Reinforcement Learning for Ecosystem Management

Tom Dietterich
Majid Alkaee Taleghan
Sean McGregor

Outline

- Ecosystem Management
  - Example: Invasive Species
- Markov Decision Problems
- Solution Scenarios
  - Known MDP
  - Simulator
  - Real World
- Learning Algorithms

Ecosystem Management

- Human activity is affecting ecosystems worldwide
- Result: Ecosystems require active management to be healthy
  - Endangered species management
  - Invasive species management
  - Habitat preservation and reserve design
  - Wildfire management
- Agriculture is also a form of managed ecosystem
  - Disease management
  - Soils management
  - Cropping management

Example: Tamarisk Invasions in US

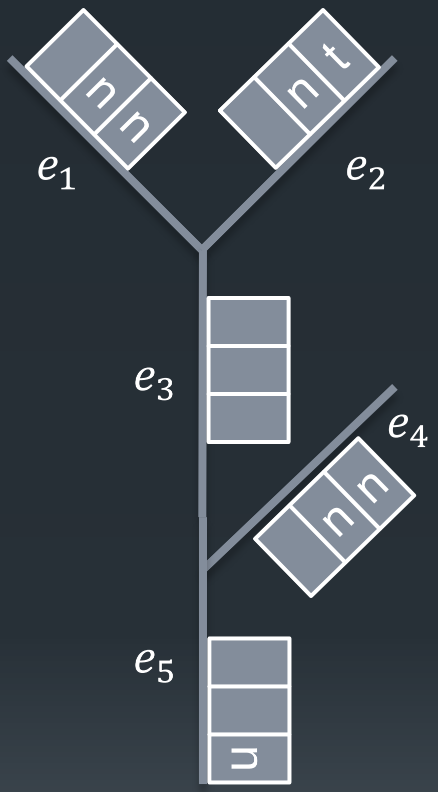
- Tamarisk: invasive tree from the Middle East
  - Has invaded over 3 million acres of land in the Western United States
  - Out-competes native vegetation for water
  - Reduces biodiversity and causes species extinction
  - Economically costly

C.C. Shock, Oregon State University

- What is the best way to manage a spatially-spreading organism?

Tamarisk Mathematical Model

- Tree-structured river network
  - Each edge $e \in E$ has H “sites” where a tree can grow.
  - Each site can be
    - {empty, occupied by native, occupied by invasive}

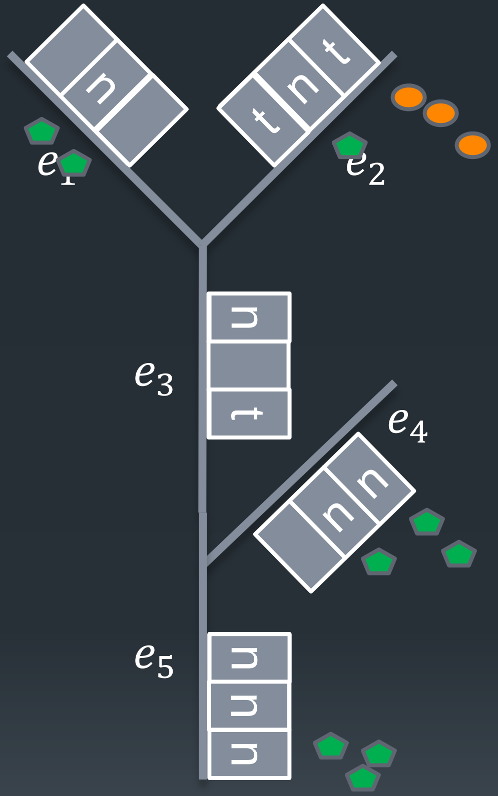


- Management actions
  - Each edge: {do nothing, eradicate, plant, restore (=eradicate + plant)}

Muneepeerakul, et al., 2007 J. Theoretical Biology

Dynamics

- Discrete time transition model
- In each time period
  - Natural death
  - Seed production
  - Seed dispersal (preferentially downstream)
  - Seed competition to become established



Optimization Goal

- Each action in each edge has a cost $C(a_e, e)$
- Budget constraint: $\sum_e C(a_e, e) \leq B$; $B = 100$

Action	Cost
Do Nothing	0
Eradicate (per slot)	0.5
Plant Native (per slot)	0.9
Both (per slot)	1.4

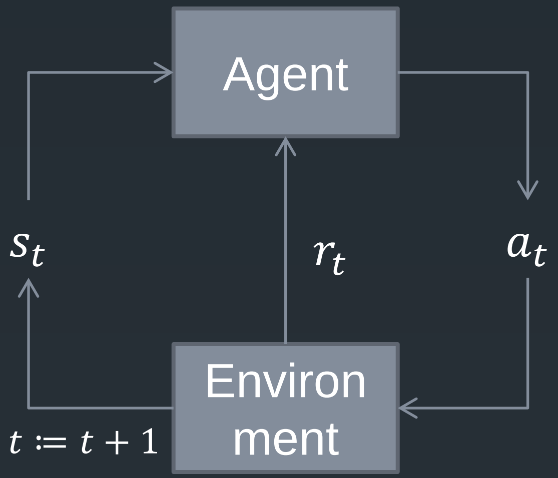
- Create a “virtual cost” for the invasion
  - At each time step, charge a cost for
    - each invaded edge: 10
    - for each Tamarisk tree in each invaded edge: 0.1
- Minimize the cumulative infinite-horizon discounted cost of management
  - $\sum_{t=0}^{\infty} \gamma^t c_t$ where $\gamma = 0.9$ is the discount rate and c_t is the cost at time t

Formulation as a Markov Decision Process

MDP $\langle S, A, R, T, \gamma, P_0 \rangle$

- S: Set of states (discrete or continuous).
- A: Set of actions (discrete or continuous). We will only consider MDPs with a smallish number of discrete actions
- R: Reward function. $r_t = R(s_t, a_t)$
- T: Probability transition function ("dynamics"): $T(s, a, s') = P(s'|s, a)$
- γ: Discount factor $\gamma \in (0,1)$
- P_0: Distribution of starting states

- We often assume $r_t \in [0, R_{max}]$



Solution to an MDP

- The solution to an MDP is called a Policy: $\pi: S \mapsto A$

- The optimal policy maximizes the expected cumulative discounted sum of rewards:

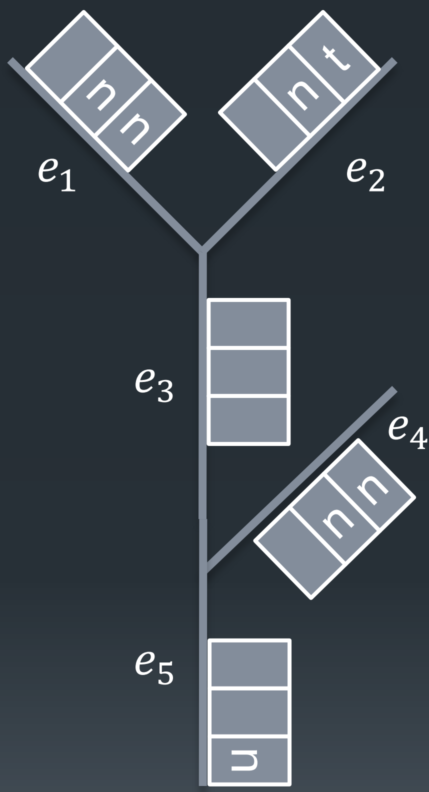
$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| a_t = \pi(s_t)\right]$$

$$\pi^* = \arg\max_{\pi} J(\pi)$$

- where the expectation is taken with respect to $P_0(s_0)$ and $P(s_{t+1}|s_t, a_t)$

