Deep Reinforcement Learning

Lecture 4: Passive Learning and Model-Based Active Learning

Instructor: Chongjie Zhang

Tsinghua University

(With some slides from Dan Klein and Sergey Levine)

So far ….

- Given an MDP model we know how to find optimal policies (for moderately-sized MDPs)
  - Value Iteration or Policy Iteration

- What if we don’t have a model or simulator?
  - Like when we were babies . . .
  - Like in many real-world applications
  - All we can do is wander around the world observing what happens, getting rewarded and punished

- Enters reinforcement learning

Outline

- Introduction

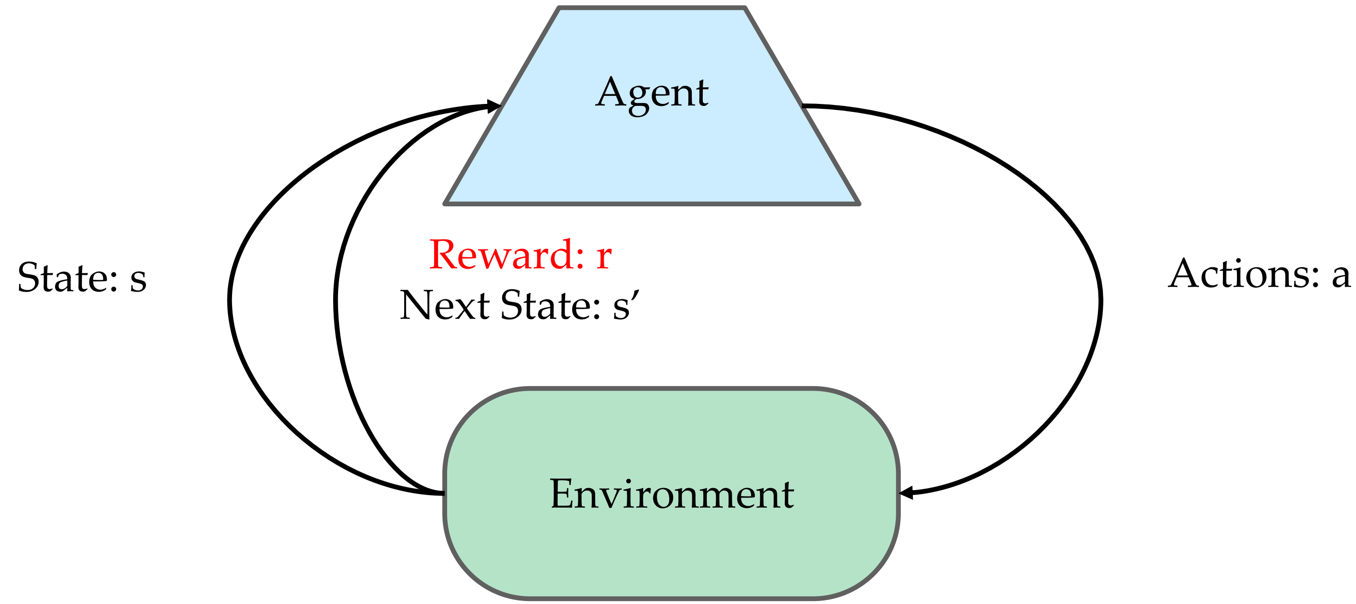
- Passive learning
  - Monte Carlo direct estimation (model-free)
  - Adaptive dynamic programming (ADP) (model-based)
  - Temporal difference (TD) learning (model-free)

- Active learning
  - ADP-based learning (model-based)
  - TD-based learning
  - Q-learning (model-free)

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s)
- Still looking for a policy $\pi(s)$
- New twist: don't know T or R
  - i.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

Reinforcement Learning



- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

Example: Learning to Walk

Initial

A Learning Trial

After Learning [1K Trials]

