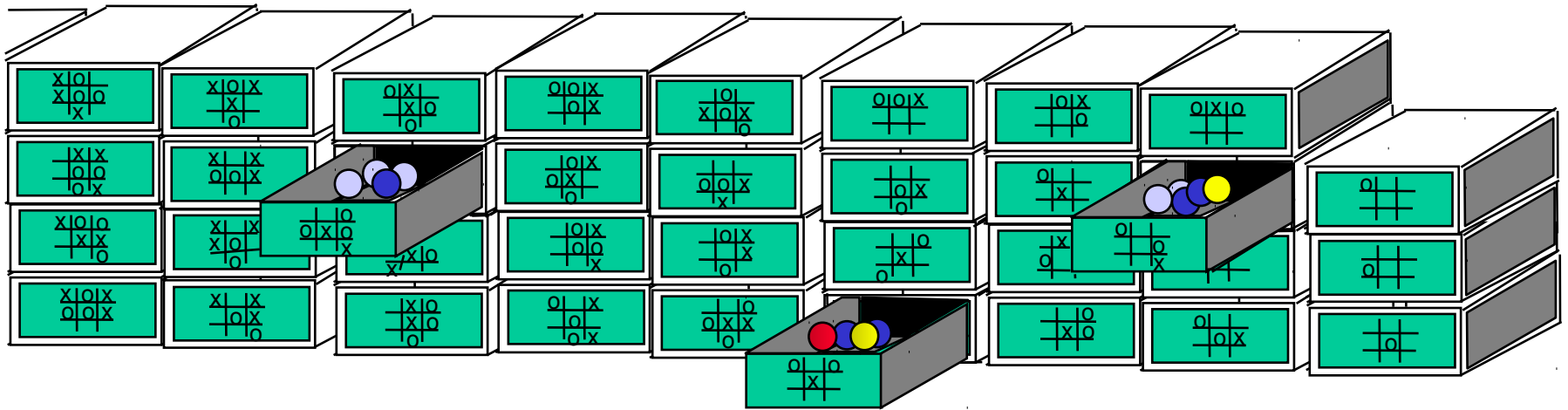
Bellman/Howard (OR)

Werbos

Watkins

MENACE (Michie 1961)

“Matchbox Educable Noughts and Crosses Engine”



- States are represented by matchboxes
- Each action's value represented by marbles of one colour
- All boxes initially have the same number of marbles and same number of each colour
- Updates alter number of marbles to be closer to successor state
- After training it could play reasonable games

Chapters

- Part I: The Problem
 - 1 - Introduction
 - 2 - Evaluative Feedback
 - 3 - The Reinforcement Learning Problem
- Part II: Elementary Solution Methods
 - 4 - Dynamic Programming
 - 5 - Monte Carlo Methods
 - 6 - Temporal Difference Learning
- Part III: A Unified View
 - 7 - Eligibility Traces
 - 8 - Generalization and Function Approximation
 - 9 - Planning and Learning
 - 10 - Dimensions of Reinforcement Learning
 - Ch. 11 / Lecture 10 - Case Studies