MODEL-BASED REINFORCEMENT LEARNING

Scott O'Hara
Metrowest Developers Machine Learning Group
09/12/2018

REFERENCES

The material for this talk is drawn from the slides, notes and lectures from several offerings of the CS181 course at Harvard University:

- *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Sarah Finney, Spring 2009
- *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Prof. David C Brooks, Spring 2011
- *CS181 – Machine Learning*, Prof. Ryan P. Adams, Spring 2014. https://github.com/wihl/cs181-spring2014
- *CS181 – Machine Learning*, Prof. David Parkes, Spring 2017. https://harvard-ml-courses.github.io/cs181-web-2017/

As well as notes and lectures from Stanford course CS229 :

- *CS229 – Machine Learning, Andrew Ng.* https://see.stanford.edu/Course/CS229

OVERVIEW

1. **Overview**
   - Types of Machine Learning
   - Markov Decision Processes
   - Reinforcement Learning
   - Applications
2. **Review of MDP Algorithms**
   - The Bellman equations
   - Expectimax (finite horizon)
   - Value Iteration (finite horizon)
   - Value Iteration (infinite horizon)
   - Policy Iteration(infinite horizon)
3. **Reinforcement Learning**
   - The basic idea
   - Model-Based RL
   - Learning the reward and transition probabilities
   - Credit assignment
   - Exploration vs. exploitation
4. **Next Time**
   - Q-learning

TYPES OF MACHINE LEARNING

There are (at least) 3 broad categories of machine learning problems:

Supervised Learning

$\boldsymbol{Data = \{(x_1, y_1), \ldots, (x_n, y_n)\}}$

e.g., linear regression, decision trees, SVMs

Unsupervised Learning

$\boldsymbol{Data = \{x_1, , \ldots, x_n\}}$

e.g., K-means, HAC, Gaussian mixture models

Reinforcement Learning

$\boldsymbol{Data = \{s_1, a_1, r_1, s_2, a_2, r_2 \ldots\}}$

an agent learns to act in an uncertain environment by training on data that are sequences of **state, action, reward.**

MARKOV DECISION PROCESSES

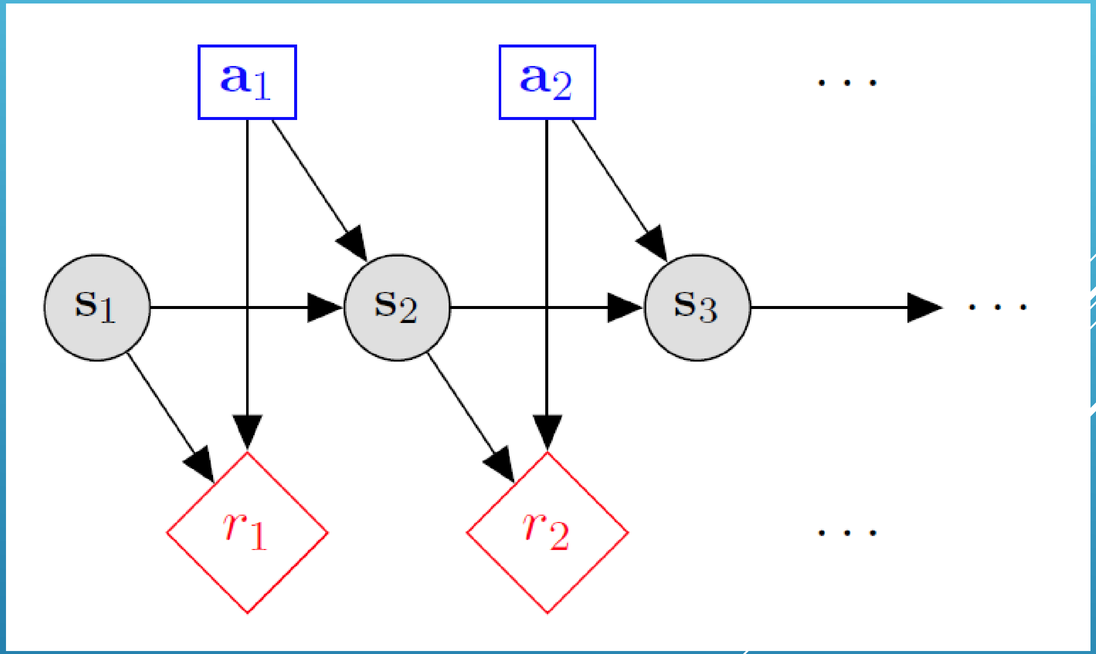
- Markov Decision Processes provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.