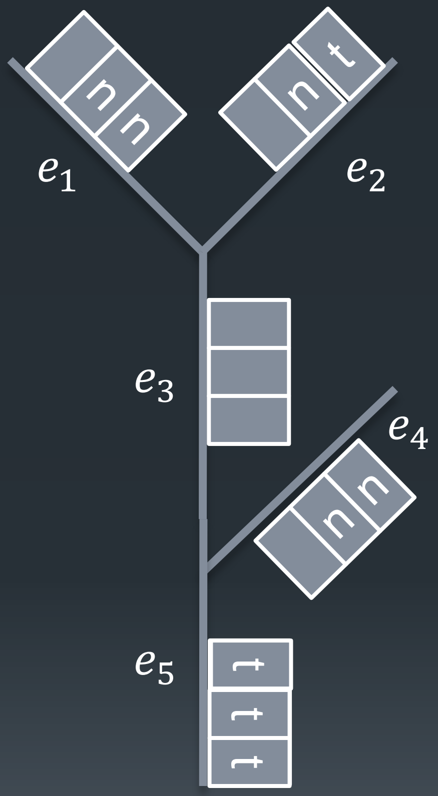
What is the Best Action?

- Try to kill the tamarisk in e_2?
- Plant native trees in e_3?

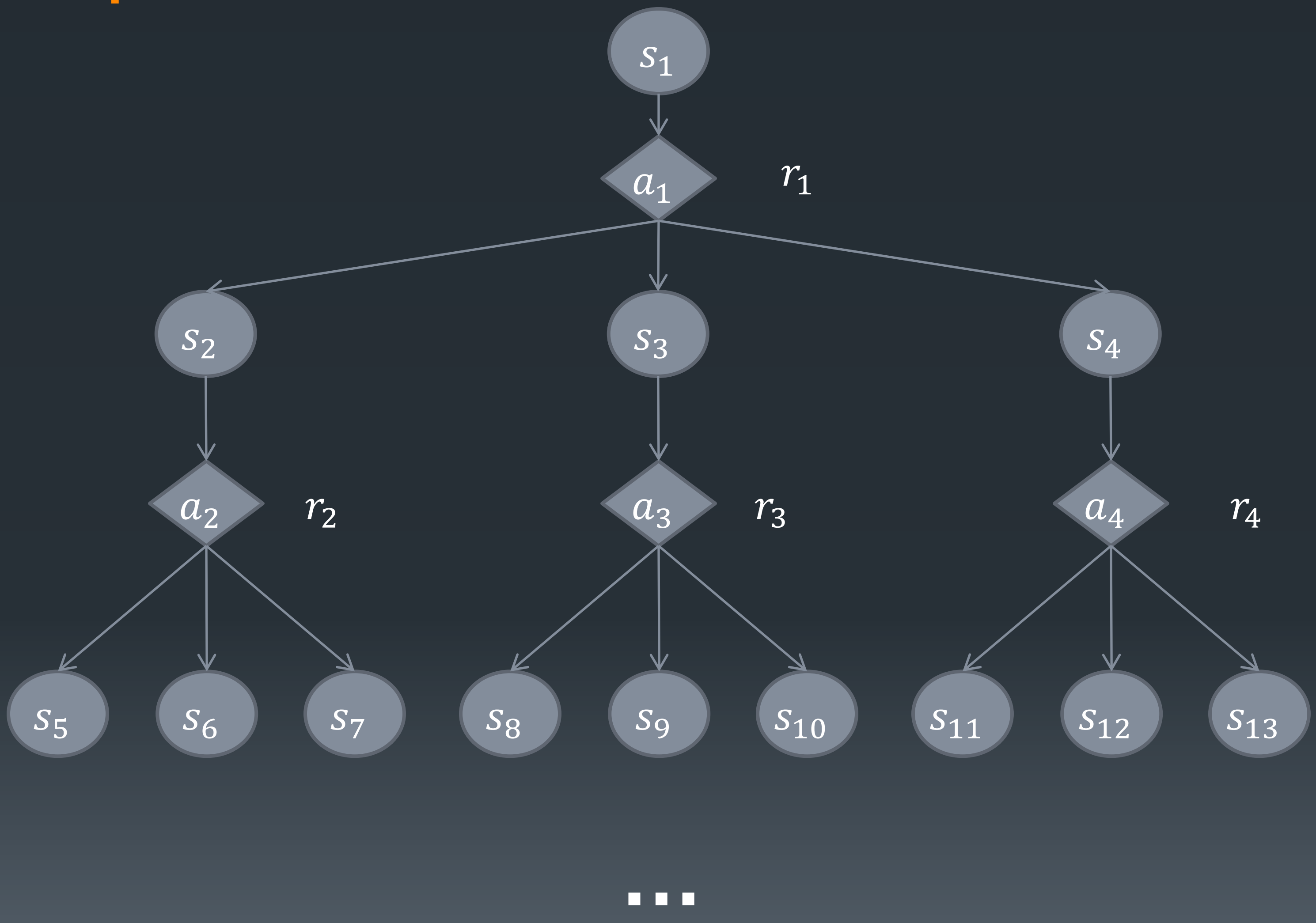


What is the Best Action?

- Try to kill the tamarisk in e_2?
- Try to kill the tamarisk in e_5?



The manager faces a branching state space



Reinforcement Learning

- Explore the state space to find the rewards
- Figure out what actions to take so that we reach those rewards

Why is it "Markov"?

- The transition dynamics only depend on the current state of the system
  - $P(s_{t+1}|s_t, a_t)$
- The reward function only depends on the current state and action

- Under these conditions, it can be proved that the optimal policy only depends on the current state

Variations and Distinctions

- MDP Variations:
  - Policies can be stochastic: $a_t \sim P(a_t|s_t)$
  - Rewards can be stochastic: $R(s_t, a_t) = P(r_t|s_t, a_t)$
  - Rewards can depend on the result state: $R(s_t, a_t, s_{t+1})$

- Contextual Multi-Armed Bandits
  - Actions do not change state, only produce rewards

Variations and Distinctions (2)

- Partially-Observable MDPs (POMDPs)
  - Agent does not observe the full state, but instead has noisy observations $P(o_t|s_s)$
  - Actions serve two purposes: (a) to gain information about the state of the system and (b) to achieve rewards in the system.

