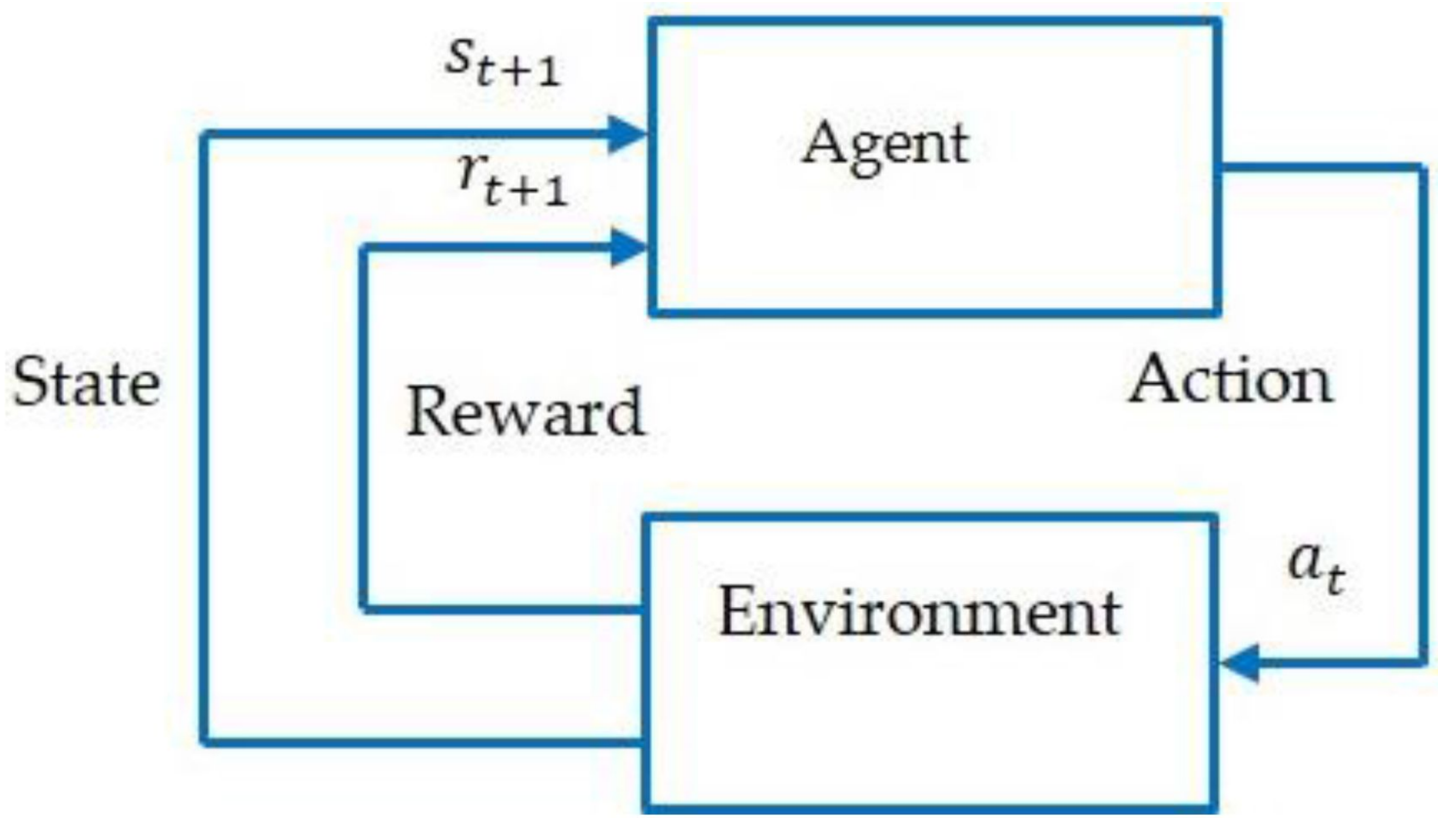
Reinforcement Learning

Emma Brunskill
Stanford University
Spring 2017

Reinforcement Learning:

Learning to make a good sequence of decisions



Policy: mapping from history of past actions, states, rewards to next action

Figure from Shttp://www.mdpi.com/energies/energies-09-00755/article_deploy/html/images/energies-09-00755-g003.png

Reinforcement Learning Involves

- Optimization
- Delayed consequences
- Generalization
- Exploration

Learning Objectives

- Define the key features of RL vs AI & other ML
- Define MDP, POMDP, bandit, batch offline RL, online RL
- Given an application problem (e.g. from computer vision, robotics, etc) decide if it should be formulated as a RL problem, if yes how to formulate, what algorithm (from class) is best suited to addressing, and justify answer
- Implement several RL algorithms incl. a deep RL approach
- Describe multiple criteria for analyzing RL algorithms and evaluate algorithms on these metrics: e.g. regret, sample complexity, computational complexity, convergence, etc.
- Describe the exploration vs exploitation challenge and compare and contrast 2 or more approaches
- List at least two open challenges or hot topics in RL

What We’ve Covered So Far

- Markov decision process planning
- Reinforcement learning in finite state and action domains
- Generalization with model-free techniques
- Exploration