[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

Training

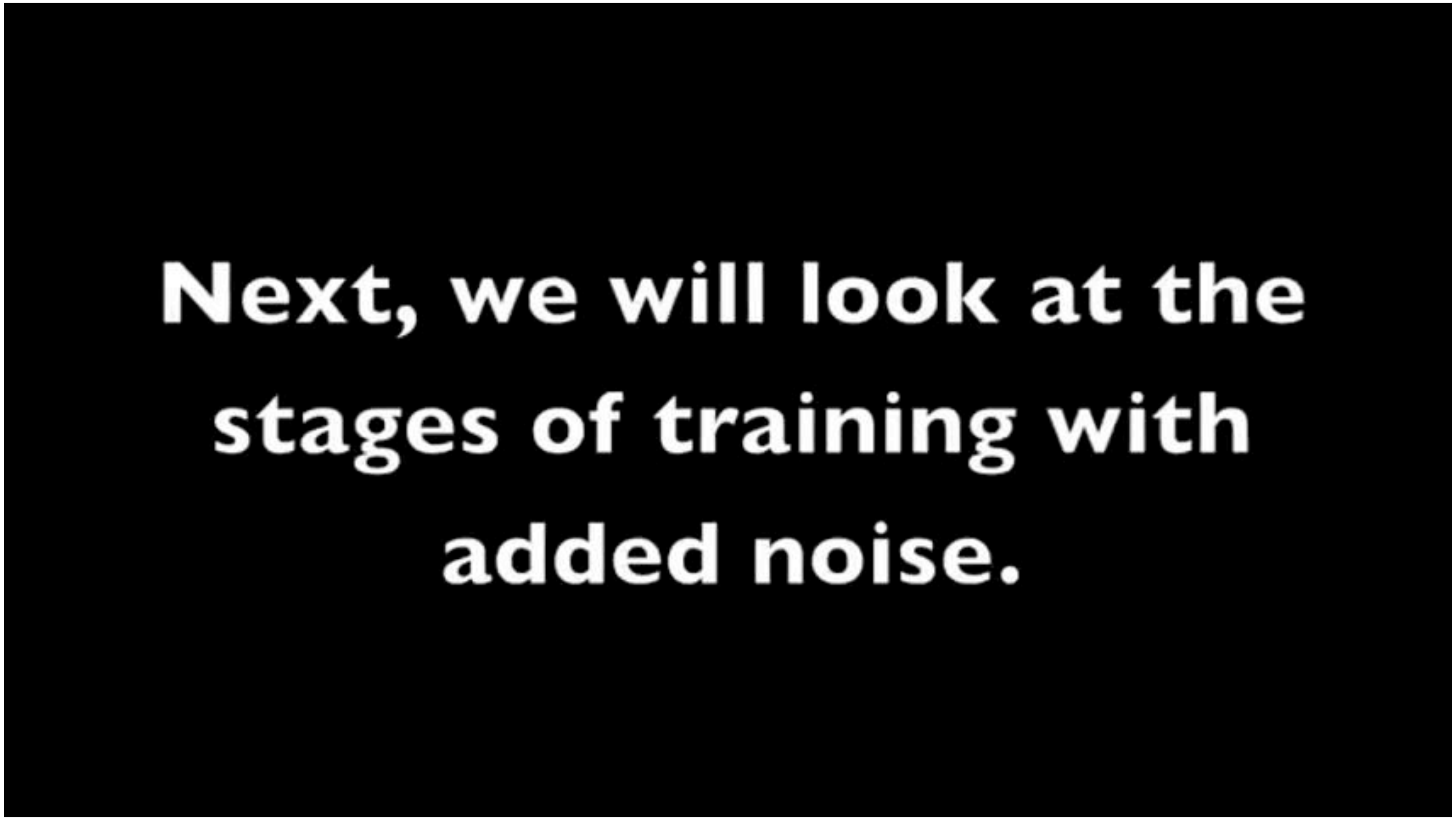
[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