- Stochastic Games
  - Two or more agents interacting with each other
  - In Markov Games, the full state is visible to both players

Three Scenarios

- Known MDP
  - The transition function $T(s, a, s')$ is available in a form that makes it easy to evaluate $T(s, a, s')$ given (s, a, s').
    - Transition matrix for each action: $T_a(s, s')$
    - Bayesian network for which inference is tractable
- Simulator MDP
  - The transition function is only available as a simulator. Given (s, a) we can draw a sample $s' \sim P(s'|s, a) = T(s, a, s')$
  - Strong simulator: Can sample from any (s, a)
  - Reset simulator: Can reset to $s_0 \sim P_0(s_0)$ otherwise sample along a trajectory
- Real World
  - We can only execute actions in the real system. Like a simulator, each action gives us a sample $s' \sim P(s'|s, a) = T(s, a, s')$

Three Scenarios (2)

- Known MDP
  - Methods from Operations Research can be applied provided that we have memory of size $|S| \times |A|$

- Simulator MDP
- Real World
  - RL was developed for these two cases
  - RL typically requires many many (s, a, r, s') interactions. This is why the simulator case is the most common. But there are some applications (e.g., in network management) where millions of examples can be acquired quickly
  - An important area today is “Sim-to-Real” transfer. RL is applied first on a simulator, and then a small amount of additional training is done in the real world to adapt the learned policy.

Two Tasks: Evaluation & Optimization

- Policy Evaluation
  - Given a fixed policy π compute $J(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \, | a_t = \pi(s_t)]$

- Policy Optimization
  - Find the policy π^* that maximizes $J(\pi)$.

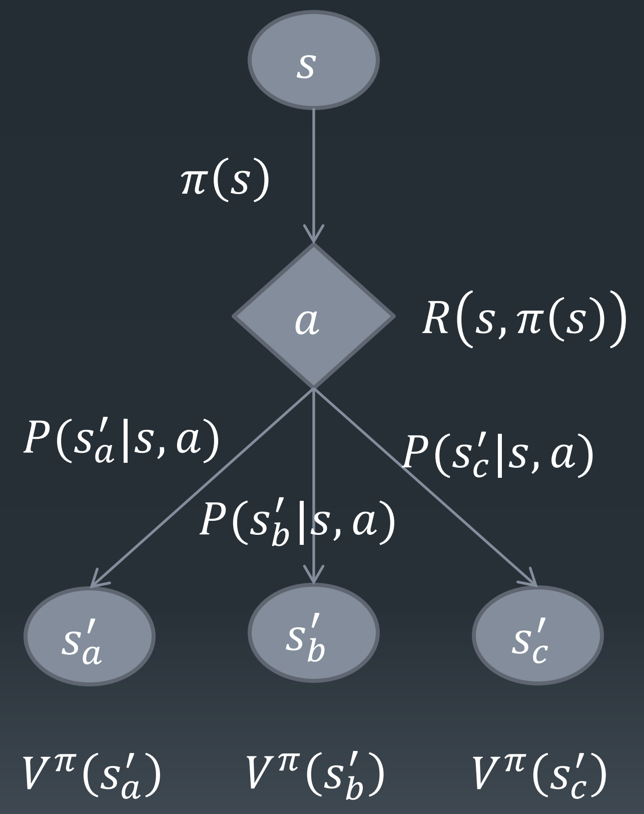
- Notation: The Value Function
  - $V^\pi(s) =$ expected cumulative discounted reward for executing policy π starting in state s.
  - $V^*(s) =$ expected cumulative discounted reward for executing the optimal policy π^* starting in state s

Policy Evaluation

- Known MDP Case
  - Let $V^{\pi}(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \,| a_t = \pi(s_t), s]$
  - Cumulative reward executing policy π starting in state s
- Bellman Equation:

$$V^{\pi}(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s'} P(s'|s,\pi(s)) V^{\pi}(s')$$

- Note: This is a system of linear equations



Policy Evaluation (2)

Known MDP Case continued

- We can compute V^{π} via the following algorithm
- For iteration = 1,…
  - For state $s \in S$ do
    - $V^{\pi}(s) \coloneqq R\big(s, \pi(s)\big) + \gamma \sum_{s'} P(s'|s, \pi(s))V^{\pi}(s')$

- This will converge to a fixed point
- This is a form of dynamic programming
- It is called Value Iteration
- We can visit the states in any order as long as we visit all states infinitely many times

Policy Evaluation (3)

Simulator Case for a specific state s

- Monte Carlo Estimate
- For $i = 1, \ldots, N$:
  - Reset the simulator to state s
  - Sample a trajectory τ_i of length H. Let $\left(r_{i,1}, \ldots, r_{i,H}\right)$ be the rewards obtained
  - $R_i \coloneqq \sum_t \gamma^{t-1} r_{i,t}$ be the observed cumulative discounted reward
- $\hat{V}^\pi(s) = \frac{1}{N} \sum_i R_i$
- This will be biased low by no more than $\gamma^H \frac{R_{max}}{1-\gamma}$ because we are truncating the infinite horizon return at horizon H, $\frac{R_{max}}{1-\gamma}$ is the cumulative discounted reward if we get reward R_{max} in every state forever.

Policy Evaluation (4)

Simulator Case

- Initialize $V^\pi(s) = R\big(s, \pi(s)\big)$ for all s
- For $t = 1, \ldots$
  - $a = \pi(s)$
  - $s' \sim P(s'|s, a)$
  - $V^\pi(s) \coloneqq (1 - \alpha_t)V^\pi(s) + \alpha_t[r_t + \gamma V^\pi(s')]$
  - $s \coloneqq s'$

- This is known as stochastic approximation. It will converge to the correct value function provided

$$\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$$

and the policy π visits every reachable state infinitely often

Stochastic Approximation

- Suppose X is a random variable with distribution $P(X)$
- I can estimate the expected value of X by taking a large sample N and computing
- $\mathbb{E}[X] = \frac{1}{N}\sum_t X_t$ where $X_t \sim P(X)$

- We can write this as the iterative algorithm
- $mean \coloneqq 0$
- For $t = 1, \ldots, N$
  - $mean \coloneqq (1 - \alpha_t)mean + \alpha_t X_t$
- If we set $\alpha_t = \frac{1}{t}$ then we get
  - $mean_1 = X_1$
  - $mean_2 = \frac{1}{2}X_1 + \frac{1}{2}X_2$
  - $mean_3 = \frac{2}{3}\left(\frac{X_1+X_2}{2}\right) + \frac{1}{3}X_3 = \frac{X_1+X_2+X_3}{3}$
  - and so on
- You can verify that $\sum_t \frac{1}{t} = \infty$ but $\sum_t \frac{1}{t^2} = \frac{\pi^2}{6} < \infty$
- [Robbins & Monro, 1951]

Policy Optimization

Known MDP Case

- Bellman Optimality Equation. The optimal policy satisfies

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

- We can apply this as an assignment statement
  - For iteration = 1,…,
    - For state $s \in S$ $V^*(s) := \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$
  - This will eventually converge to the optimal value function
  - This is also a form of Dynamic Programming

- We can recover the optimal policy by computing the action that satisfies the Bellman optimality equation:

$$\pi^*(s) \coloneqq \arg\max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

Policy Optimization (2)

Simulation Case

- Action-Value Function

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) Q^{\pi}(s', \pi(s'))$$

  - Execute action a and then following π thereafter

- Action-Value version of Bellman Optimality Equation

$$Q^{*}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^{*}(s', a')$$

- The value function can be recovered as

$$V^{*}(s) = \max_{a} Q^{*}(s,a)$$

Policy Optimization (3)

Simulation Case: Q Learning

- Let π^x be an "Exploration Policy"
- Initialize $Q(s,a) \coloneqq 0$ for all states s and actions a
- For $t = 1, \ldots$
  - $a \coloneqq \pi^x(s)$
  - $s' \coloneqq$ sampled from the simulator according to $s' \sim P(s'|s,a)$
  - $Q(s,a) \coloneqq (1-\alpha_t)Q(s,a) + \alpha_t[R(s,a) + \gamma \max'_a Q(s',a')]$