- The initial analysis of MDPs assume **complete knowledge** of states, actions, rewards, transitions, and discounts.

<https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=855934986> [accessed 11 September 2018]

MARKOV DECISION PROCESSES

- **States:** $s_1, \ldots, s_n$
- **Actions:** $a_1, \ldots, a_m$
- **Reward Function:** $\mathrm{r}(s, a) \in R$
- **Transition model:** $\mathrm{p}(s'|s, a)$
- **Discount factor:** $\gamma \in [0, 1]$



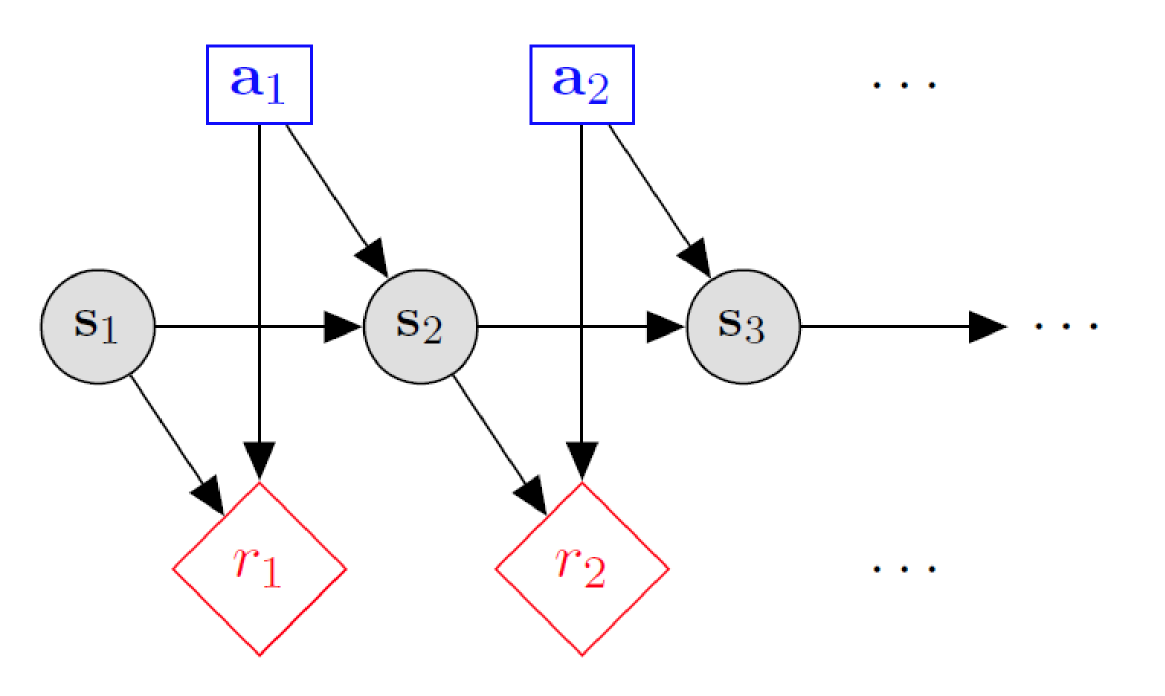
<https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=855934986> [accessed 11 September 2018]