Learning for Robotic Manipulation

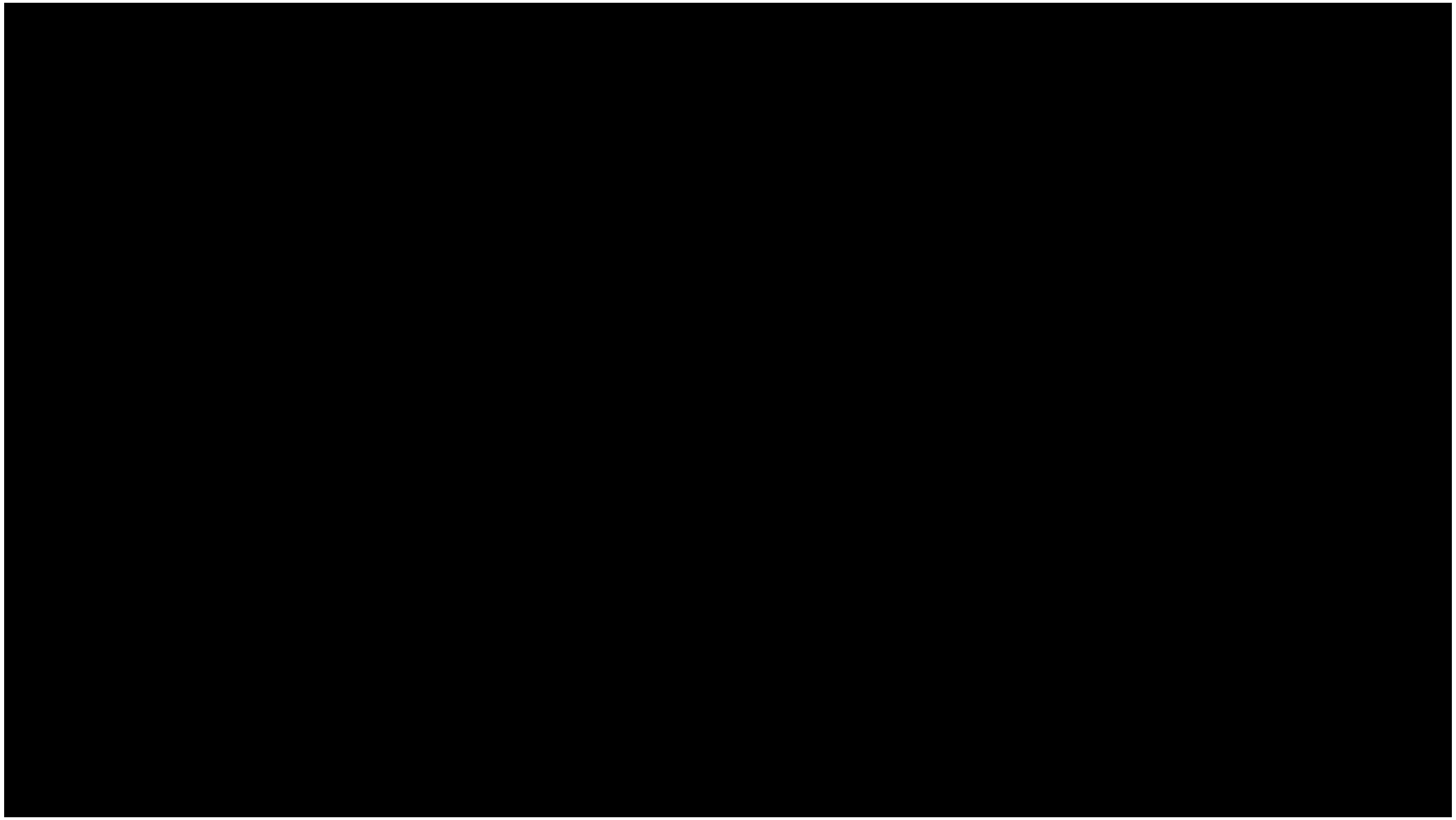


[Gu et al., ICRA 2017]

DeepMind Atari (©Two Minute Lectures)

[Minh et al., Nature 2017]

AlphaGo Zero: Start from Scratch



[Silver et al., Nature 2017]

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s)
- Still looking for a policy $\pi(s)$
- New twist: don't know T or R
  - i.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

Passive vs. Active Learning

- Passive learning
  - The agent has a fixed policy and tries to learn the utilities of states by observing the world go by
  - Analogous to policy evaluation
  - Often serves as a component of active learning algorithms
  - Often inspires active learning algorithms
- Active learning
  - The agent attempts to find an optimal (or at least good) policy by acting in the world
  - Analogous to solving the underlying MDP, but without first being given the MDP model

Model-Based vs. Model-Free RL

- Model-based approach to RL:
  - learn the MDP model, or an approximation of it
  - use it for policy evaluation or to find the optimal policy
- Model-free approach to RL:
  - derive the optimal policy without explicitly learning the model
  - useful when model is difficult to represent and/or learn
- We will consider both types of approaches

Small vs. Huge MDPs

- We will first cover RL methods for small MDPs
  - MDPs where the number of states and actions is reasonably small
  - These algorithms will inspire more advanced methods

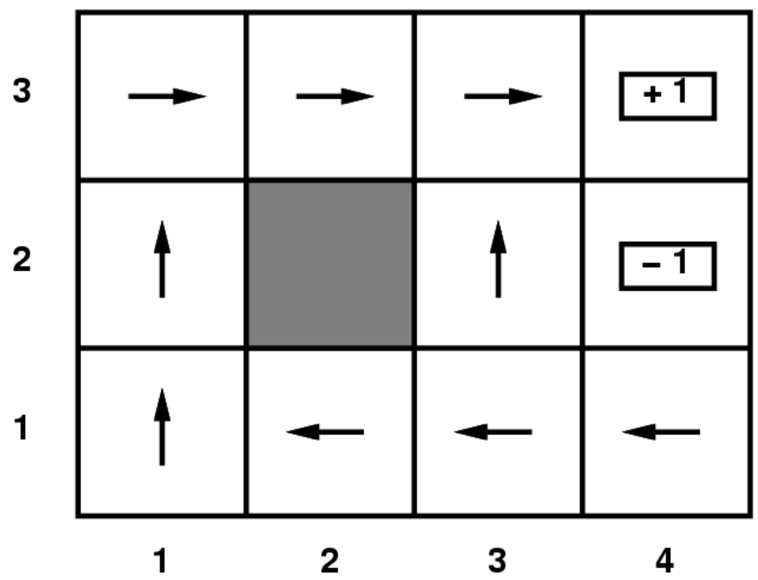
- Later we will cover algorithms for huge MDPs
  - Function Approximation Methods
  - Policy Gradient Methods
  - Actor-Critic Methods

Outline

- Introduction
- **Passive learning**
  - Monte Carlo direct estimation (model-free)
  - Adaptive dynamic programming (ADP) (model-based)
  - Temporal difference (TD) learning (model-free)
- Active learning
  - ADP-based learning (model-based)
  - TD-based learning
  - Q-learning (model-free)

