# Model Based RL and Exploration in RL/MDPs

CSCI 2951-F
Ron Parr
Brown Univeristy

## Overview

- How/why model based RL?
- Small(ish) state spaces
  - How are models learned/used?
  - Exploration in small(ish) state spaces
- Large state spaces
  - How are models learned/used
  - How is exploration handled?

## How to learn a model?

- Similar approach to bandits
- Bandit: Each machine is a binomial (2 outcomes, win or lose)
- MDP: Each (state,action) is a multinomial (outcomes = next states)
- Generic approaches (w/ no assumptions about form of model):
  - Maximum likelihood
  - Bayesian with Dirichlet prior
    - Generalization of beta prior
    - Prior = hallucinated previous transitions
    - Same pros and cons as beta prior for bandits
- Model fitting
  - Use prior knowledge that not every state is reachable from every state

## Why model based RL?

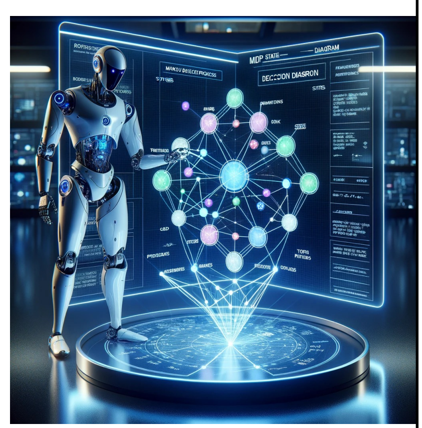
- Could learn the MDP, then solve the MDP, but…
- If we can learn Q-functions directly, why bother learning model?
  - Q-functions store SxA numbers
  - Model stores SxSxA numbers + size of R
  - Could it be more data efficient to learn the model?
    - Possibly (depends on how you look at it – see bounds later in the slide deck)
    - Definitely if you have inductive bias that makes model fitting practical
      - e.g., you know that you are in a grid world and all actions behave under a shared noise model
      - e.g., environment has some known parametric model with fewer parameters than states

## Digression: Model learning in deep RL

- Q: Can you use a deep network to learn a model?
- A: Yes!
- How: Train model to predict next states given current state, action
- Why:
  - Can be a useful auxiliary task
  - Can be used for data efficient planning if:
    - Model has a low-dimensional ”latent state” that can be learned
    - Planning can be done efficiently in the latent state space – see *Dreamer*
- Back to small state spaces for a while…

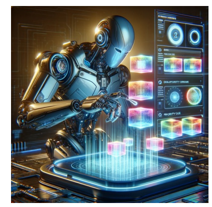
## Interleaving updates and actions

- Silly(?) approach:
  - Act randomly until you’ve tried every (s,a) enough
  - Solve the learned model
- Smarter approaches
  - Update model as you go
  - Use model in some smart way to choose actions
- But how do we update the MDP solution?
  - Re-solve the entire MDP after each new experience
  - Asynchronous updates



# Asynchronous value function updates

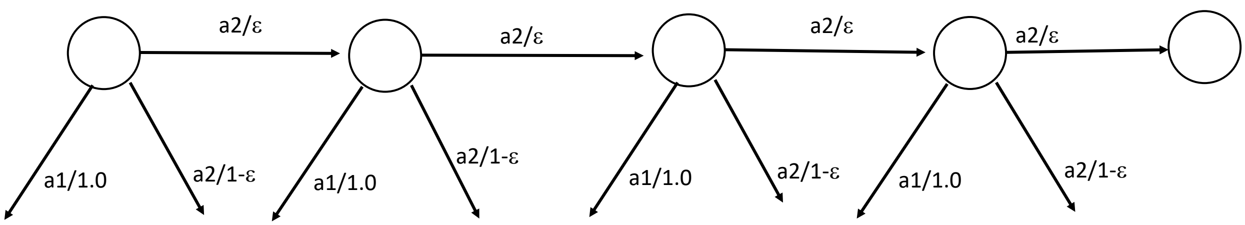
- Value iteration/determination operate synchronously using a model
  - State values at iteration i are fixed
  - Used to create a totally new set of state values at iteration i+1
- RL updates one state at time based upon an observed transition
- Anything between these two extremes also works (with mild assumptions)
- Update strategies while learning a model:
  - Update only the state for which the transition model has changed
  - Update some additional randomly selected states
  - Use a priority queue to track states with values that are most out of date
    - e.g. If state s jumps in value, other states w/transitions to s need updating too.
    - Called “prioritized sweeping”, extended to “prioritized replay” for replay buffers (DQN)