MDP GOAL: FIND AN OPTIMAL POLICY π

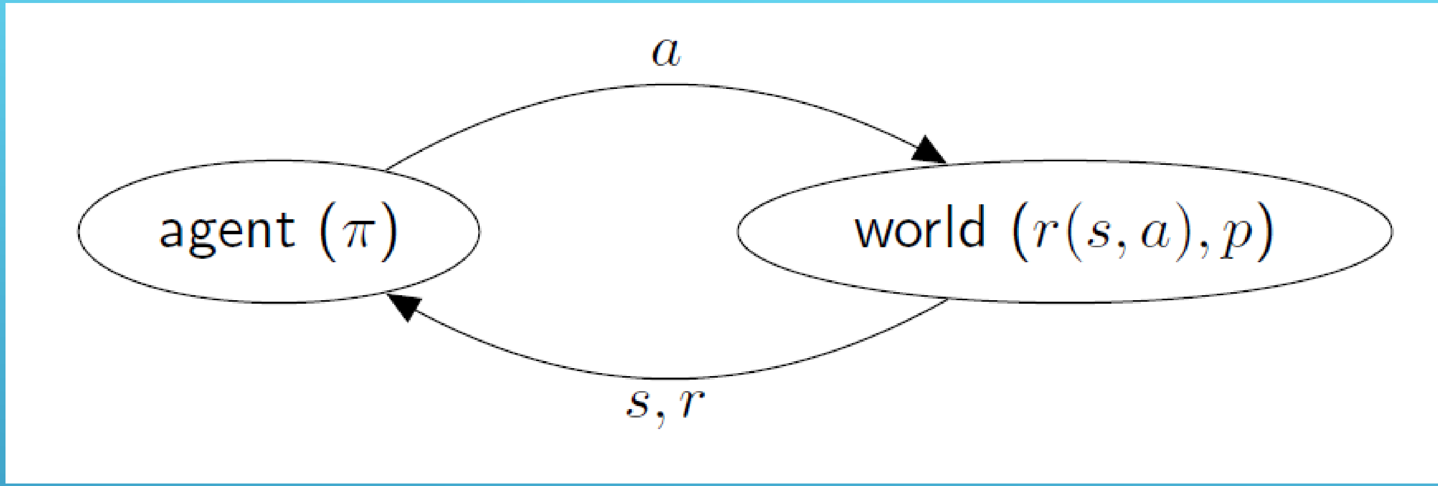
GOAL: find a **policy π** that tells you what action to take in each state. We want to find 'rewarding' policies.

n state policy:

$$\pi(s_1) = a_1$$
$$\pi(s_2) = a_2$$
$$\pi(s_3) = a_3$$
$$\dots$$
$$\pi(s_n) = a_n$$



REINFORCEMENT LEARNING



- Agent knows the current state *s*, takes action *a*, and gets reward *r*.
- There is **no access** to reward model *r(s,a)* or transition model *p(s' | s,a)*
- Agent only sees the outcome reward *r* and the next state *s'*.
- Under these conditions, it is a very challenging problem to learn π.

REVIEW OF MDP ALGORITHMS

REVIEW OF MDP ALGORITHMS

There are 4 standard MDP algorithms that assume complete knowledge:

- **Expectimax** (finite horizon, $\gamma = 1$)
- **Value Iteration** (finite horizon, $\gamma = 1$)
- **Value Iteration** (infinite horizon, $\gamma \in [0,1)$)
- **Policy Iteration** (infinite horizon , $\gamma \in [0,1)$)

BELLMAN EQUATIONS

- The planning problem for an MDP is:

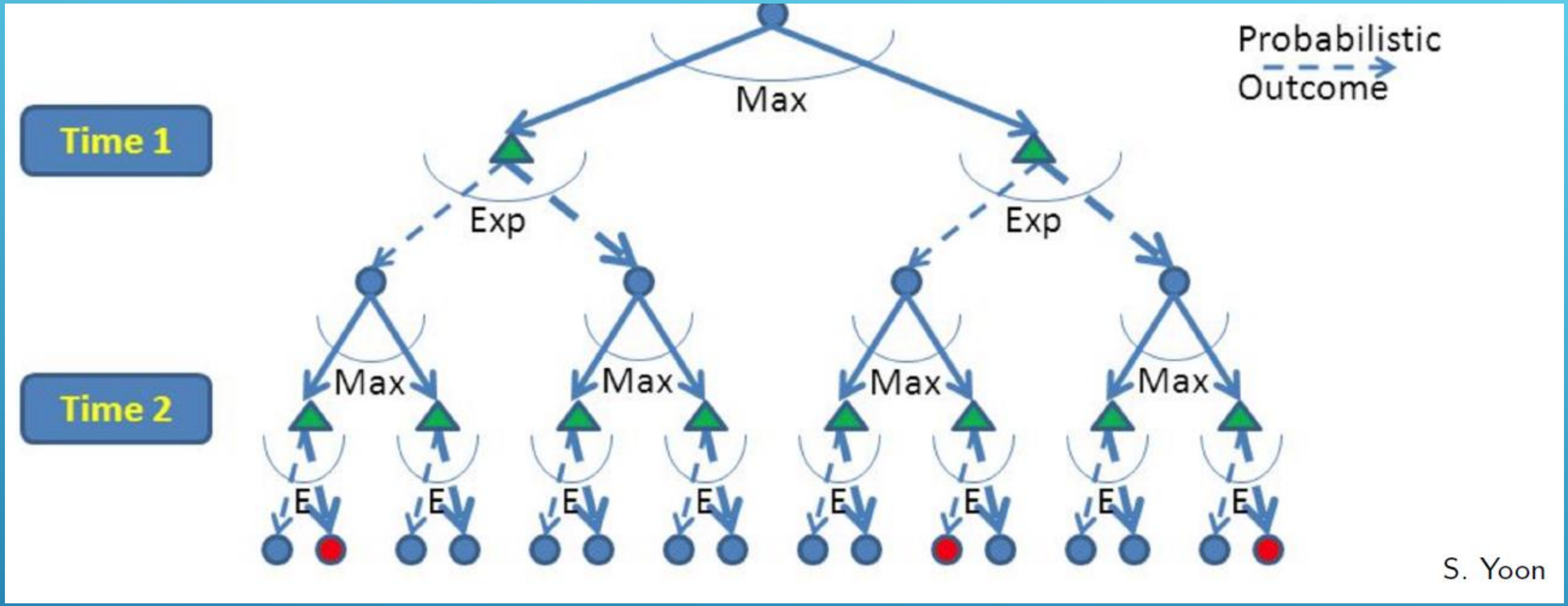
$$\pi^* \in \arg\max_{\pi} V^{\pi}(s)$$

- Bellman equations:

$$V^*(s) = \max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s')\right], \forall s$$

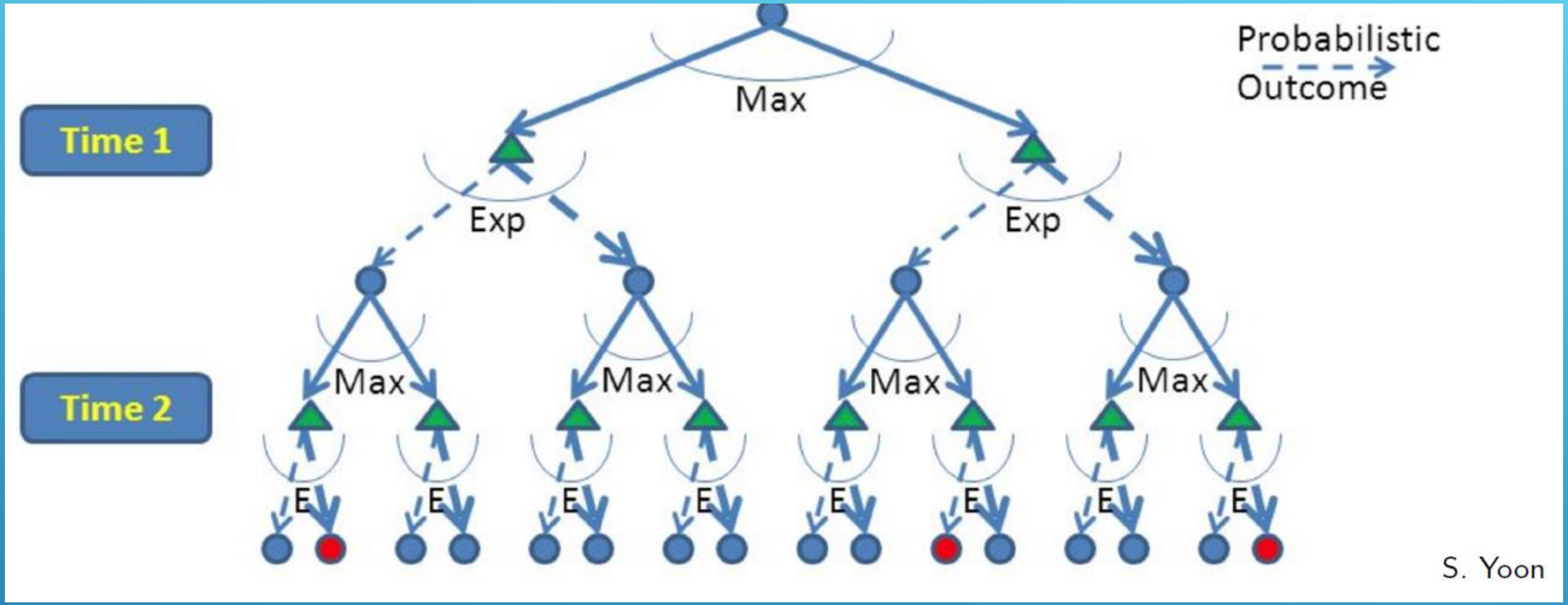
- All MDP algorithms use some variant of the Bellman equations.