Example: Passive RL

- Suppose given a stationary policy (shown by arrows)
  - Actions can stochastically lead to unintended grid cell
- Want to determine how good it is



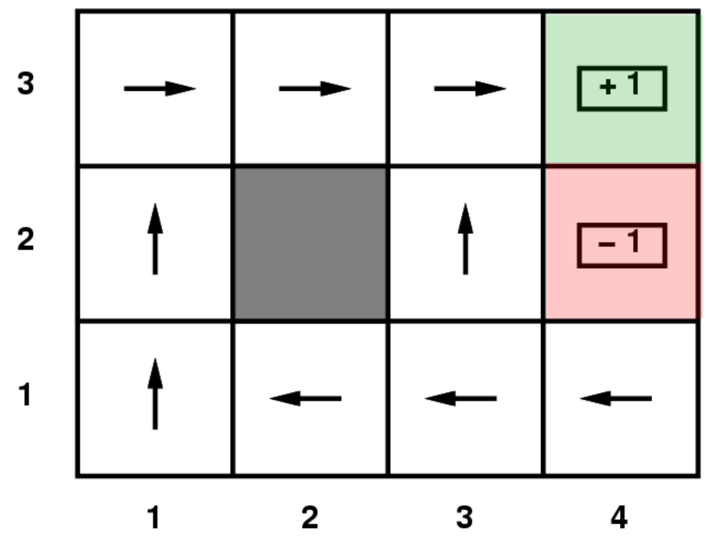
Objective: Value Function

3	0.812	0.868	0.918	+ 1
2	0.762		0.660	– 1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Passive RL

- Estimate $V^{\pi}(s)$
- Not given
  - transition matrix
  - reward function!
- Follow the policy for many epsidoes giving training sequences.

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1



- Assume that after entering +1 or -1 state the agent enters zero reward terminal state
  - So we don't bother showing those transitions

Approach 1: Direct Estimation

- Direct estimation (also called Monte Carlo)
  - Estimate $V^{\pi}(s)$ as average total reward of episodes containing s (calculating from s to end of epoch)
- ***Reward to go*** of a state s
  - the sum of the (discounted) rewards from that state until a terminal state is reached
- Key: use observed ***reward to go*** of the state as the direct evidence of the actual expected utility of that state
- Averaging the reward-to-go samples will converge to true value at state

Direct Estimation

- Converge very slowly to correct utilities values (requires a lot of sequences)

- Doesn't exploit Bellman constraints on policy values

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, a, s') V^{\pi}(s')$$

  - It is happy to consider value function estimates that violate this property badly.

How can we incorporate the Bellman constraints?

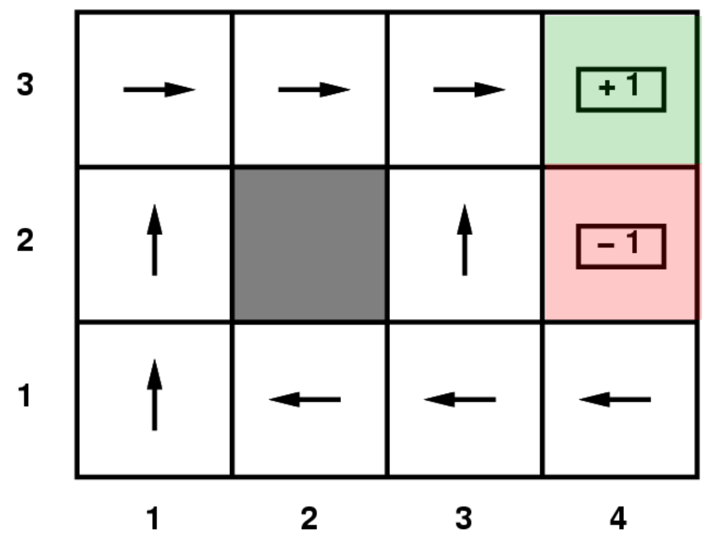
Approach 2: Adaptive Dynamic Programming (ADP)