# Exploration enters the conversation

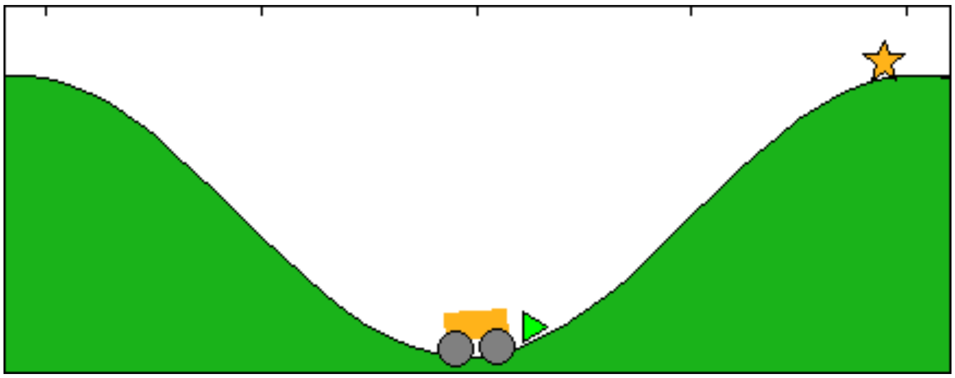
# How to think about exploration

- Similar to pulling arms in a bandit problem, but
- Events can be rare in multiple ways
- Individual transitions can have low probability
- Low(ish) probability events can chain to create even lower probabilities
- Even without domain randomness, randomly choosing actions can lead to low probabilities of reaching distant states



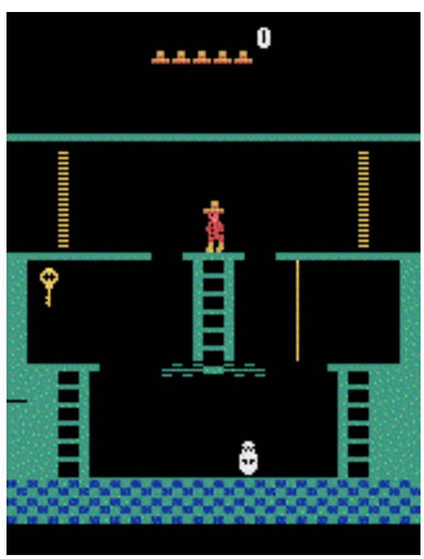

# Narrow passages

# Hard-to-reach places (narrow passages)



https://en.wikipedia.org/wiki/Mountain_car_problem

# Montezuma's Revenge (long sequences)




Source: https://gymnasium.farama.org/v0.27.1/environments/atari/montezuma_revenge/

# Questions about exploration

- How do you know when you are "done" exploring?
- What is the right way to think about the (opportunity) cost of exploring?
- If we need to take millions of actions to discover the good parts of the state space, is it wrong to take short term reward instead?
- In very large state spaces, it is impossible to visit every state, so how do we think about exploration?

# Approaches to exploration

- Ignore it and assume that randomness takes care of things
  - Pro: Easy
  - Con: May not work
- $\varepsilon$-greedy
  - Pro: Easy to implement, favors "good" actions
  - Con: Picking $\varepsilon$, hard to prove that it balances exploration vs. exploitation
- Boltzman exploration (softmax with adjustable temperature)
  - Pro: More nuanced than $\varepsilon$-greedy
  - Cons: Hard to pick $\lambda$, mostly same issues as a $\varepsilon$-greedy
- Ad-hoc exploration bonuses based on how many times an action has been tried in a state
  - Pro: Easy to implement for discrete MDPs, biases exploration towards "new" states
  - Con: Not clear how to pick bonuses, hard to prove anything
- Optimism in the face of uncertainty
  - Assume all states (stat-action pairs) we haven't tried are really good
  - Rely upon RL updates to prove us wrong
- Bayesian exploration

## Balancing exploration vs. exploitation

- PAC optimal RL (PAC-MDP)
  - Borrows ideas from computational learning theory, PAC bandits
  - WHP bounds on the number of steps in which we make "bad" decisions
- Bayesian approaches
  - Solve exploration MDP as a MDP
  - Generalize Thompson sampling from bandits

## PAC MDP optimality criterion

- **P**robably **A**pproximately Correct
- Motivation:
  - With real data, can't guarantee that any finite sample sees all or enough of the data/state space to get things right
  - For a finite sample of data, we can only hope to get close to the truth, not an exact estimate of real numbers
- We aim to get within $\varepsilon$ of best/correct answer
- With probability $1-\delta$
- Goal: Scaling (computation, sample complexity) in $1/\varepsilon$, $1/\delta$

## What does PAC-MDP mean

- Application of PAC concepts to RL
- With probability 1-$\delta$, algorithm makes a bounded number of steps that are $\varepsilon$ worse than optimal
- Amount of computation, number of suboptimal steps should scale polynomially with |S|, |A|, 1/$\varepsilon$, 1/$\delta$, and 1/(1-$\gamma$)