Reinforcement Learning

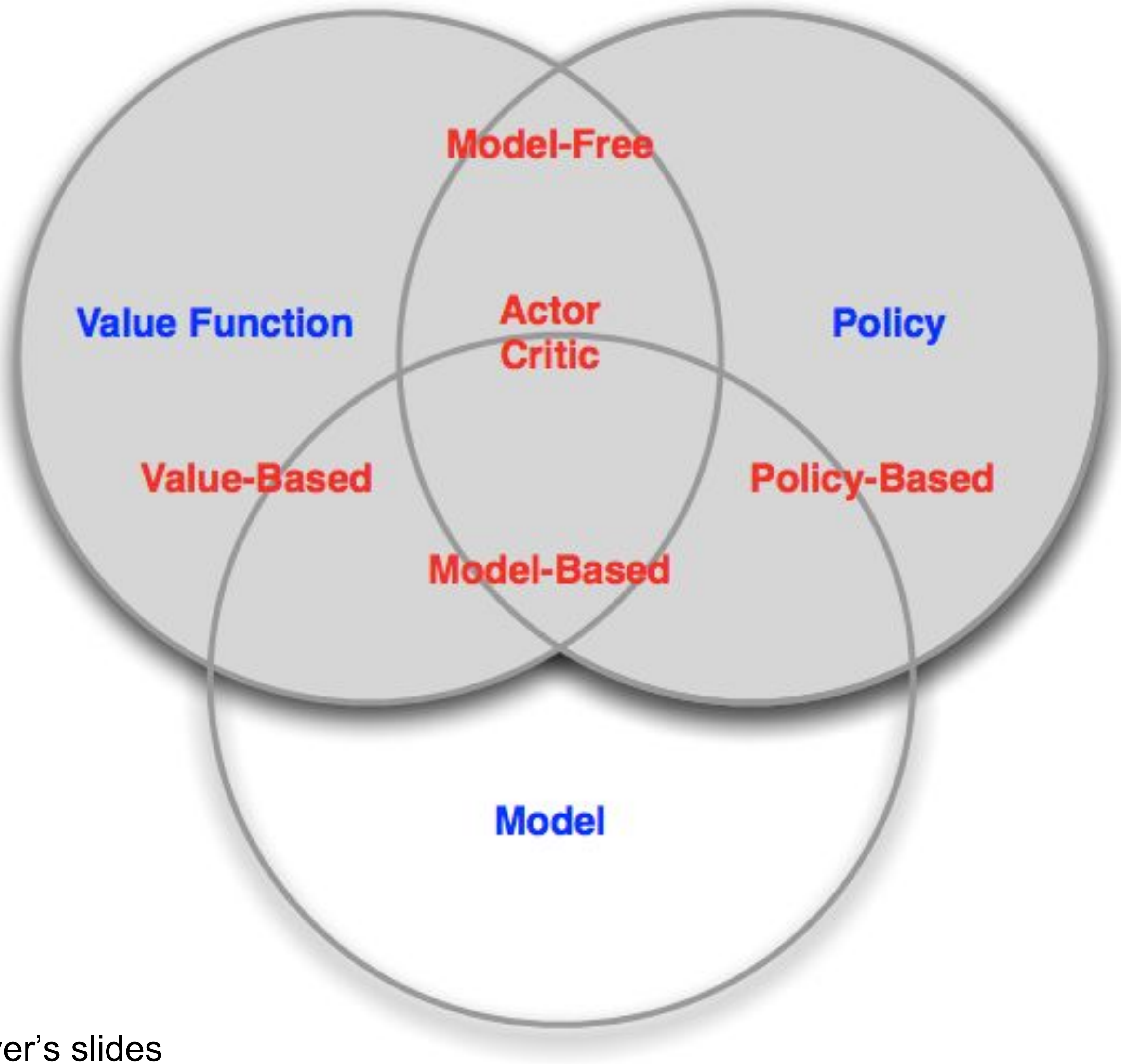


Figure from David Silver's slides

Reinforcement Learning

model ⟶ value ⟶ policy

(ordering sufficient but not necessary, e.g. having a model is not required to learn a value)

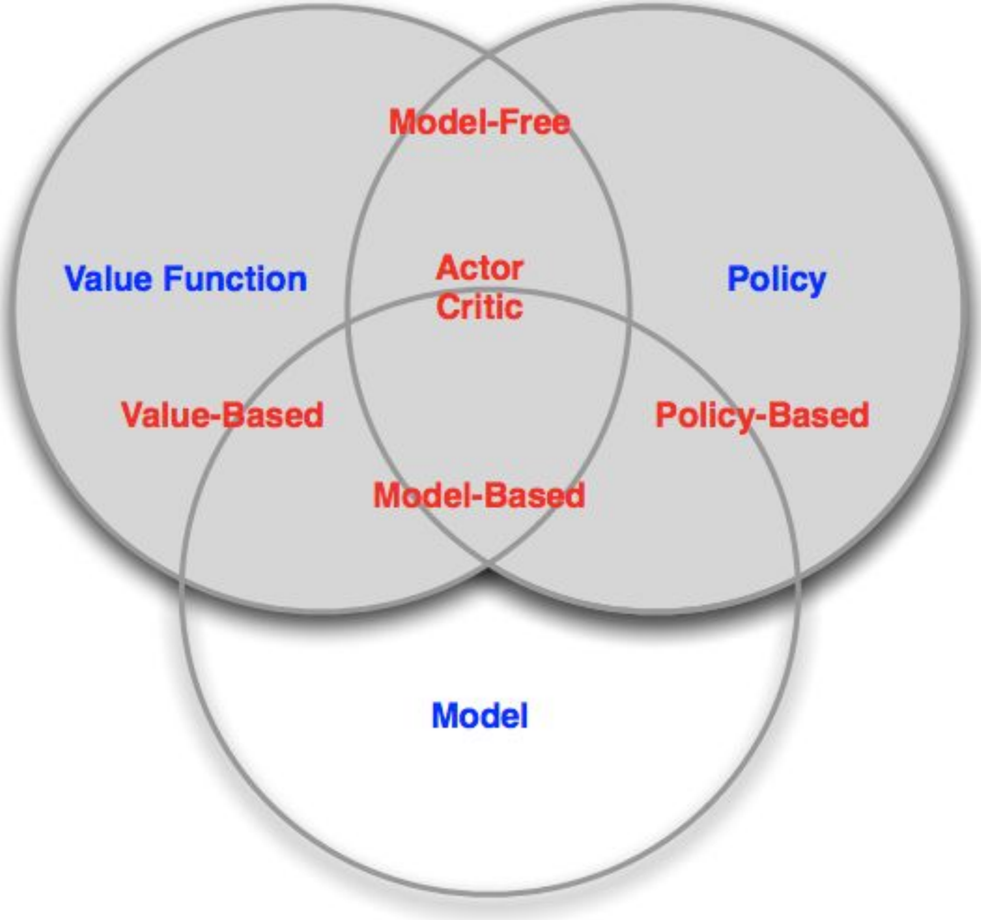
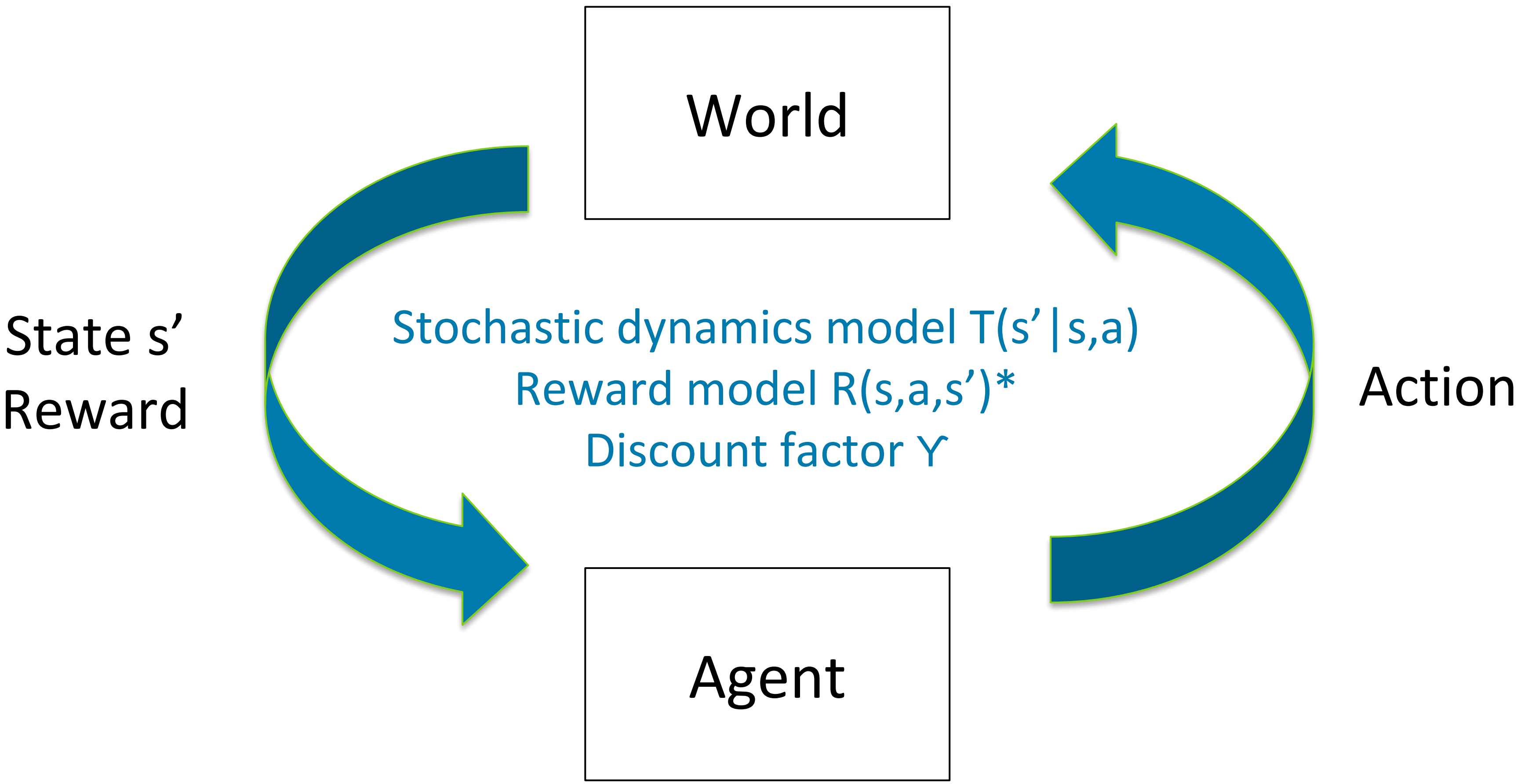