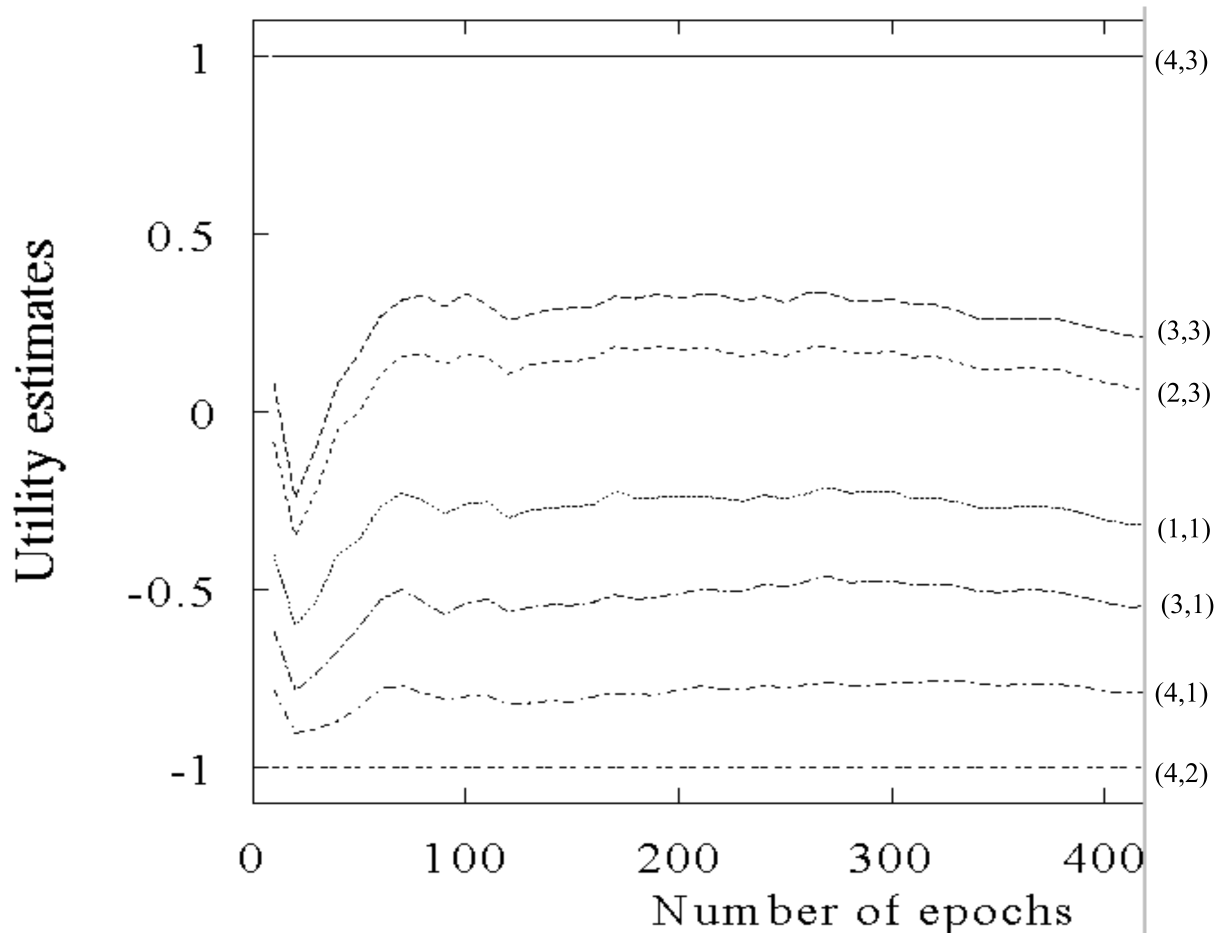
- ADP is a model based approach
  - Follow the policy for a while
  - Estimate transition model based on observations
  - Learn reward function
  - Use estimated model to compute utility of policy

$$V^{\pi}(s) = \boxed{R(s)} + \gamma \sum_{s'} \boxed{T(s,a,s')} V^{\pi}(s')$$

- How can we estimate transition model T(s, a, s')?
  - Simply the fraction of times we see s' after taking a in state s
  - NOTE: Can bound error with Chernoff bounds if we want

ADP learning curves





Approach 2: Adaptive Dynamic Programming (ADP)

- ADP is a model based approach
  - Follow the policy for a while
  - Estimate transition model based on observations
  - Learn reward function
  - **Use estimated model to compute utility of policy**

$$V^{\pi}(s) = \boxed{R(s)} + \gamma \sum_{s'} \boxed{T(s,a,s')} V^{\pi}(s')$$

- Can we avoid the computational expense of full DP policy evaluation?

Approach 3: Temporal Difference Learning (TD)

- Temporal Difference Learning (model-free)
  - Do local updates of utility/value function on a per-action basis
  - Don’t try to estimate entire transition function!
  - For each transition from s to s’, we perform the following update:

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(R(s) + \gamma V^{\pi}(s') - V^{\pi}(s))$$

updated estimate learning rate discount factor

- Intuitively moves us closer to satisfying Bellman constraint

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, a, s') V^{\pi}(s')$$

$\pi(s)$

Why?