- Again this is a stochastic approximation version of Value Iteration over the Action Value Function
- π^x must try every action a in every state s infinitely often to guarantee convergence

Generic Exploration Policies

- Epsilon-Greedy
  - With probability $1 - \epsilon$, select $a = \arg\max_{a'} Q(s, a')$. This is the “greedy” action
  - With probability ϵ, select $a \in A$ uniformly at random

Generic Exploration Policies (2)

- Boltzmann Exploration
  - Select a according to the Boltzmann distribution
  - $P(a) = \frac{\exp \frac{Q(s,a)}{\tau}}{\sum_{a'} \exp \frac{Q(s,a')}{\tau}}$
  - Here, τ is the temperature parameter. As $\tau \rightarrow 0$, this approaches the greedy distribution that assigns probability 1 to $\arg\max_{a'} Q(s, a')$
  - This is called the Softmax Distribution
  - Typically τ is started high and gradually decreased toward 0

Reinforcement Learning with Function Approximation

- All of the methods discussed so far assume we can store V (size $|S|$) or Q (size $|S| \times |A|$)
- This is not always feasible

- The research community explored using neural networks (and other function approximators) to represent $Q(s, a) = Q(s, a; \theta)$, where θ is the set of parameters of a neural network

- This often fails badly

Policy Search Methods

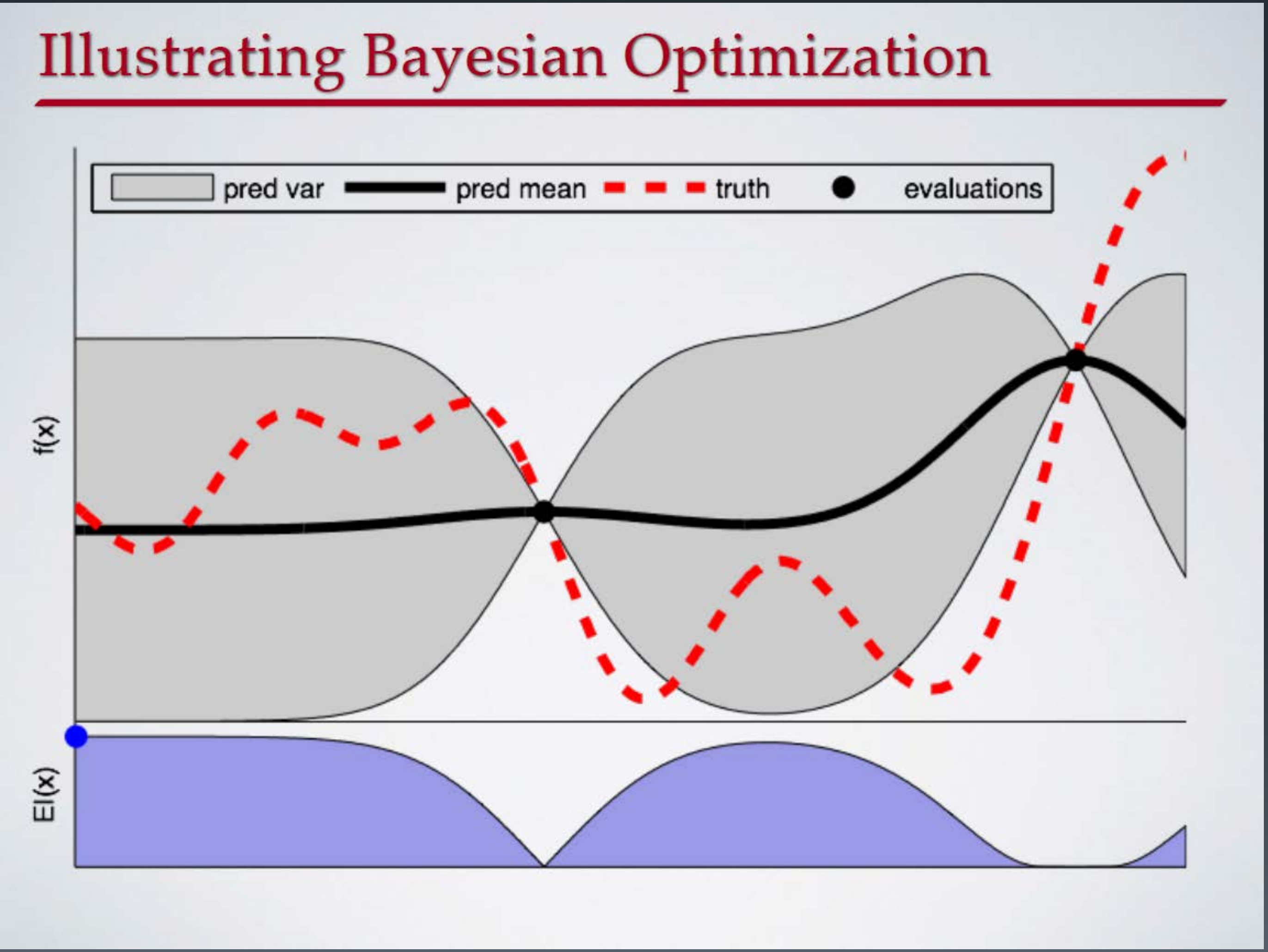
- Let $\pi(s;\theta)$ be a parameterized class of policies