EXPECTIMAX: TOP-DOWN, RECURSIVE



- Build out a look-ahead tree to the decision horizon; take the max over actions, expectations over next states.
- Solve from the leaves, backing-up the expectimax values.
- Problems: (1) computation is exponential in the horizon; (2) may expand the same subtree multiple times.

EXPECTIMAX: A GAME AGAINST NATURE



- Expectimax is like a game-playing algorithm except the opponent is nature.
- Expectimax is strongly related to the minmax algorithm used in game theory, but the response is probabilistic.
- Nodes where you move: S (△)
- Nodes where nature moves: <S,A> (○)

EXPECTIMAX $(Finite\ Horizon\ T)$

function *EXPECTIMAX*(s)

if s is a terminal **then**

return 0

else

return $\max_{a \in A}[R(s,a) + \sum_{s'} P(s'|s,a)\ EXPECTIMAX(s')]$

$\pi^T(s) = EXPECTIMAX(s)$

VALUE ITERATION $(Finite\ Horizon\ T)$

- Value iteration with a finite horizon works from the bottom-up using dynamic programming.
- The idea is to break up the problem by number of the number of steps to go.
- Start with the base case values with no timesteps to go.
- Given optimal policy for k-1 steps to go, compute values for k steps to go

VALUE ITERATION ($\text{F}inite\ Horizon\ T$)

1. For each state s, initialize $V_0(s) = 0$

2. For $k \leftarrow 1 \ldots T$ {

   For every state s:

   $$V_k(s) = \max_{a \in A}[R(s,a) + \sum_{s'} P(s'|s,a)V_{k-1}(s')]$$

}

3. $\pi^T(s) = \underset{a \in A}{\operatorname{argmax}}[\,R(s,a) + \sum_{s' \in S} P(s'|s,a)V_T(s')]$

VALUE ITERATION (∞ HORIZON, $\gamma \in [0,1)$)

- Value iteration with an infinite horizon is a generalization of value iteration with a finite horizon.

- The main change is that the discount factor γ is set to less than one discounting the value of states farther in the future.

- It can be shown that V will converge over time to V^*

- Iterate the update equation until the changes in V fall below some ϵ.

VALUE ITERATION (∞ HORIZON, $\gamma \in [0,1)$)

1. For each state s, initialize $V(s) = 0$

2. Repeat until convergence {

   For every state s:

   $$V'(s) = \max_{a \in A}[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')]$$

   $V \leftarrow V'$

}

3. $\pi^*(s) = \operatorname{argmax}_{a \in A}[\, R(s,a) + \sum_{s' \in S} P(s'|s,a)V_T(s')]$

POLICY ITERATION (∞ HORIZON, $\gamma \in [0,1)$)

- For value iteration, the policy often stops changing long before the values converge.
- Policy iteration, iterates on the policy rather than V.
- Policy iteration can converge in many fewer iterations than value iteration.
- However, the loop body in policy iteration takes much longer than value iteration.
- Harvard CS181 notes say policy iteration is faster. Andrew Ng says that value iteration is faster.