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers ($x_1, x_2, x_3, \ldots$)
  - E.g., to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1} \sum_{i=1}^{n+1} x_i$$

average of n+1 samples

- Given a new sample x_{n+1}, the new mean is the old estimate (for n samples) plus the weighted difference between the new sample and the old estimate

Approach 3: Temporal Difference Learning (TD)

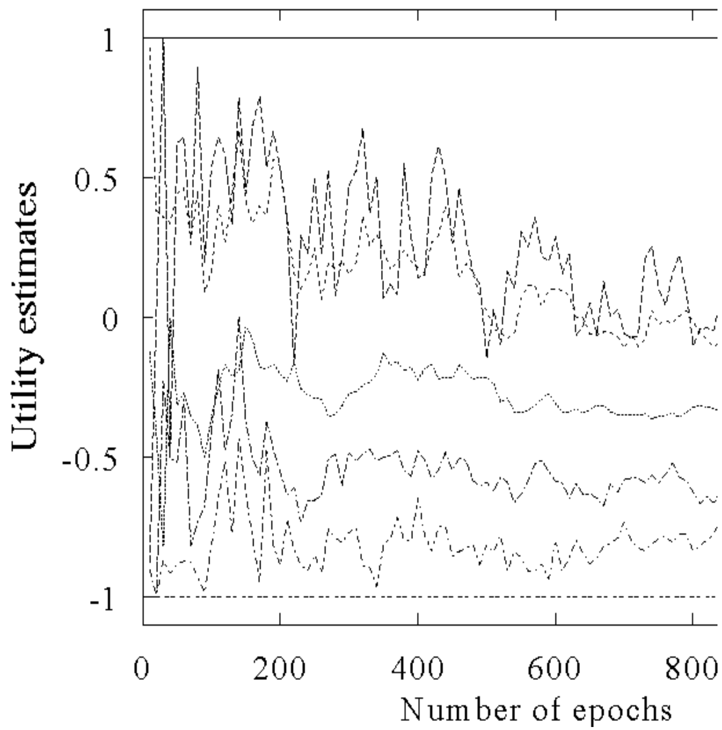
- TD update for transition from s to s':

$$\underbrace{V^{\pi}(s)}_{\text{updated estimate}} \leftarrow V^{\pi}(s) + \underbrace{\alpha}_{\text{learning rate}}(\underbrace{R(s) + \gamma V^{\pi}(s')}_{\text{(noisy) sample of value at s based on next state s'}} - V^{\pi}(s))$$

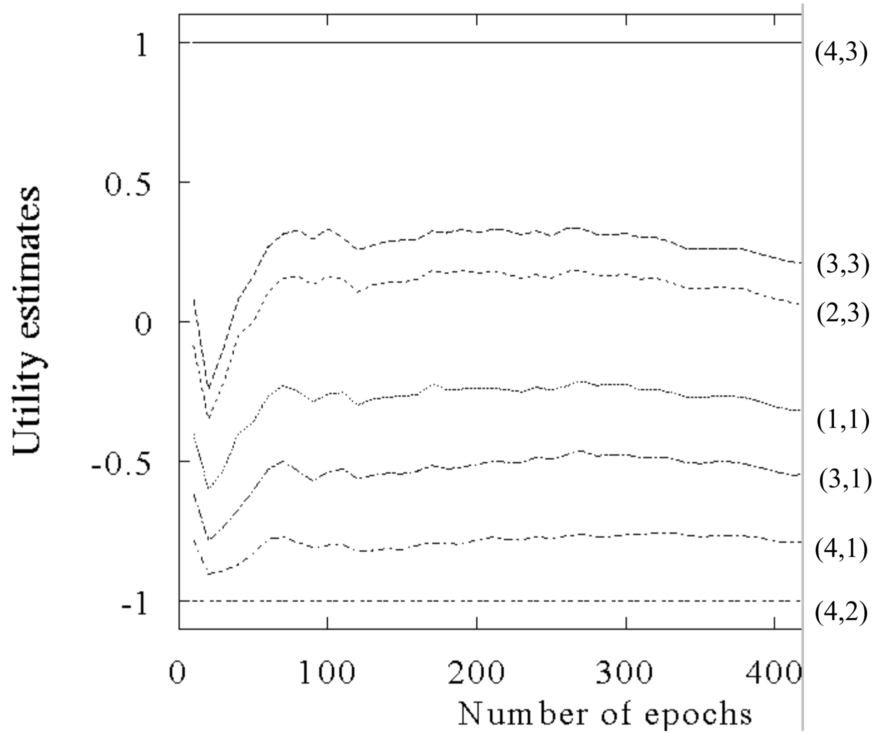
- So the update is maintaining a "mean" of the (noisy) value samples
- If the learning rate decreases appropriately with the number of samples (e.g. 1/n) then the value estimates will converge to true values! (non-trivial)

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s,a,s') V^{\pi}(s')$$

TD learning curve



ADP learning curve



- **Tradeoff:** requires more training experience (epochs) than ADP but much less computation per epoch
- Choice depends on relative cost of experience vs. computation

Passive RL: Comparisons

- Monte-Carlo Direct Estimation (model-free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Adaptive Dynamic Programming (model-based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Temporal Difference Learning (model-free)
  - Update speed and implementation similiar to direct estimation
  - Partially exploits Bellman constraints---adjusts state to ‘agree’ with observed successor
    - Not ***all*** possible successors as in ADP
  - Convergence speed in between direct estimation and ADP