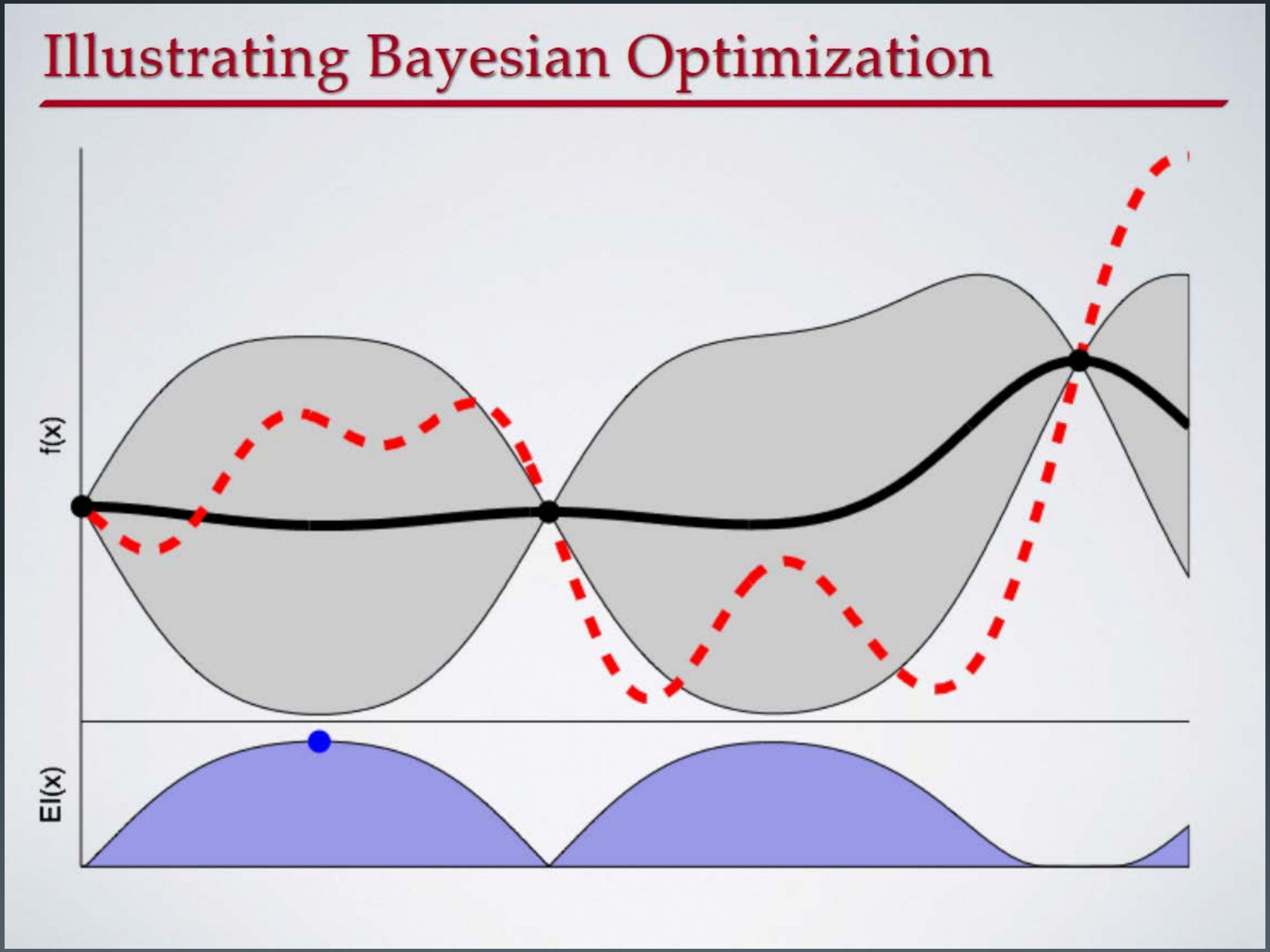
- Goal: Find θ to maximize
$J(\theta) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \,|a_t = \pi(s_t;\theta)]$

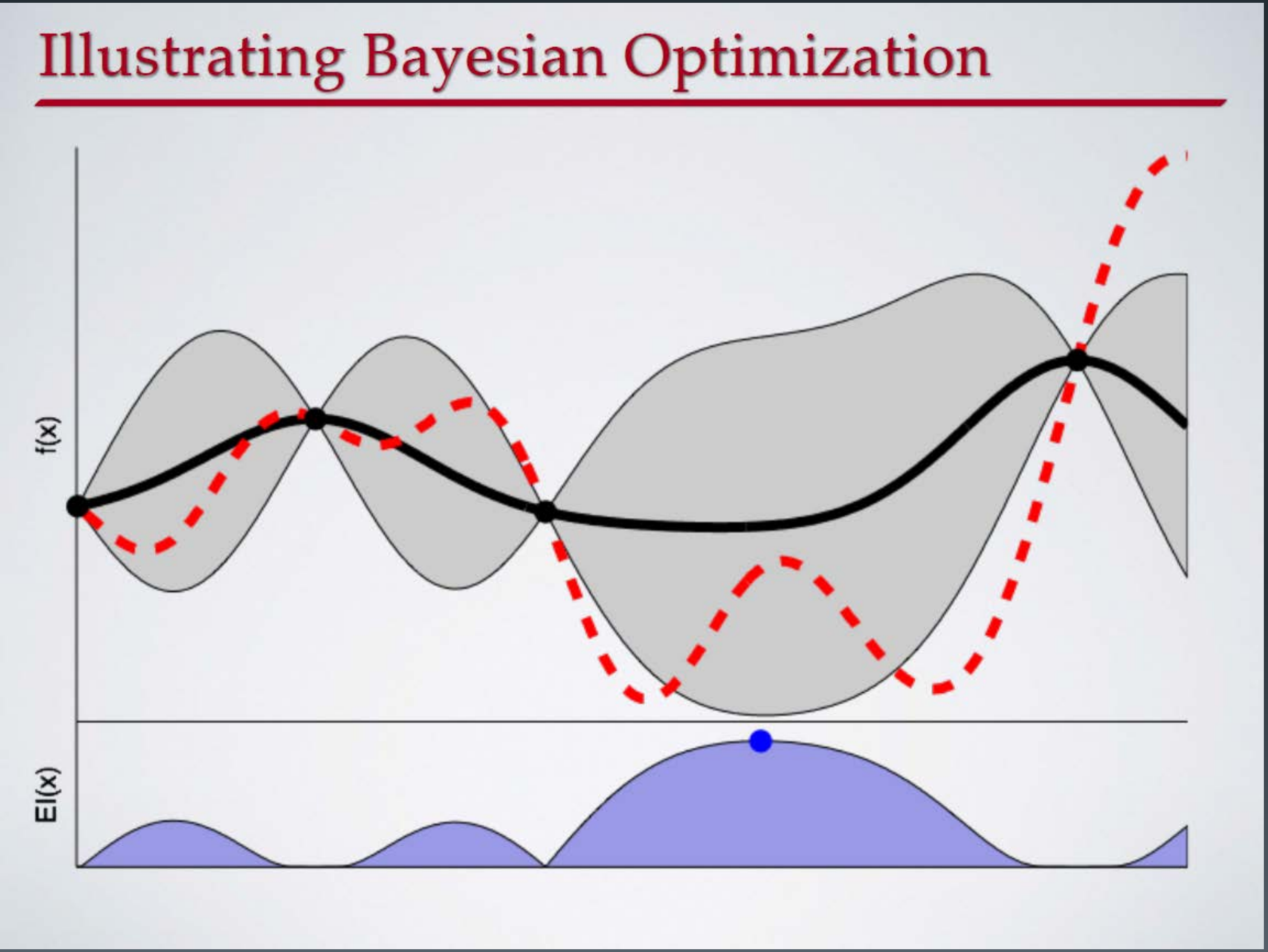
Sequential Model-Based Optimization (aka “Bayesian Optimization”)