Figure from David Silver's slides

What We've Covered So Far

- Markov decision process planning
  - definition of MDP
  - Bellman operator, contraction
  - value iteration, policy iteration
- Reinforcement learning in finite state and action domains
- Generalization with model-free techniques
- Exploration

Model: Frequently model as a Markov Decision Process, <S,A,R,T,Υ>



Policy mapping from state → action

MDPs

- Define a MDP <S,A,R,T,ϒ>
- Markov property
  - What is this, why is it important
- What are the MDP models / values V / state-action values Q / policy
- What is MDP planning? What is difference from reinforcement learning?
  - Planning = know the reward & dynamics
  - Learning = don’t know reward & dynamics

Value Iteration (VI)

1. Initialize $V_0(s_i)$=0 for all states $s_{i,}$
2. Set k=1
3. Loop until [finite horizon, convergence]
   - For each state s,

$$V_{k+1}(s) = \max_a \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|a,s) V_k(s') \right]$$

4. Extract Policy

Bellman backup

Contraction Operator

- Let O be an operator
- If |OV – OV’| <= |V-V|’
- Then O is a contraction operator

- Let B be the Bellman backup operator

$$
\begin{aligned}
V_{k+1}(s) &= \max_a \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|a,s) V_k(s') \right] \\
&= BV_k
\end{aligned}
$$

Will Value Iteration Converge to a Single Fixed Point?

- Yes, if discount factor $\gamma < 1$ or end up in a terminal state with probability 1
- Bellman backup is a contraction if discount factor, $\gamma < 1$
- If apply it to two different value functions, distance between value functions shrinks after apply Bellman equation to each

Value vs Policy Iteration

- Value iteration:
  - Compute optimal value if horizon=k
    - Note this can be used to compute optimal policy if horizon = k
  - Increment k
- Policy iteration:
  - Compute infinite horizon value of a policy
  - Use to select another (better) policy
  - Closely related to a very popular method in RL: policy gradient

Policy Iteration (PI)

1. i=0; Initialize $\pi_0(s)$ randomly for all states s
2. Converged = 0;
3. While i == 0 or $|\pi_i - \pi_{i-1}| > 0$
   - i=i+1
   - Policy evaluation: Compute V^{π}
   - Policy improvement:

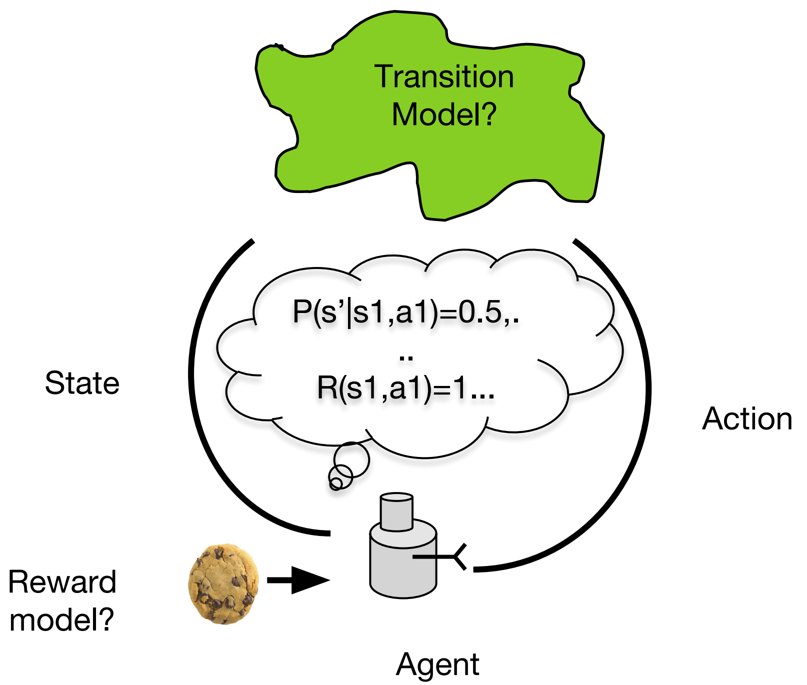
$$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

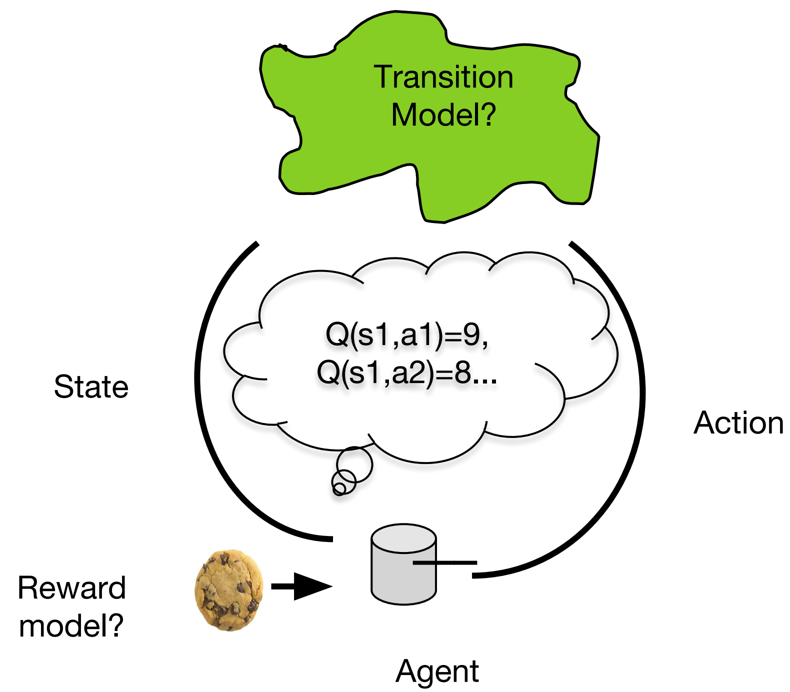
What We’ve Covered So Far

- Markov decision process planning
- Reinforcement learning in finite state and action domains
  - model-based RL
  - model-free RL
  - Passive RL (estimate V while following 1 policy)
  - General/control RL -- policy may change, may want to estimate value of a another policy
- Generalization with model-free techniques
- Exploration

Model-based RL: Agent estimates a reward & dynamics model… and then computes V/Q



Model-free RL: Agent directly estimates Q / V



Model-Based Passive Reinforcement Learning

- Follow policy π
- Estimate MDP model parameters from data
  - If finite set of states and actions: count & average
- Given estimated MDP, compute value of policy π
  - Planning problem! Policy evaluation with a model