## What does $\varepsilon$ worse than optimal mean?

$$V^{\mathcal{A}_t}(s_t) < V^*(s_t) - \varepsilon$$

- Considers the value of whatever policy you are following at time t ($\mathcal{A}_t$)
- (Because policy is constantly changing as you learn)
- Could be arbitrarily bad – all $\varepsilon$ worse actions count the same
- Also, could be inconsequential if you take an action from a stupid policy but change policies before you experience the consequences
- Disconnects somewhat from actual rewards accrued

# The R-Max algorithm

- Model based RL
- Initially assumes all states-actions pairs have max possible Q-value ($V_{max}$)
- Always act greedily WRT current model, value function
- When a new state has at least m samples of (s,a)
  - Estimate P(s'|s,a)
  - Recompute Q-values for entire model
- Choose m high enough so that you have "enough" experiences in each state for P(s'|s,a) to be close to correct (m is a messy, but polynomial, function of # of states, # of actions, $1/\delta$, $1/\varepsilon$, and $V_{max}$)
- Resulting policy always draws you towards unexplored states

# Intuition for why R-max works

- If m is "large", and all states are "known", then your model is approximately correct, and your policy will be close to optimal for the real MDP
- If not all states states are known, then either:
  - Your policy will take actions to reach to unknown states, eventually making them "known", or
  - You don't care because you can achieve close to the highest value possible without leaving the "known states"
- But how efficient is it?

# R-Max sample complexity

$$\tilde{O}(S^2A/(\varepsilon^3(1-\gamma)^6))$$

- Ignores log factors
- $S^2A$ shouldn't be surprising – size of transition matrices
- $\varepsilon^3(1-\gamma)^6$ isn't great

# A model-free approach: Delayed Q-learning

BROWN

- Initialize all Q-values to max possible value
- Always act greedily WRT Q-functions
- Delay updating Q-functions unless:
  - A state has had m (different m from Rmax) new (s,a) experiences
  - The update is sufficiently large
- Stop making updates if no significant changes
- Also adds a small "exploration" bonus for states with values that haven't (approximately) converged yet

- Turns Q-learning into a semi-batch method

https://www.jmlr.org/papers/volume10/strehl09a/strehl09a.pdf

## Delayed Q-learning sample complexity

$$\tilde{O}(SA/(\varepsilon^4(1-\gamma)^8))$$

- Ignores log factors
- SA shouldn't be surprising – size of Q-functions
- $\varepsilon^4(1-\gamma)^8$ is eye-watering

## Are these bounds bad?

- They might not be tight in all variables
- Can't avoid trying every state and action "enough" times to figure out if it's worthwhile
- What if we had prior knowledge about (some) state values?

## Bayesian RL

- Maintain a probability distribution over (models)MDPs
- Sample models, choose actions that are optimal WRT to sample
- (Similar to Thompson sampling)
- Tricky in practice – space of RL models is not easy to manage

- Not the focus of this lecture

## Admissible heuristics

- For PAC MDPs h(x) is admissible if it never underestimates the true value

- Concept taken from heuristic search, e.g., A*, but reversed

- Standard R-Max and Delayed Q are admissible, but what if we have extra knowledge?

## Bounds with admissible heuristics

- R-max

$$\tilde{O}\left(\frac{V_{\max}^3 S|\{(s,a)\in\mathcal{S}\times\mathsf{A}|U(s,a)\geq V^*(s)-\varepsilon\}|}{\varepsilon^3(1-\gamma)^3}\right)$$

- Delayed Q

$$\tilde{O}\left(\frac{V_{\max}^3\sum_{(s,a)\in\mathcal{S}\times\mathsf{A}}[U(s,a)-V^*(s)]_+}{\varepsilon^4(1-\gamma)^4}\right)$$

Here, *admissible* means $Q^*(s,a) \leq U(s,a) \leq V_{max}$

## What's the best we can hope to do?

- Two bounds:

$$\Omega\left(\frac{SA}{\varepsilon(1-\gamma)^2}\ln\frac{1}{\delta}\right)$$

- Improved (in most cases) to:

$$\Omega\left(\frac{SA}{\varepsilon^2}\ln\frac{S}{\delta}\right)$$

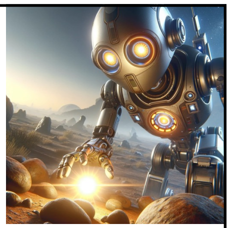
## A Bayesian Approach

- Recall how to solve a bandit as an MDP
  - State = #w,#l for each arm (win probability for each arm)
  - beta prior implies distribution over next states
  - Solve MDP in this state space to find optimal exploration strategy
- Generalization to MDPs
  - State is current Dirichlet distribution parameters for all states
  - Implies distribution over next states (next MDP parameters)
  - Could solve this as a huge MDP (but you don't want to!)