Between ADP and TD

- Moving TD toward ADP
  - At each step perform TD updates based on observed transition and “imagined” transitions
  - Imagined transition are generated using estimated model
- The more imagined transitions used, the more like ADP
  - Making estimate more consistent with next state distribution
  - Converges in the limit of infinite imagined transitions to ADP
- Trade-off computational and experience efficiency
  - More imagined transitions require more time per step, but fewer steps of actual experience

Active Reinforcement Learning

- So far, we’ve assumed agent ***has*** a policy
  - We just learned how good it is
- Now, suppose agent must learn a good policy (ideally optimal)
  - While acting in uncertain world

Outline

- Introduction
- Passive learning
  - Monte Carlo direct estimation (model-free)
  - Adaptive dynamic programming (ADP) (model-based)
  - Temporal difference (TD) learning (model-free)
- **Active learning**
  - ADP-based learning (model-based)
  - TD-based learning
  - Q-learning (model-free)

Naïve Model-Based Approach

1. Act Randomly for a (long) time
   - Or systematically explore all possible actions
2. Learn
   - Transition function
   - Reward function
3. Use value iteration, policy iteration, …
4. Follow resulting policy thereafter.

Will this work? Yes (if we do step 1 long enough and there are no “dead-ends”)

Any problems? We will act randomly for a long time before **exploiting** what we know.

Revision of Naïve Approach

1. Start with an initial (uninformed) model
2. Solve for the optimal policy given the current model
   (using value or policy iteration)
3. Execute an action suggested by the policy in the current state
4. Update the estimated model based on the observed transition
5. Goto 2

This is just like ADP but we follow the greedy policy suggested by current value estimate

Will this work? No. Can get stuck in local minima.
What can be done?

Exploration vs. Exploitation

- Two reasons to take an action in RL
  - **<u>Exploitation</u>**: To try to get reward. We exploit our current knowledge to get a payoff.
  - **<u>Exploration</u>**: Get more information about the world. How do we know if there is not a pot of gold around the corner.
- To explore we typically need to take actions that do not seem best according to our current model
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic intuition behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

ADP-based (model-based) RL

1. Start with initial model
2. Solve for optimal policy given current model
(using value or policy iteration)
3. Take action according to an exploration/exploitation policy
(explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the exploration/exploitation policy

Will this work? Depends on the explore/exploit policy.
Any ideas?

Explore/Exploit Policies

- Greedy action is the action maximizing estimated Q-value

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s')V(s')$$

  - where V is current optimal value function estimate (based on current model), and R, T are current estimates of model
  - Q(s, a) is the expected value of taking action a in state s and then getting the estimated value V(s') of the next state s'
- Want an exploration policy that is greedy in the limit of infinite exploration (GLIE)
  - Guarantees convergence
- GLIE Policy 1
  - On time step t select random action with probability p(t) and greedy action with probability 1-p(t)
  - p(t) = 1/t will lead to convergence, but is slow

Explore/Exploit Policies

- GLIE Policy 1
  - On time step t select random action with probability p(t) and greedy action with probability 1-p(t)
  - p(t) = 1/t will lead to convergence, but is slow

- In practice it is common to simply set p(t) to a small constant ε (e.g. ε=0.1 or ε=0.01)
  - Called ε-greedy exploration

Explore/Exploit Policies

- GLIE Policy 2: Boltzmann Exploration
  - Select action a with probability,

$$\Pr(a \mid s) = \frac{\exp\big(Q(s,a)/T\big)}{\sum_{a' \in A} \exp\big(Q(s,a')/T\big)}$$

  - T is the temperature. Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically start with large T and decrease with time

The Impact of Temperature

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

- Suppose we have two actions and that $Q(s, a_1) = 1$, $Q(s, a_2) = 2$
- $T=10$ gives $\Pr(a_1 \mid s) = 0.48$, $\Pr(a_2 \mid s) = 0.52$
  - Almost equal probability, so will explore
- $T= 1$ gives $\Pr(a_1 \mid s) = 0.27$, $\Pr(a_2 \mid s) = 0.73$
  - Probabilities more skewed, so explore a_1 less
- $T = 0.25$ gives $\Pr(a_1 \mid s) = 0.02$, $\Pr(a_2 \mid s) = 0.98$
  - Almost always exploit a_2

Alternative Model-Based Approach: Optimistic Exploration

1. Start with initial model
2. Solve for "optimistic policy"
   (uses optimistic variant of value iteration)
   (inflates value of actions leading to unexplored regions)
3. Take greedy action according to optimistic policy
4. Update estimated model
5. Goto 2

Basically act as if all "unexplored" state-action pairs are maximally rewarding.