- Does this give us dynamics model parameter estimates for all actions?
  - No. But all ones need to estimate the value of the policy.
- How good is the model parameter estimates?
  - Depends on amount of data we have
- What about the resulting policy value estimate?
  - Depends on quality of model parameters

Model-Based Reinforcement Learning

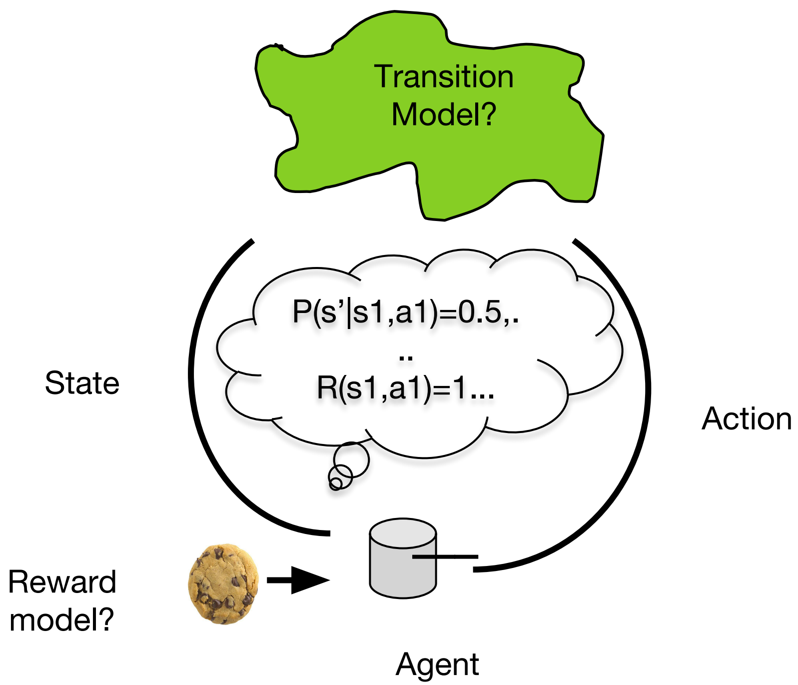
1. Take action a for current state s given policy π
   a. Observe next state and reward
2. Create a MDP given all previous data the data
3. Given that MDP compute policy π using planning
4. Go to step 1

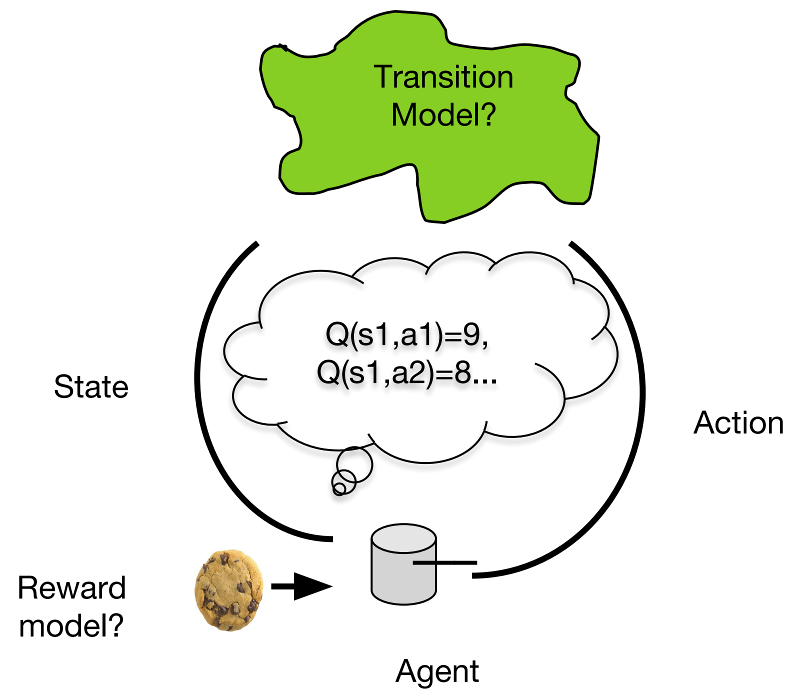
Model-Based Reinforcement Learning* (Will discuss other exploration later)

1. Take action a for current state s given policy π
   a. Observe next state and reward
2. Create a MDP given all previous data the data
3. Given that MDP compute policy π using planning
   - Certainty equivalence:
     - Estimate MLE MDP M1 parameters from data
     - Compute ~Q* for M1 and let π(s) = argmax ~Q*(s,a)
   - E-greedy
     - Estimate MLE MDP M1 parameters from data
     - Compute ~Q* for M1 and π(s) = argmax ~Q*(s,a) w prob (1-e), else select action at random
4. Go to step 1

Model-based RL: Agent estimates a reward & dynamics model… and then computes V/Q

Model-free RL: Agent directly estimates Q / V





Model-free Passive RL

- Directly estimate Q or V of a policy π as act using π
- The Q function for a particular policy is the **expected discounted sum of future rewards** obtained by following policy starting with (s,a)
- For Markov decision processes,

$$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

Model-free Passive RL

- Directly estimate Q or V of a policy from data
- The Q function for a particular policy is the **expected discounted sum of future rewards** obtained by following policy starting with (s,a)
- For Markov decision processes,

$$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

- Consider episodic domains
  - Act in world for H steps, then reset back to state sampled from starting distribution
- MC: directly average episodic rewards
- TD/Q-learning: use a “target” to bootstrap

MC vs Dynamic Programming vs TD

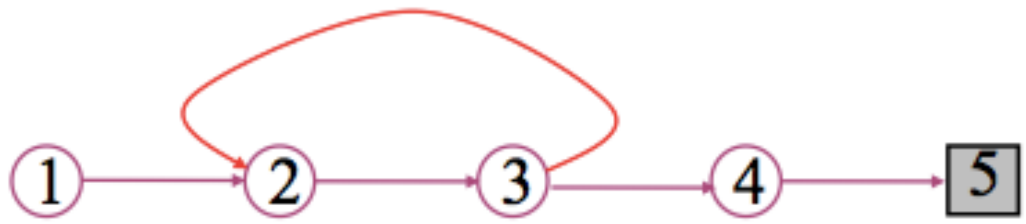
- Updating a state/action value (V or Q)

$$V^\pi(s) = r(s,\pi(s)) + \gamma \sum_{s' \in S} p(s'|s,\pi(s))V^\pi(s')$$

- Bootstrapping: update involves an estimate of V
  - MC does not bootstrap
  - Dynamic programming (value iteration) bootstraps
  - TD bootstraps
- Sampling: update samples an expectation
  - MC samples
  - Dynamic programming does not sample
  - TD samples
  - (e.g. samples s' instead of taking expectation directly)

Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi(s)$ from episodes of experience under policy π
- Idea: Average returns observed after visits to s:



- Every-Visit MC: average returns for every time s is visited in an episode
- First-visit MC: average returns only for first time s is visited in an episode
- Both converge asymptotically

  - Showing this for First-visit is a few lines— see chp 5 in new Sutton & Barto textbook
  - Showing this for Every-Visit MC is more subtle, see Singh and Sutton 1996 Machine Learning paper