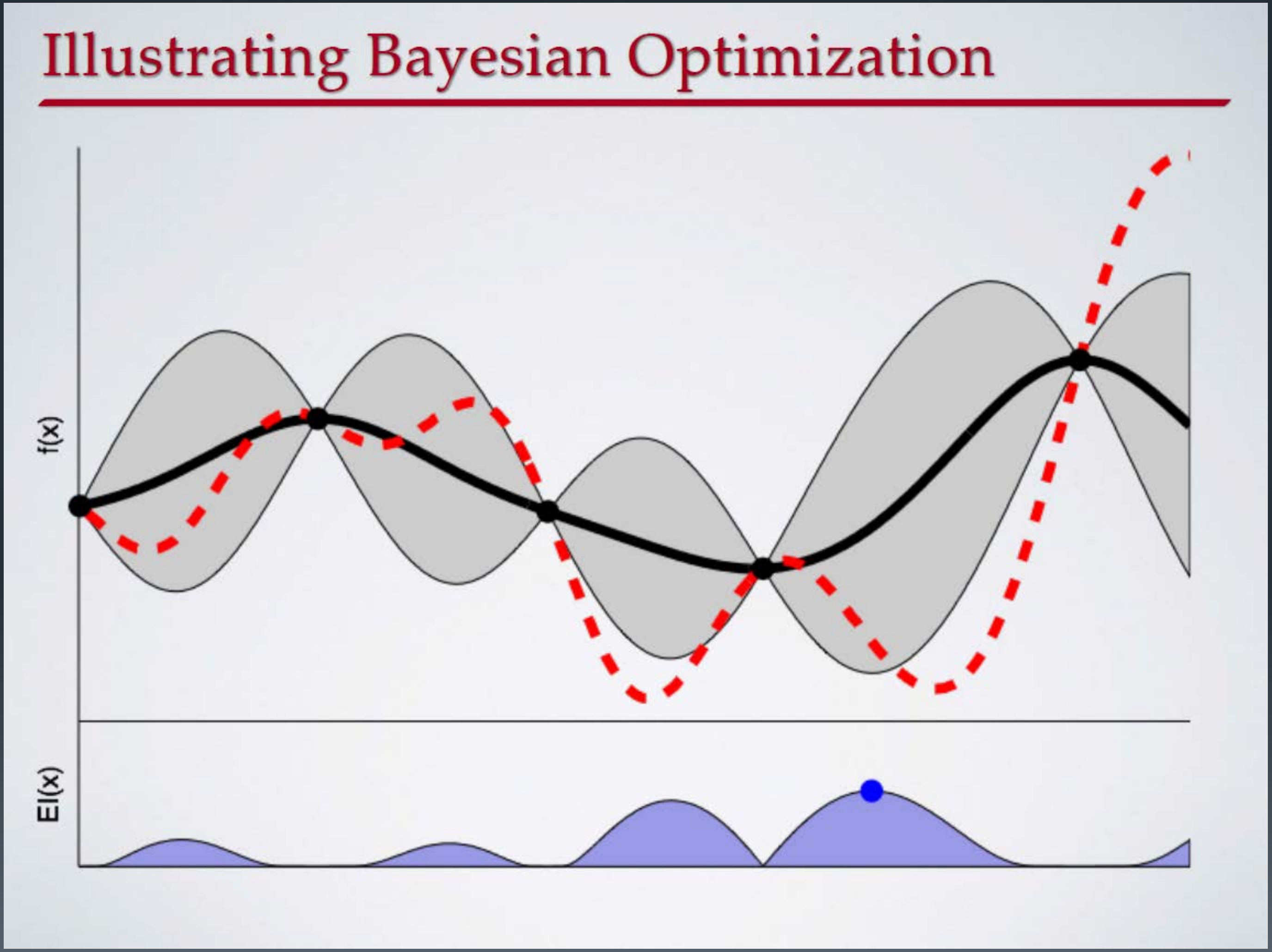
- Initialize a model $\tilde{J}(\theta, \omega)$ of the shape of the $J(\theta)$ “landscape”
  - This is typically a Gaussian process model with parameters ω
- Repeat until no further improvement:
  - Select a θ to evaluate using $\tilde{J}(\theta, \omega)$, according to an “acquisition function”
    - Typically “Expected Improvement”
  - Estimate $J(\theta)$ from one or more Monte Carlo trials
  - Update $\tilde{J}(\theta, \omega)$

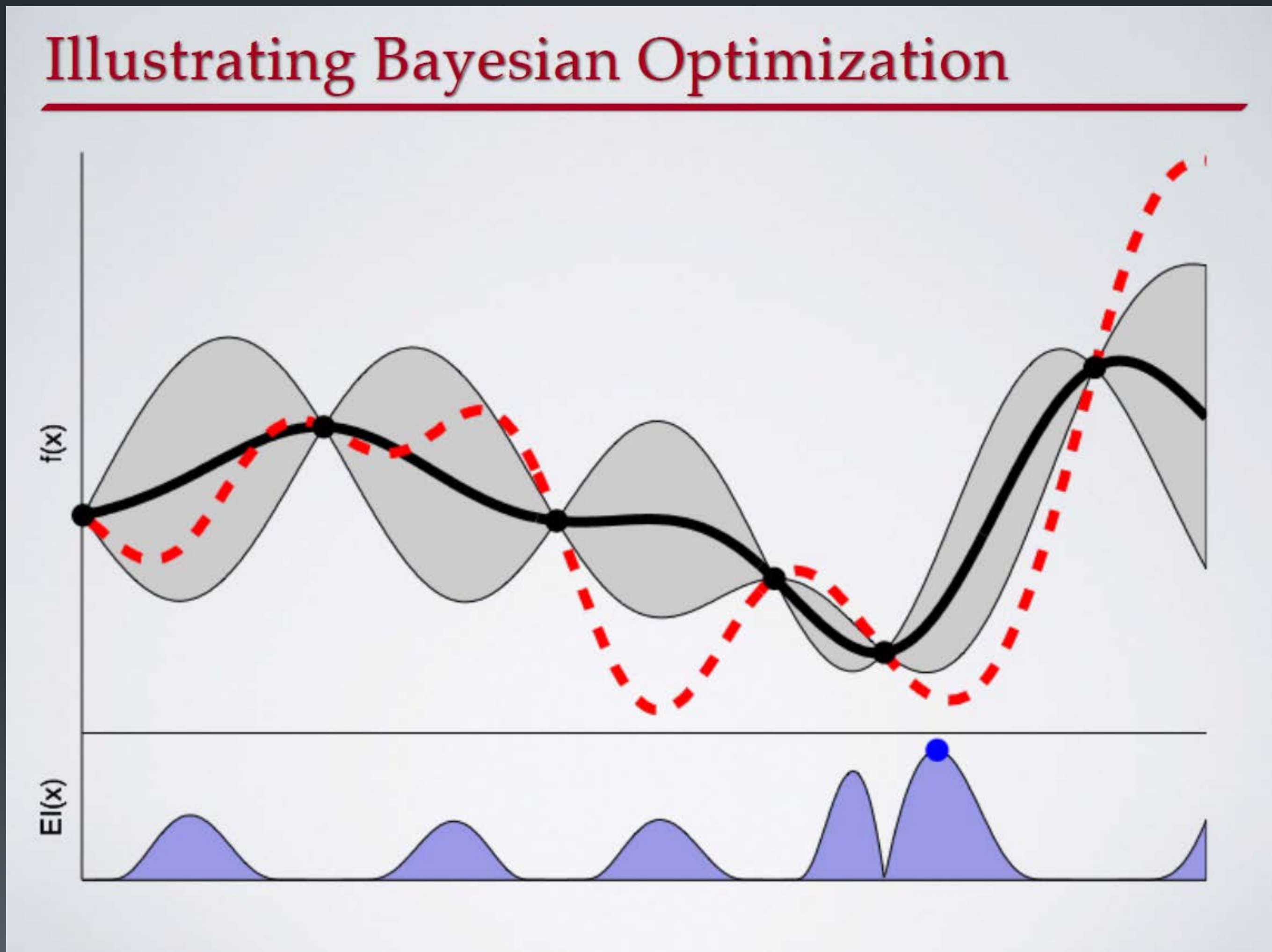
Visualization (from Ryan Adams)
Goal: minimize $f(x)$.
Blue dot: x value to sample next