## Thompson sampling for MDPs

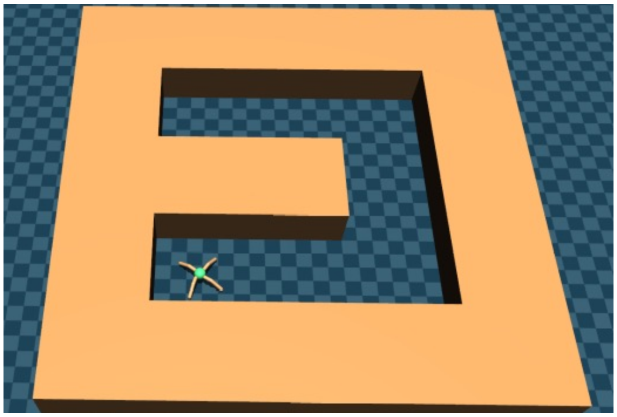
- Active research area
- General approach:
  - Maintain a distribution over MDPs
  - Sample an MDP from this distribution
  - Compute the optimal policy for the MDP and act in it for a while, collecting data
  - Update distribution
  - Repeat until some convergence condition
- Recent results have emphasized regret bounds in finite horizon MDPs

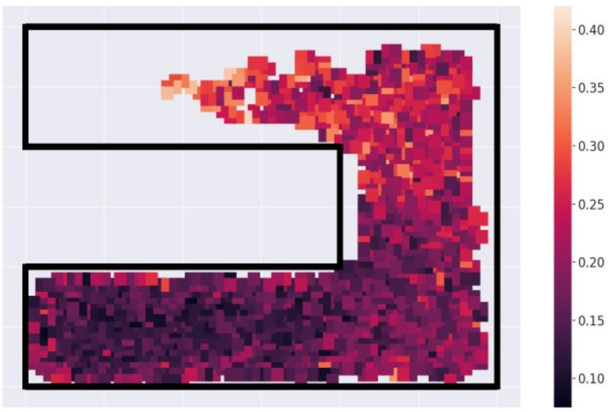
# Exploration in large state spaces



- Methods used so far rely upon collecting data for every (s,a)
- For large state spaces:
  - Too many states
  - May never visit the same (s,a) twice (if continuous)
  - Rely upon function approximation (e.g. DQN)
- Idea:
  - Approximate behavior of R-max by giving an exploration bonus to states that have not been visited much (supplementary reward)
  - Attenuate the bonus as states are visited more
  - "Pseudo counts"
  - Challenge: How do you estimate this for large state spaces?

# Exploration bonus example






Images courtesy of Sam Lobel

# Implementing pseudo counts

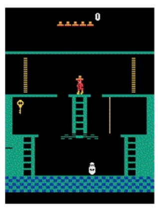
- Goal: (Approximately) estimate how often an agent has been in a state (lumping together "similar" states)
- Challenges:
  - Input space can be enormous (image space)
  - Chicken and egg problem – we may not know what relevant similarities are until we've solved the problem, but can't solve the problem w/o expoloration
- One family of approaches:
  - Measure similarity in a lower dimensional and/or discreteized space such as
  - Bottleneck in the NN (last layer, or embedding space depending upon architecture)
  - Some hash or projection of the input space

# Indirect approaches

- Use neural network to ***indirectly*** estimate state visitation frequency
- Random Network Distillation [Burda et al.]
  - Initialize a random neural network $f(s,\theta)$ – fixed weights, not trained
  - Train a new network $g(s,\omega)$ starting from a different random initialization to match $f(s,\theta)$ – one gradient step every time we visit s
  - Exploration bonus increases with difference between $| f(s,\theta)- g(s,\omega) |$
  - Intuition: As we get more experience "around" s, networks should converge
  - Works strangely well despite some obvious concerns about representation, local optima, etc.
- Coin flipping network [Lobel et al.]
  - Train a network with state s as input, uniformly randomly selected +1/-1 target
  - Ideally, network will converge to 0 for all states
  - Distance from 0 is indication of state novelty
  - More justifiable and (often) works better than RND

BROWN

Couresy of Sam Lobel


$f^*_\theta($  $) = \overline{\underline{1 \qquad 0}} \xrightarrow{\mathrm{RMS}(\|v\|)} \frac{1}{\sqrt{2}}$

## What do you do in practice?

- ε-greedy still the most popular form of exploration
- Methods like RND and CFN that are motivated by theoretical results on small state spaces getting increasing traction