First-Visit MC Policy Evaluation

- To evaluate state s
- The first time-step t that state s is visited in an episode,
- Increment counter: $N(s) \leftarrow N(s) + 1$
- Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Gt = total sum of discounted rewards in episode t from state s to end of episode

Incremental Monte Carlo Updates

- Update V(s) incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\left(G_t - V(S_t)\right)$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

MC Estimation of Action Values (Q)

- Monte Carlo (MC) is most useful when a model is not available
  - We want to learn q*(s,a)
- $q_\pi(s,a)$ - average return starting from state s and action a following π

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big].
\end{aligned}
$$

- Converges asymptotically if every state-action pair is visited
- Exploring starts: Every state-action pair has a non-zero probability of being the starting pair

TD Learning

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^\pi(s')$$

- Update $V^\pi(s)$ each time after each transition (s, a, s', r)
- Approximating expectation over next state with samples
- Bootstraps because uses estimate of V instead of real V when creating the target/sampled V

Decrease learning rate over time

$$V_{samp}(s) = r + \gamma V^\pi(s')$$

$$V^\pi(s) = (1 - \alpha) V^\pi(s) + \alpha V_{samp}(s)$$

Model-Free Learning w/Random Actions

- TD learning for policy evaluation:
  - As act in the world go through (s,a,r,s',a',r',…)
  - Update V^{π} estimates at each step
- Over time updates mimic Bellman updates
- Now do for Q values

Q-Learning

- Update Q(s,a) every time experience (s,a,s',r)
  - Create new sample estimate

$$\begin{aligned} Q_{samp}(s,a) &= r + \gamma V(s') \\ &= r + \gamma \max_{a'} Q(s', a') \end{aligned}$$

  - Update estimate of Q(s,a)

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha Q_{samp}(s,a)$$

Convergence with Acting Randomly

- Choose actions randomly
- Do certainty based MDP planning with lookup table representation
  - Compute V and pi will converge to optimal v and policy in limit of infinite data
- Q learning will converge in limit to Q*
- Under reachability assumption
  - Take all actions in all states infinitely often

Q-Learning Properties

- If acting randomly*, Q-learning converges Q*
  - Optimal Q values
  - Finds optimal policy
- Off-policy learning
  - Can act in one way
  - But learning values of another π (the optimal one!)

*Again, under mild reachability assumptions

What We’ve Covered So Far

- Markov decision process planning
- Reinforcement learning in finite state and action domains
- Generalization with model-free techniques
  - linear value/Q function approximation
  - deep reinforcement learning
- Exploration

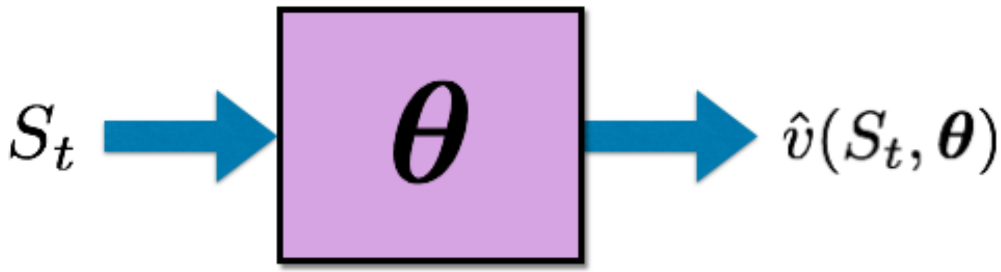
Scaling Up

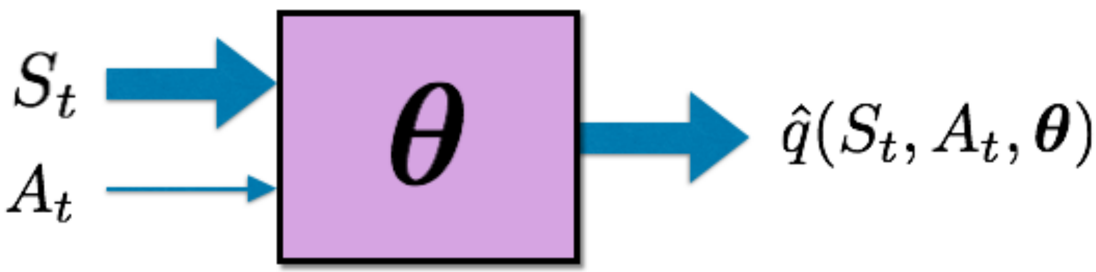
S1	S2	S3	S4	S5	S6	S7
Okay Field Site +1						Fantastic Field Site +10

- Want to be able to tackle problems with enormous or infinite state spaces
- Tabular representation is insufficient

Value Function Approximation (VFA)

- Value function approximation (VFA) replaces the table with a general parameterized form:





Stochastic Gradient Descent

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$:

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

- Gradient descent finds a local minimum:

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Stochastic gradient descent (SGD) samples the gradient:

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

Linear Value Function Approximation (VFA)

- Represent value function by a linear combination of features

$$\hat{v}(S,\mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters w

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2\right]$$

- Update rule is particularly simple

$$\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w}) = \mathbf{x}(S)$$

$$\Delta\mathbf{w} = \alpha(\underline{v_\pi(S)} - \hat{v}(S,\mathbf{w}))\mathbf{x}(S)$$

Don't know true V! How approximate it?

- Update = step-size × prediction error × feature value
- Later, we will look at the neural networks as function approximators.

Monte Carlo with VFA

Gradient Monte Carlo Algorithm for Approximating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^n \to \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta}$ as appropriate (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat forever:
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  For $t = 0, 1, \ldots, T-1$:
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \big[G_t - \hat{v}(S_t, \boldsymbol{\theta}) \big] \nabla \hat{v}(S_t, \boldsymbol{\theta})$

Gt = total sum of discounted rewards in episode t from state s to end of episode

TD Learning with VFA

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^n \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\boldsymbol{\theta}$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{v}(S',\boldsymbol{\theta}) - \hat{v}(S,\boldsymbol{\theta})\big]\nabla\hat{v}(S,\boldsymbol{\theta})$
        $S \leftarrow S'$
    until S' is terminal

Use Vsamp as estimate of “true” value function. Takes immediate reward and value of next s’ under the current value function approximation.

Linear Action-Value Function Approximation

- Represent state and action by a feature vector

$$\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$$

- Represent action-value function by linear combination of features

$$\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A)\mathbf{w}_j$$

- Stochastic gradient descent update

$$\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$$

$$\Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\mathbf{x}(S, A)$$

Incremental Control Algorithms

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \rightarrow \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta} \in \mathbb{R}^n$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        If S' is terminal:
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \boldsymbol{\theta})$ (e.g., ε-greedy)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{q}(S', A', \boldsymbol{\theta}) - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
        $S \leftarrow S'$
        $A \leftarrow A'$

SGD with Experience Replay

- Given experience consisting of ⟨state, value⟩ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Repeat
  - Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

  - Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha (v^\pi - \hat{v}(s, \mathbf{w})) \nabla_\mathbf{w} \hat{v}(s, \mathbf{w})$$

- Converges to least squares solution

Monte Carlo vs TD Learning: Convergence in On Policy Case

- Evaluating value of a single policy

$$MSVE(w) = \sum_{s \in S} d(s) \left(V^{\pi}(s) - \tilde{V}^{\pi}(s, w) \right)^2$$

- where
  - d(s) is generally the on-policy π stationary distrib
  - ~V(s,w) is the value function approximation

Convergence given infinite amount of data?