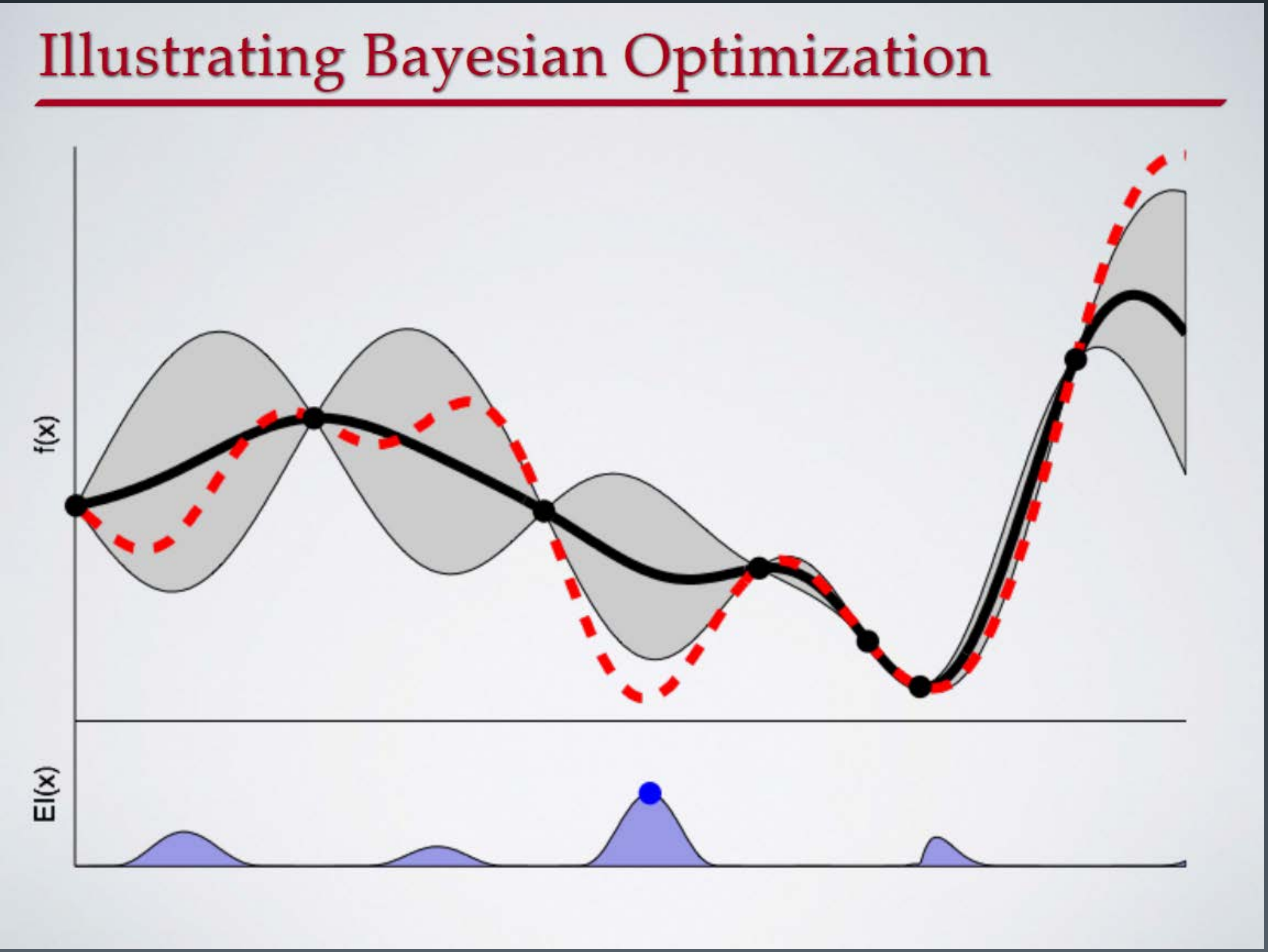


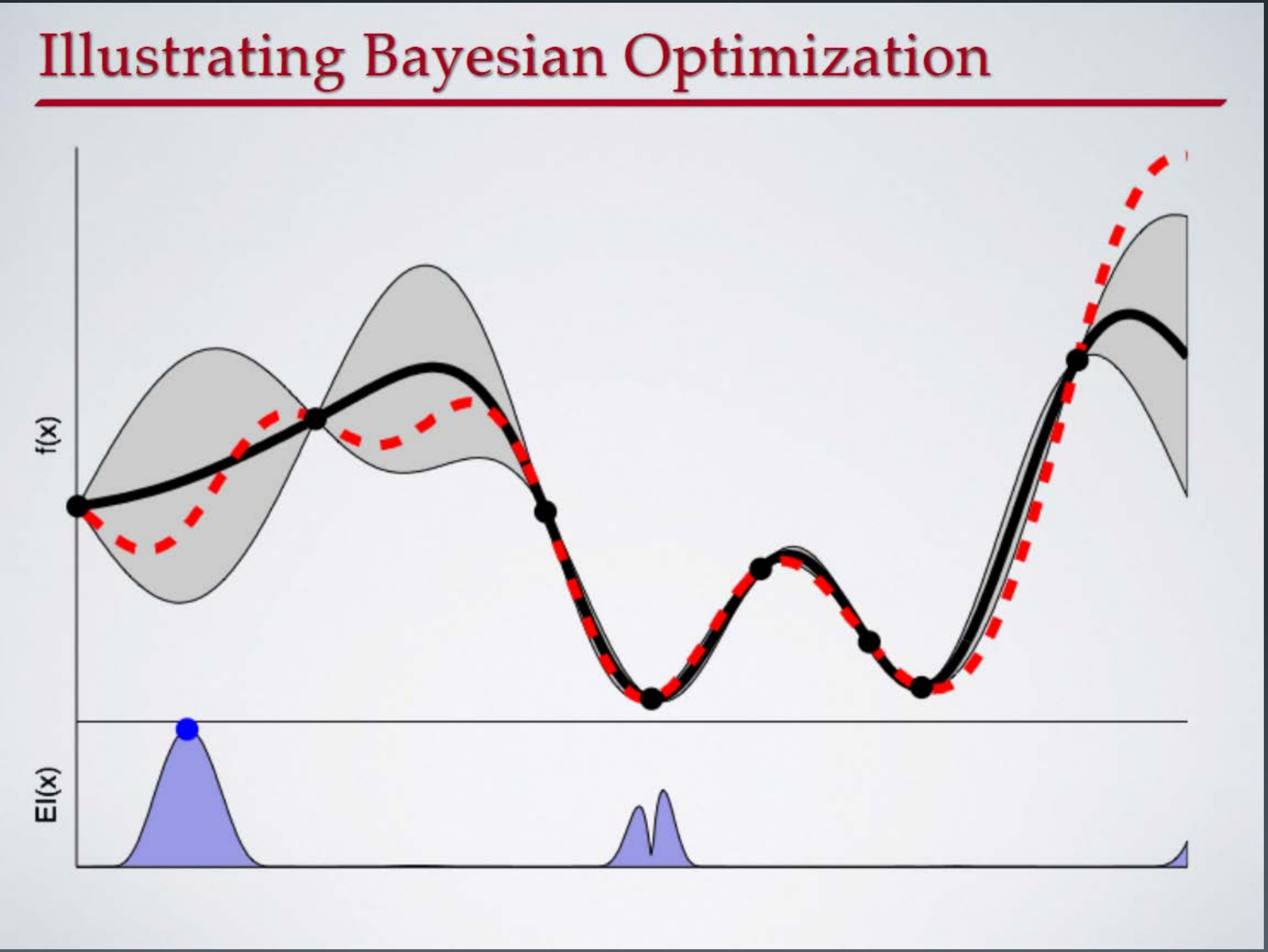

Illustrating Bayesian Optimization
f(x)
EI(x)