POLICY ITERATION (∞ HORIZON, $\gamma \in [0,1)$)

1. Initialize policy π randomly

2. Repeat until policy π does not change {

   a) Solve $V^{\pi}(s) = R(s, \pi(s)\,) + \gamma \sum_{s' \in S} P(s'|s, \pi(s))V(s')]$

   b) For every state s:
   let $\pi'(s) = \underset{a \in A}{argmax}\, R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V^{\pi}(s')$

   c) $\pi \leftarrow \pi'$

}

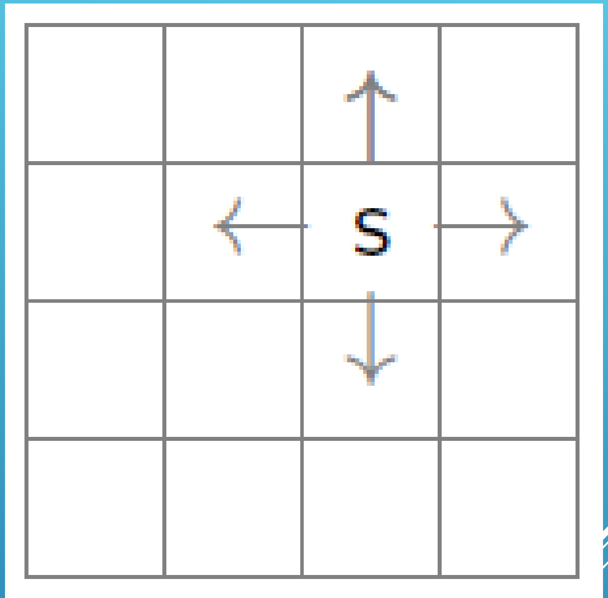
REINFORCEMENT LEARNING

REINFORCEMENT LEARNING: THE BASIC IDEA

- Select an action
- If action leads to reward, reinforce that action
- If action leads to punishment, avoid that action
- Basically, a computational form of Behaviorism (Pavlov, B. F. Skinner)

THE LEARNING FRAMEWORK

- Learning is performed **online**, learn as we interact with the world



- In contrast with supervised learning, there are no training or test sets. The reward is accumulated over interactions with the environment.

- Data is not fixed, more information is acquired as you go.

- The training distribution can be influenced by action decisions.

CHALLENGES

- **credit assignment** problem: how do you know which actions were responsible for success or failure?
- **exploration vs exploitation:** should you use your current model to collect rewards or should risk lower rewards now for a better model and higher rewards later?

MODEL-BASED REINFORCEMENT LEARNING

- Mechanism is Markov Decision Process
  - assume states and actions are known
  - assume states are fully observable

- Approach:
  - learn the MDP reward and transition parameters
  - solve the MDP to determine an optimal policy

- Appropriate when the model is unknown, but small enough to store and solve.

LEARN THE REWARD AND TRANSITION DISTRIBUTIONS

- Try every action in each state a number of times
- RTotal(a,s) = total reward for action a in state s
- N(a,s) = number of times action a taken in state s
- N(a,s,s') = number of transitions from s to s' on action a
- R(s,a) = RTotal(a,s) / N(a,s)
- T(a,s,s') = N(a,s) / N(a,s,s')

REWARD PARAMETER TABLE

Actions

States

R(s0,a0)	R(s0,a1)	R(s0,a2)
R(s1,a0)	R(s1,a1)	R(s1,a2)
R(s2,a0)	R(s2,a1)	R(s2,a2)
R(s3,a0)	R(s3,a1)	R(s3,a2)

TRANSITION PARAMETER TABLE

Action a

Next State

Current State

T(a,s0,s0)	T(a,s0,s1)	T(a,s0,s2)	T(a,s0,s3)
T(a,s1,s0)	T(a,s1,s1)	T(a,s1,s2)	T(a,s1,s3)
T(a,s2,s0)	T(a,s2,s1)	T(a,s2,s2)	T(a,s2,s3)
T(a,s3,s0)	T(a,s3,s1)	T(a,s3,s2)	T(a,s3,s3)

ITERATIVE IMPROVEMENT

- Swap between learning the model and solving the model to determine the optimal policy

- Why?
  - A poor policy may be expensive
  - might want to avoid learning a perfect model everywhere

- How often should one solve the MDP?
  - it depends on the relative costs

MODEL-BASED RL

Let π^0 be arbitrary

$k \leftarrow 0$

Experience $\leftarrow \emptyset$

Repeat

- $k \leftarrow k + 1$
- Begin in state i
- For a while:
    - Choose action a based on π^{k-1}
    - Receive reward r and transition to j
    - Experience $\leftarrow$ Experience $\cup < i, a, r, j >$
    - $i \leftarrow j$
- Learn MDP M from Experience
- Solve M to obtain π^k

CREDIT ASSIGNMENT

- How does model-based RL deal with the credit assignment problem?
- Learned MDP tells us reward values and transition probabilities
- Solving the MDP deals with long-term planning
- So, the problem of credit assignment is solved optimally.