Tsitsiklis and Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. 1997

Monte Carlo Convergence: Linear VFA

- Evaluating value of a single policy

$$MSVE(w) = \sum_{s \in S} d(s) \left(V^{\pi}(s) - \tilde{V}^{\pi}(s,w) \right)^2$$

- where
  - d(s) is generally the on-policy π stationary distrib
  - ~V(s,w) is the value function approximation
- Linear VFA: $V(s) = \sum_i w_i f_i(s)$
- **Monte Carlo converges to min MSE possible!**

$$MSVE(w_{MC}) = \min_w \sum_{s \in S} d(s) \left(V^{\pi}(s) - \tilde{V}^{\pi}(s,w) \right)^2$$

Tsitsiklis and Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. 1997

TD Learning Convergence: Linear VFA

- Evaluating value of a single policy

$$MSVE(w) = \sum_{s \in S} d(s) \left(V^{\pi}(s) - \tilde{V}^{\pi}(s,w) \right)^2$$

- where
  - d(s) is generally the on-policy π stationary distrib
  - ~V(s,w) is the value function approximation
- Linear VFA: $V(s) = \sum_i w_i f_i(s)$
- **TD converges to constant factor of best MSE**

$$\begin{aligned} MSVE(w_{TD}) &= \frac{1}{1-\gamma} \min_w \sum_{s \in S} d(s) \left(V^{\pi}(s) - \tilde{V}^{\pi}(s,w) \right)^2 \\ &= \frac{1}{1-\gamma} MSVE(w_{MC}) \end{aligned}$$

Tsitsiklis and Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. 1997

TD Learning vs Monte Carlo:

Finite Data, Lookup Table, Which is Better?

- MC converges to minimum MSE
- TD w/infinite experience replay converges to estimate as if computed MLE MDP model and then computed value of policy
  - This is often better if the domain is a MDP!
  - Leverages Markov structure

Off Policy Learning

Can use importance sampling with MC to estimate the value of a policy didn't try (but that has support in behavior policy)

Q-learning with function approximation can diverge (not converge even given infinite data)

- See examples in Chapter 11 (Sutton and Barto)

Deep Learning & Model Free Q learning

$$\mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}}\right)^2\right]$$

- Running stochastic gradient descent
- Now use a deep network to approximate Q

Challenges

- Challenge of using function approximation
  - Local updates (s,a,r,s’) highly correlated
  - “Target” (approximation to true value of s’) can change quickly and lead to instabilities

DQN: Q-learning with DL

- Experience replay of mix of prior (s_i,a_i,r_i,s_{i+1}) tuples to update Q(w)
- Fix target Q (w–) for number of steps, then update
- Optimize MSE between current Q and Q target

$$\mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(\underbrace{r+\gamma\max_{a'}Q(s',a';w_i^-)}_{\text{Q-learning target}}-\underbrace{Q(s,a;w_i)}_{\text{Q-network}}\right)^2\right]$$

- Use stochastic gradient descent

Deep RL

- Experience replay is hugely helpful
- Target stablization is also helpful
- No guarantees on convergence (yet)
- Some other influential ideas
  - Prioritize experience replay
  - Double Q (two separate networks, each act as a "target" for each other)
  - Dueling: separate value and advantage
  - Many of these ideas build on prior ideas for RL with look up table representations

What We've Covered So Far

- Markov decision process planning
- Reinforcement learning in finite state and action domains
- Generalization with model-free techniques
- Exploration
  - e-greedy
  - PAC: Rmax
  - regret: UCRL
  - know what different things guarantee, algorithms that achieve, when might want one criteria or another. Do not have to be able to derive full proofs

Performance of RL Algorithms

- Convergence
- Asymptotically optimal
  - In limit of infinite data, will converge to optimal π
  - E.g. Q-learning with e-greedy action selection
  - Says nothing about finite-data performance
- Probably approximately correct
- Minimize / sublinear regret

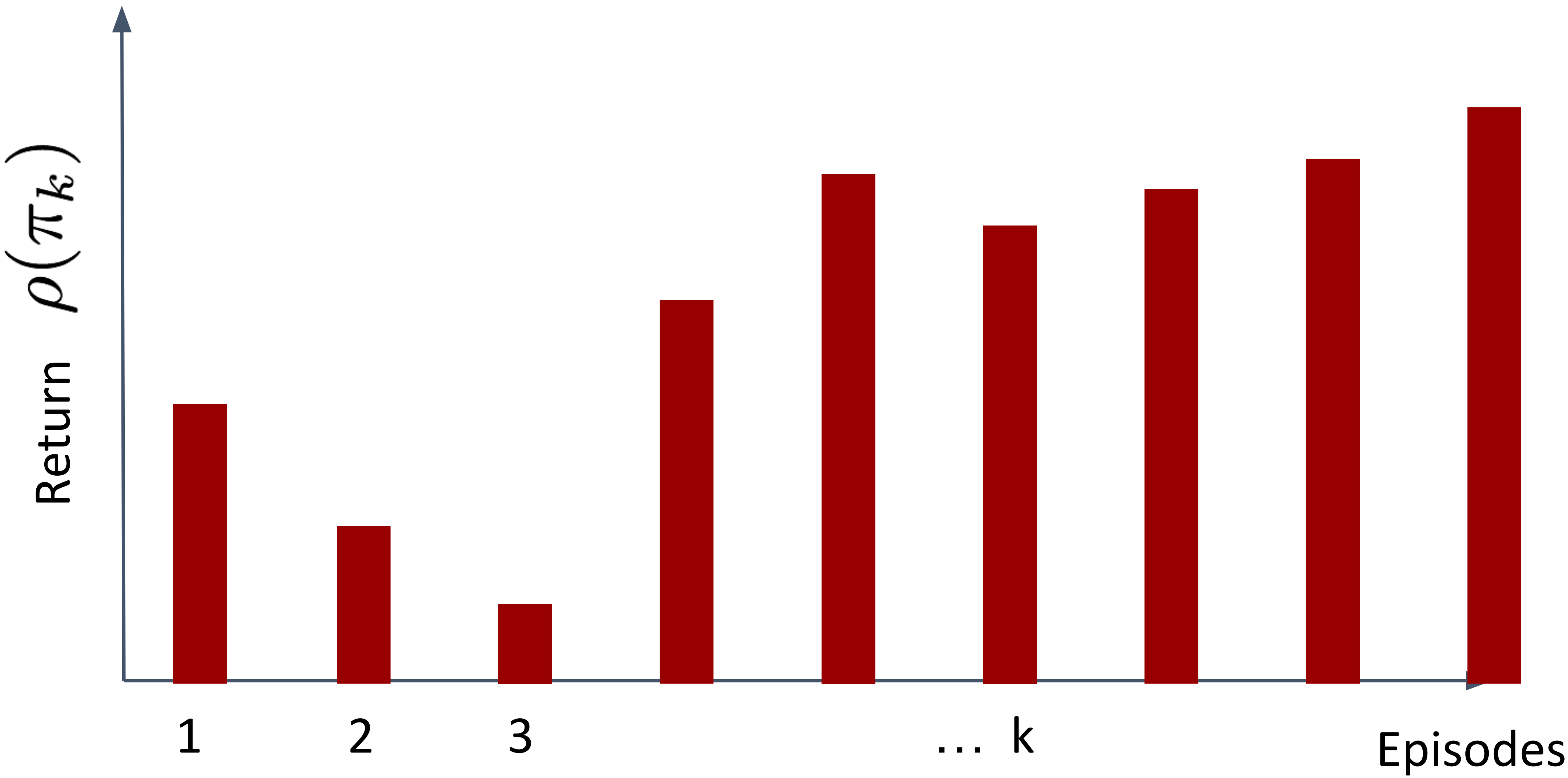
Optimism Under Uncertainty

R-max (Brafman and Tennenholtz).
Slight modification of R-max (Algorithm 1) pseudo code in [Reinforcement Learning in Finite MDPs: PAC Analysis]
(Strehl, Li, LIttman 2009)