Optimistic Exploration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s')V(s')$$

- **Optimistic VI:** assigns highest possible value V^{max} to any state-action pair that has not been explored enough
  - Maximum value is when we get maximum reward forever

$$V^{\max} = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\text{max}}/(1-\gamma)$$

- What do we mean by "explored enough"?
  - $N(s,a) > N_e$, where $N(s,a)$ is number of times action a has been tried in state s and N_e is a user selected parameter

Optimistic Value Iteration

$$V(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s')V(s')$$

Standard VI

- Optimistic value iteration computes an optimistic value function V^+ using following updates

$$V^+(s) \leftarrow R(s) + \gamma \max_a \begin{cases} V^{\max}, & N(s,a) < N_e \\ \sum_{s'} T(s,a,s')V^+(s'), & N(s,a) \geq N_e \end{cases}$$

- The agent will behave initially as if there were wonderful rewards scattered all over around– **optimistic** .
- But after actions are tried enough times we will perform standard “non-optimistic” value iteration

Optimistic Exploration: Review

1. Start with initial model
2. Solve for optimistic policy using optimistic value iteration
3. Take greedy action according to optimistic policy
4. Update estimated model; Goto 2

Can any guarantees be made for the algorithm?

- If N_e is large enough and all state-action pairs are explored that many times, then the model will be accurate and lead to close to optimal policy
- But, perhaps some state-action pairs will never be explored enough or it will take a very long time to do so
- Optimistic exploration is equivalent to another algorithm, Rmax, which has been proven to efficiently converge

Another View of Optimistic Exploration: The Rmax Algorithm [Brafman & Tennenholtz, 2002]

1. Start with an optimistic model
   (assign largest possible reward to "unexplored states")
   (actions from "unexplored states" only self transition)
2. Solve for optimal policy in optimistic model (standard VI)
3. Take greedy action according to the computed policy
4. Update optimistic estimated model
   (if a state becomes "known" then use its true statistics)
5. Goto 2

Agent always acts greedily according to a model that assumes all "unexplored" states are maximally rewarding

Rmax: Optimistic Model

- Keep track of number of times a state-action pair is tried
- If $N(s, a) < N_e$ then $T(s,a,s)=1$ and $R(s) = Rmax$ in optimistic model,
- Otherwise $T(s,a,s')$ and $R(s)$ are based on estimates obtained from the N_e experiences (the estimate of true model)
  - N_e can be determined by using Chernoff Bound
- An optimal policy for this optimistic model will try to reach unexplored states (those with unexplored actions) since it can stay at those states and accumulate maximum reward
- Never explicitly explores. Is always greedy, but with respect to an optimistic outlook.

Optimistic Exploration

- Rmax is equivalent to optimistic exploration via optimistic VI
  - Convince yourself of this.
- Is Rmax provably efficient?
  - If the model is very completely learned (i.e. $N(s, a) > N_e$, for all (s,a), then the policy will be near optimal
  - Recent results show that this will happen “quickly”
- **PAC Guarantee (Roughly speaking):** *There is a value of N_e, such that with high probability the Rmax algorithm will select at most a polynomial number of actions with value less than ε of optimal)*
- RL can be solved in poly-time in num. actions, num. states, and discount factor!

Optimistic RL: Generic Algorithm

Algorithm. (for Infinite horizon RL problems)
Initialise $\hat{p}$, $\hat{r}$, and $N(s,a)$ For $t = 1, 2, \ldots$

1. Build an optimistic reward model $(\bar{Q}(s,a))_{s,a}$ from $\hat{p}$, $\hat{r}$, and $N(s,a)$
2. Select action $a(t)$ maximising $\bar{Q}(s(t),a)$ over $\mathcal{A}_{s(t)}$
3. Observe the transition to $s(t+1)$ and collect reward $r(s(t),a(t))$
4. Update $\hat{p}$, $\hat{r}$, and $N(s,a)$

Model-Based RL for Discounted Rewards: State-of-the-Art

Algorithm	Setup	Sample Complexity
R-max	–	$\widetilde{\mathcal{O}}\left(\frac{S^2 A}{(1-\lambda)^6 \varepsilon^3} \log \delta^{-1}\right)$
MBIE	–	$\widetilde{\mathcal{O}}\left(\frac{S^2 A}{(1-\lambda)^6 \varepsilon^3} \log \delta^{-1}\right)$
Delayed Q-Learning	known reward	$\widetilde{\mathcal{O}}\left(\frac{S A}{(1-\lambda)^8 \varepsilon^4} \log \delta^{-1}\right)$
MoRmax	–	$\widetilde{\mathcal{O}}\left(\frac{S A}{(1-\lambda)^6 \varepsilon^2} \log \delta^{-1}\right)$
UCRL-γ	$\lvert\mathrm{supp}(p(\cdot\lvert s,a))\rvert \le 2,\ \forall(s,a)$	$\widetilde{\mathcal{O}}\left(\frac{N}{(1-\lambda)^3 \varepsilon^2} \log \delta^{-1}\right)^*$
Lower Bound	–	$\widetilde{\Omega}\left(\frac{S A}{(1-\lambda)^3 \varepsilon^2} \log \delta^{-1}\right)$

*N denotes the number of non-zero transitions in the true MDP

$\widetilde{\mathcal{O}}(\cdot)$ hides poly-logarithmic (in S, A, ε) terms