CHOOSING AN ACTION

Let π^0 be arbitrary
$k \leftarrow 0$
Experience $\leftarrow \varnothing$
Repeat
  $k \leftarrow k + 1$
  Begin in state i
  For a while:
    Choose action a based on π^{k-1}
    Receive reward r and transition to j
    Experience $\leftarrow$ Experience $\cup < i, a, r, j >$
    $i \leftarrow j$
  Learn MDP M from Experience
  Solve M to obtain π^k

CHOOSING AN ACTION: EXPLORATION VS EXPLOITATION

- How should an agent choose an action? An obvious answer is simply follow the current policy. However, this is often the best way to improve your model.
- **Exploit:** use your current model to maximize the expected utility now.
- **Explore:** choose an action that will help you improve your model.

HOW/WHEN SHOULD WE EXPLORE / EXPLOIT ?

- **How to Exploit?** use the current policy.

- **How to Explore?**
  - choose an action randomly
  - choose an action you haven't chosen yet
  - choose an action that will take you to an unexplored state.

- **When to exploit? When to explore? How much time in exploration vs time in exploitation?**

CONVERGENCE TO THE OPTIMAL POLICY

- **If:**
  - every action is taken in every state infinitely often
  - the probability of exploration tends to zero
- **Then:**
  - model-based RL will converge to the optimal policy
- Choose exploration strategy that approximates these conditions.

EXPLORATION STRATEGY 1: TWO STAGES

- Explore until time T, then exploit

- Weaknesses:
  - we may not explore log enough to get an accurate model
  - we may get stuck with a sub-optimal policy.
  - we may have to behave stupidly for a long time.

- Uses:
  - appropriate if we only need to learn/solve MDP once
  - learn/solve in a simulation then act in the real world.

EXPLORATION STRATEGY 2: ϵ-GREEDY

- Explore with probability ϵ. Exploit with probability $1 - \epsilon$.
- Weaknesses:
  - Does not exploit when learning has converged.
- Uses:
  - appropriate if the world is changing.

EXPLORATION STRATEGY 3: BOLTZMANN

- In state s, choose action a with probability p:

$$p = \frac{e^{\frac{Q(s,a)}{t}}}{\sum_{a'} e^{\frac{Q(s,a')}{t}}}$$

- Simulated annealing: *t* is a "temperature"
- High temperature means more exploration
- Over time, t cools, reducing exploration
- Sensitive to cooling schedule.

EXPLORATION STRATEGY 4: R-MAX

- Initialize reward for each state to R_{max}, the largest reward possible
- Keep track of the number of times each state has been visited
- After c visits, mark the state as known and update and update reward and transition probabilities.
- Need to know R_{max}

MODEL-BASED RL: PROS AND CONS

- **Pros:**
  - Makes maximal use of experience
  - Solves model optimally, given enough experience
- **Cons:**
  - Assumes model is small enough to solve
  - Requires expensive solution procedure

NEXT TIME: Q-LEARNING

- Q-learning uses the Q-function version of the Bellman equations:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a \in A}[Q^*(s',a')], \forall s,a$$

- Q-learning dispenses with learning a model, but tries to learn the Q function directly from which you can read off the policy.

- It is appropriate to use when the model is too large to solve or learn.

BELLMAN EQUATIONS

- The planning problem for an MDP is:

$$\pi^* \in \arg\max_{\pi} V^{\pi}(s)$$

- Bellman equations:

$$V^*(s) = \max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s')\right], \forall s$$

- All MDP algorithms use some variant of the Bellman equations.

BELLMAN EQUATIONS USING Q-FUNCTION

- The planning problem for an MDP is:

$$\pi^* \in \arg\max_{\pi} Q^{\pi}(s,a)$$

- Q-Function version of Bellman equations:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a \in A}[Q^*(s',a')], \forall s, a$$