Inputs: $S, A, \gamma, m, \varepsilon_1$, and $U(\cdot,\cdot)$
for all (s,a) **do**
    $Q(s,a) \leftarrow \boxed{R_{max} / (1-\gamma)}$
    $r(s,a) \leftarrow 0$
    $n(s,a) \leftarrow 0$
    for all $s' \in S$ **do**
        $n(s,a,s') \leftarrow 0$
    end for
end for
for $t = 1,2,3,\cdots$ **do**
    Let s denote the state at time t.
    Choose action $a := \text{argmax}_{a' \in A} Q(s,a')$.
    Let r be the immediate reward and s' the next state after executing action a from state s.
    if $n(s,a) < m$ **then**
        $n(s,a) \leftarrow n(s,a) + 1$
        $r(s,a) \leftarrow r(s,a) + r$ // *Record immediate reward*
        $n(s,a,s') \leftarrow n(s,a,s') + 1$ // *Record immediate next-state*
        if $n(s,a) = m$ **then**
            for $i = 1,2,3,\cdots,\left\lceil \frac{\ln(1/(\varepsilon_1(1-\gamma)))}{1-\gamma} \right\rceil$ **do**
                for all $(\bar{s},\bar{a})$ **do**
                    if $n(\bar{s},\bar{a}) \geq m$ **then**
                        $Q(\bar{s},\bar{a}) \leftarrow \hat{R}(\bar{s},\bar{a}) + \gamma \sum_{s'} \hat{T}(s'|\bar{s},\bar{a}) \max_{a'} Q(s',a')$.
                    end if
                end for
            end for
        end if
    end if
end for

Regret vs PAC vs ...?

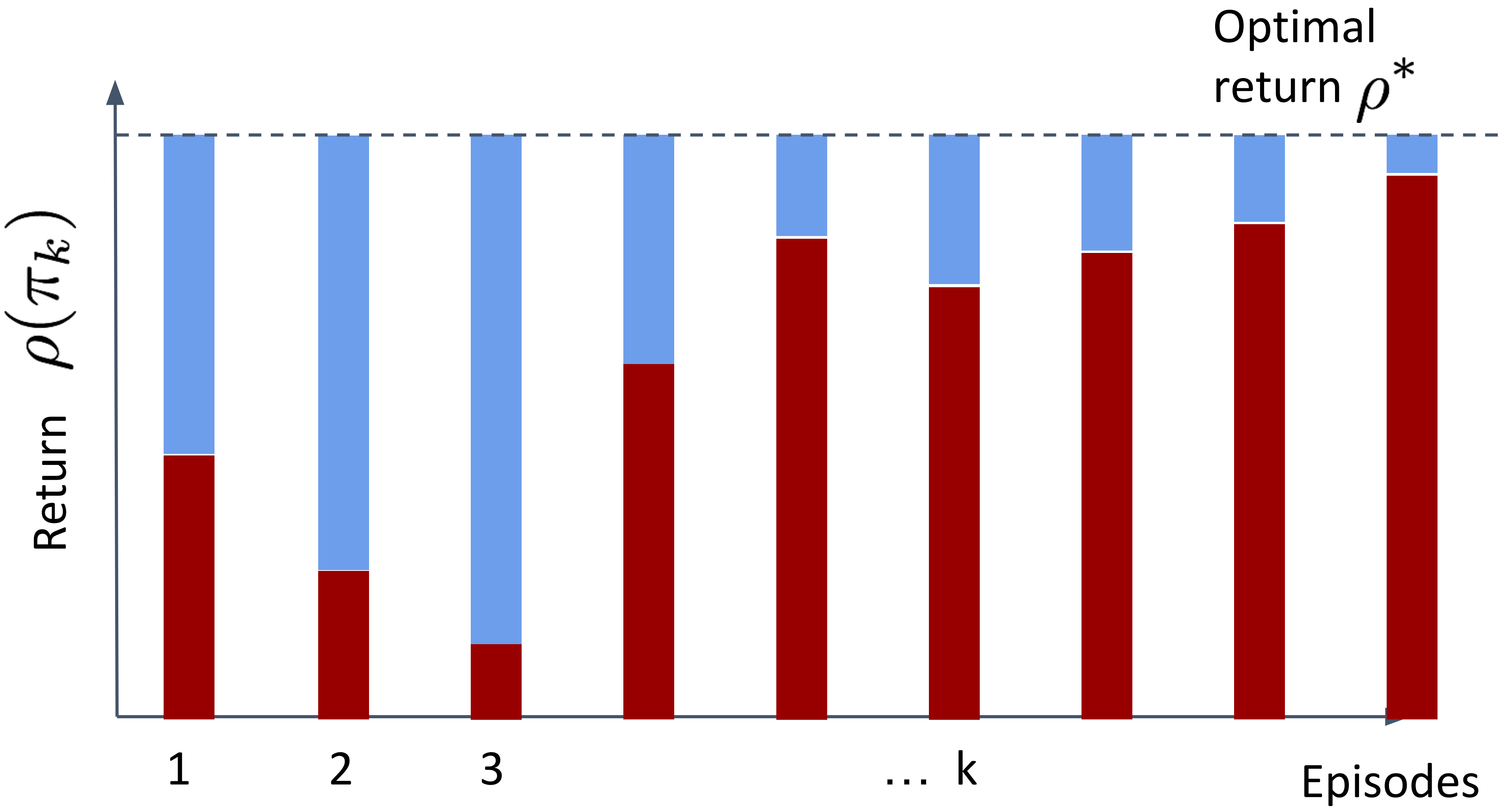
- What choice of performance should we care about?
- For simplicity, consider **episodic** setting
- Return is the sum of rewards in an episode



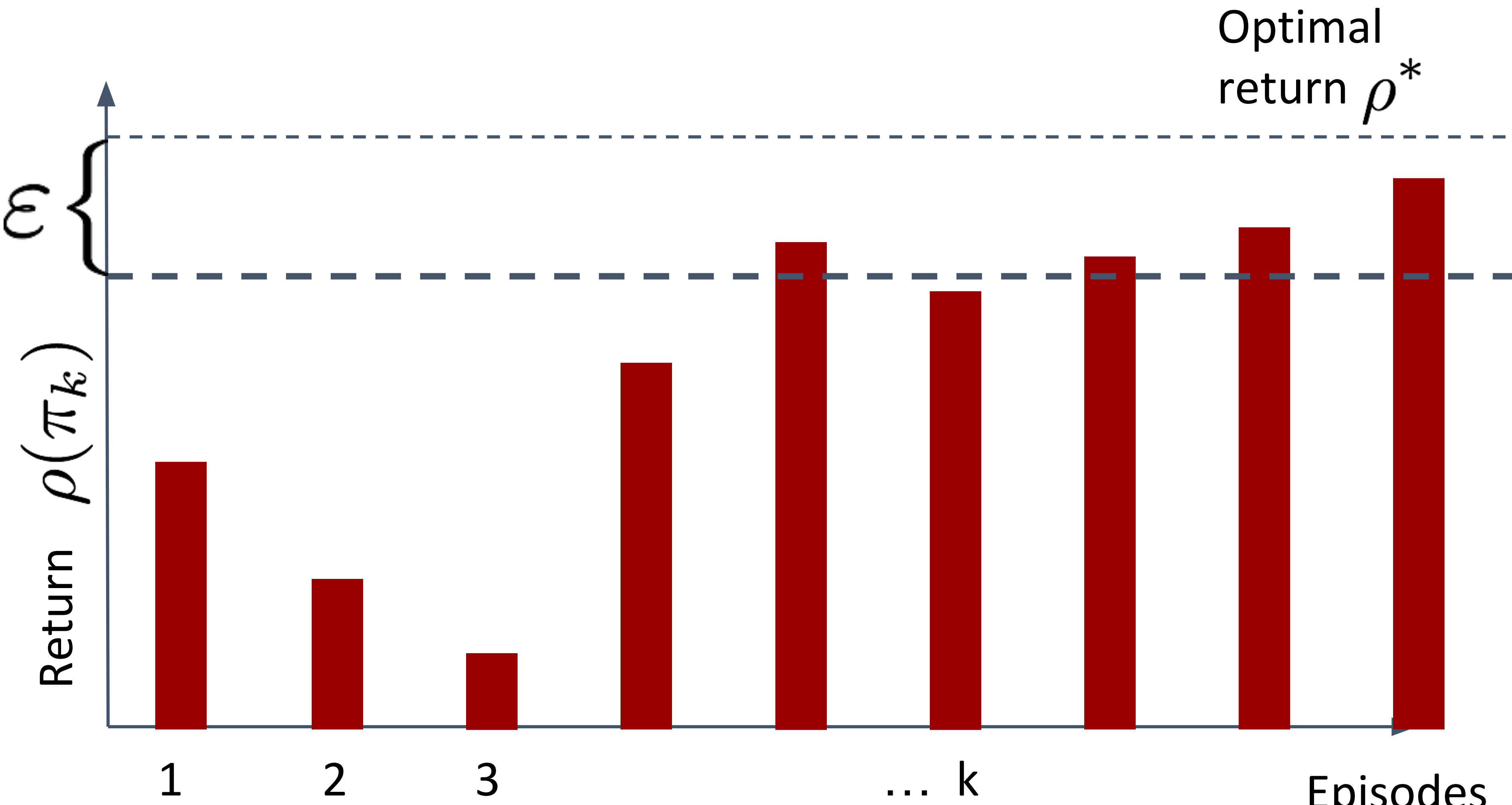
Regret Bounds

$$R(T) = T\rho^* - \sum_{k=1}^{T} \rho(\pi_k)$$

Guarantees bound expected regret $\mathbb{E}[R(T)]$



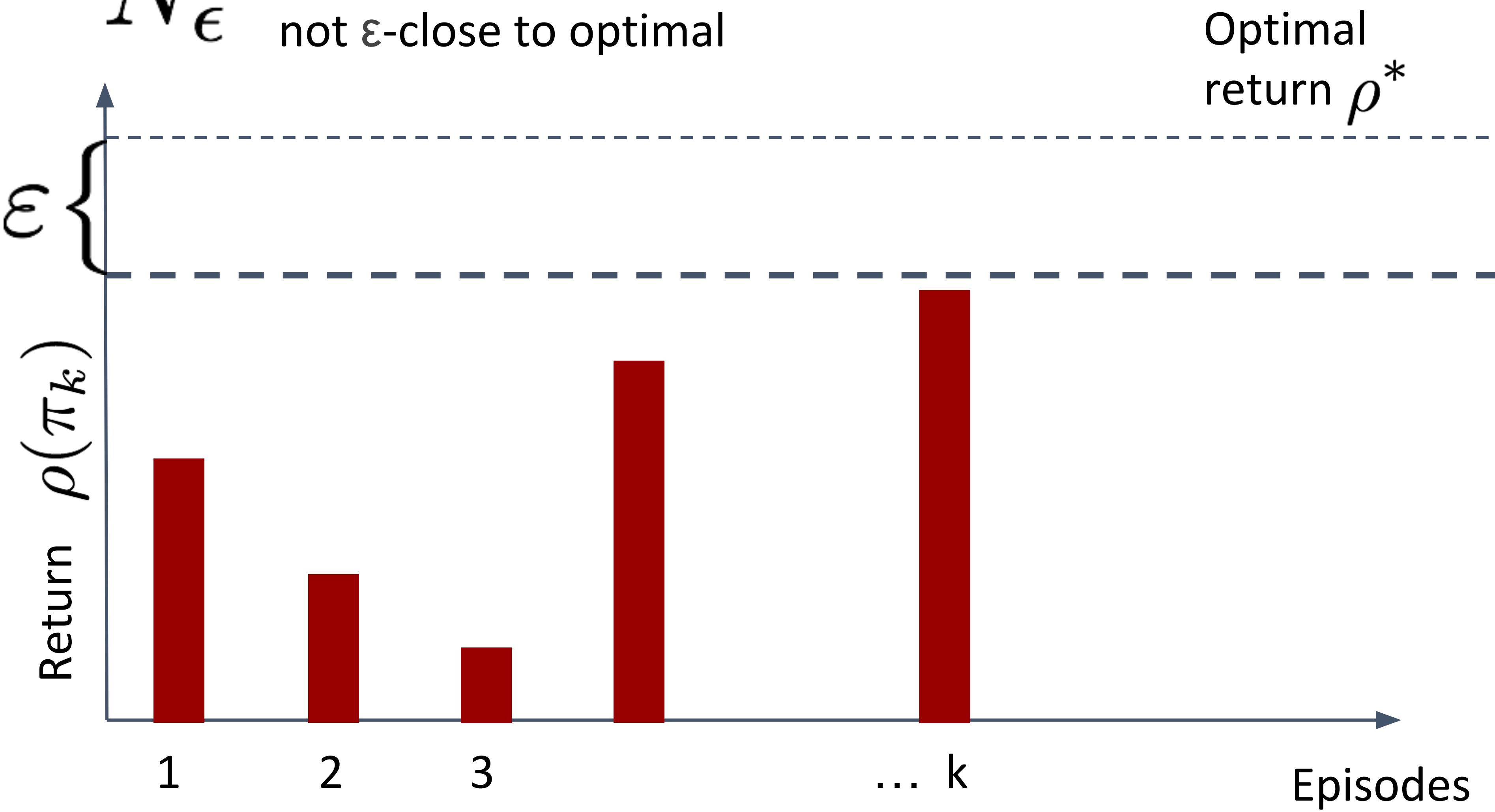
(ε,δ) - Probably Approximately Correct



(ε,δ) - Probably Approximately Correct

$$\mathbb{P}(N_\epsilon \leq F(S, A, H, \epsilon, \delta)) \geq 1 - \delta$$

N_ϵ Number of episodes with policies not ε-close to optimal



Uniform-PAC

(Dann, Lattimore, Brunskill, arxiv, 2017)

$$\mathbb{P}(\forall \varepsilon : N_\varepsilon \leq F(S, A, H, \varepsilon, \delta)) \geq 1 - \delta$$

A δ-Uniform PAC-bound implies with prob. > 1 - δ :

- Convergence to π^*
- (ε,δ) - PAC $\forall\, \varepsilon$
- Regret bound $R(T)$