Illustrating Bayesian Optimization
f(x)
EI(x)


Illustrating Bayesian Optimization
f(x)
EI(x)


Illustrating Bayesian Optimization
f(x)
EI(x)


Illustrating Bayesian Optimization
f(x)
EI(x)


Illustrating Bayesian Optimization
f(x)
EI(x)

Other Direct Policy Search Methods

- CMA-ES: Covariance Matrix Adaptation-Evolution Strategies
- SMAC: Random Forest-based Method

- Note that none of these require that $\pi(s;\theta)$ be differentiable with respect to θ

Policy Gradient Methods

- Let $P(a|s) = \pi(s, a; \theta)$ be a differentiable stochastic policy (e.g. a neural network with softmax output layer)
- We can use Monte Carlo trials to estimate the gradient

$$\nabla_\theta J(\theta)$$

- We can then take a step

$$\theta \coloneqq \theta + \eta \nabla_\theta J(\theta)$$

  in the direction of the gradient to improve the policy

- We do this until the gradient is zero, which means we have reached a (local) maximum

Policy Gradient Methods

- Policy Gradient Methods
  - Let $P(a|s) = \pi(s, a; \theta)$ be a differentiable stochastic policy (e.g. a neural network with softmax output layer)

- We can obtain Monte Carlo estimates of the gradient $\nabla_\theta J(\theta)$ as follows:
  - Let H be a chosen "horizon time"
  - Starting in state s_t, select actions $a_t \sim \pi(s_t, a_t; \theta)$ to produce a trajectory $(s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, \dots, s_{t+H-1}, a_{t+H-1}, r_{t+H-1}, s_{t+H})$
  - Compute the observed cumulative discounted return along this trajectory:
  - $R_H = r_t + \gamma r_{t+1} + \cdots + \gamma^{H-1} r_{t+H-1}$
  - $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi(s_t, a_t)\, R_H$
  - $\theta \coloneqq \theta + \eta \nabla_\theta \log \pi(s_t, a_t)\, R_H$ using learning rate η
  - This is called H-step REINFORCE
  - Unfortunately, the gradient can be very noisy, so η must be very small

Actor-Critic Method

- Stabilizes REINFORCE by including the value function
  - Historically, $\pi(s, a; \theta)$ was called "The Actor"
  - and $V^{\theta}(s; \omega)$ was called "The Critic" (implemented as a second neural network)

$$\nabla_{\theta} J(\theta) \approx \nabla_{\theta} \log \pi(s_t, a_t) \left(R_H - V^{\theta}(s_t; \omega)\right)$$

- V^{θ} is an instance of a "baseline" method. These are standard methods employed to reduce the variance of Monte Carlo estimates
- Actor Critic methods work well even when the value function is quite bad

State of the Art: A3C

- $t \coloneqq 1; T \coloneqq 0$. Initialize θ and ω
- repeat
  - reset gradients: $d\theta \coloneqq 0; d\omega \coloneqq 0$
  - create "fast" copies: $\theta' \coloneqq \theta; \omega' \coloneqq \omega$
  - $t_{start} \coloneqq t$
  - repeat (generate trajectory of length H)
    - perform $a_t \sim \pi(s_t, a_t; \theta')$ receive r_t and observe s_{t+1}
    - $t \coloneqq t + 1; T \coloneqq T + 1$
  - until $(t - t_{start}) == H$
  - $R \coloneqq V(s_t; \omega')$
  - for i from $t - 1$ downto t_{start} do
    - $R \coloneqq r_i + \gamma R$
    - $d\theta \coloneqq d\theta + \nabla_{\theta'} \log \pi(s_i, a_i; \theta') \, [R - V(s_i; \omega')]$
    - $d\omega \coloneqq d\omega + \partial [R - V(s_i; \omega')]^2 / \partial \omega'$
  - $\theta \coloneqq \theta + \eta d\theta;\ \omega \coloneqq \omega + \eta d\omega$ update "slow" parameters
- until $T > T_{max}$

Distributed, Asynchronous Updates

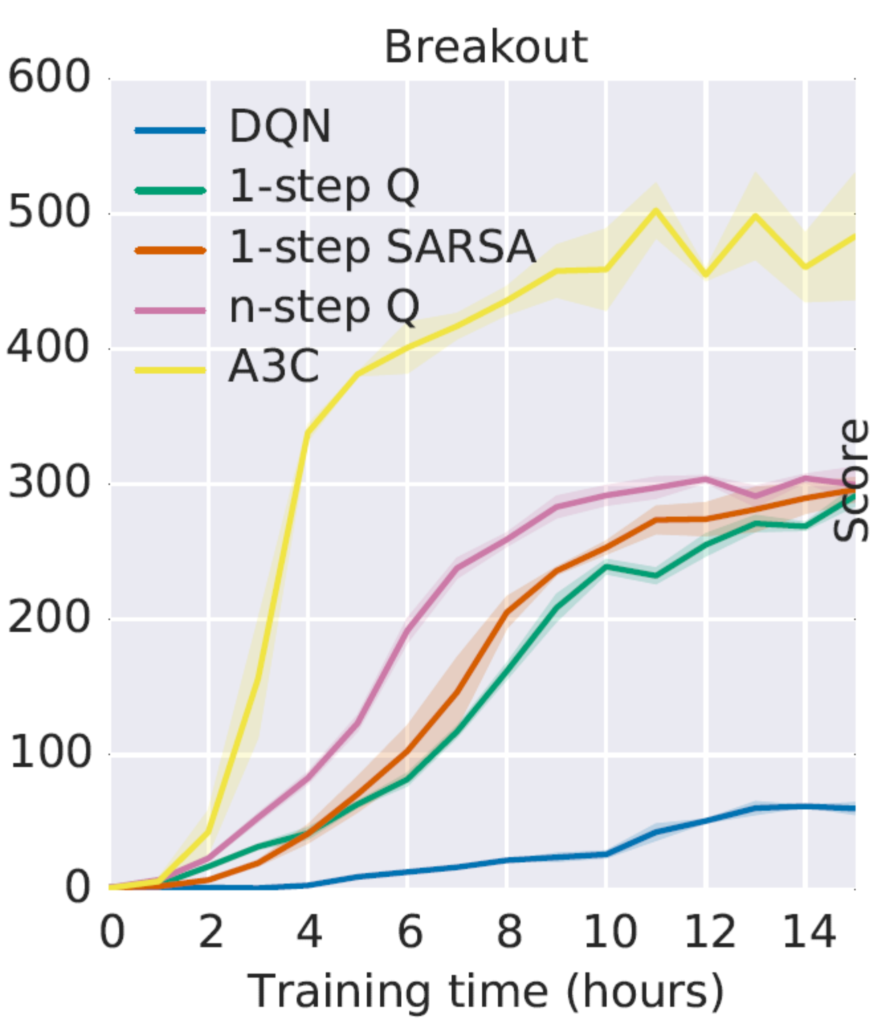
- Execute multiple copies running in parallel threads
- Shared global variables: θ, ω, T

A3C Discussion

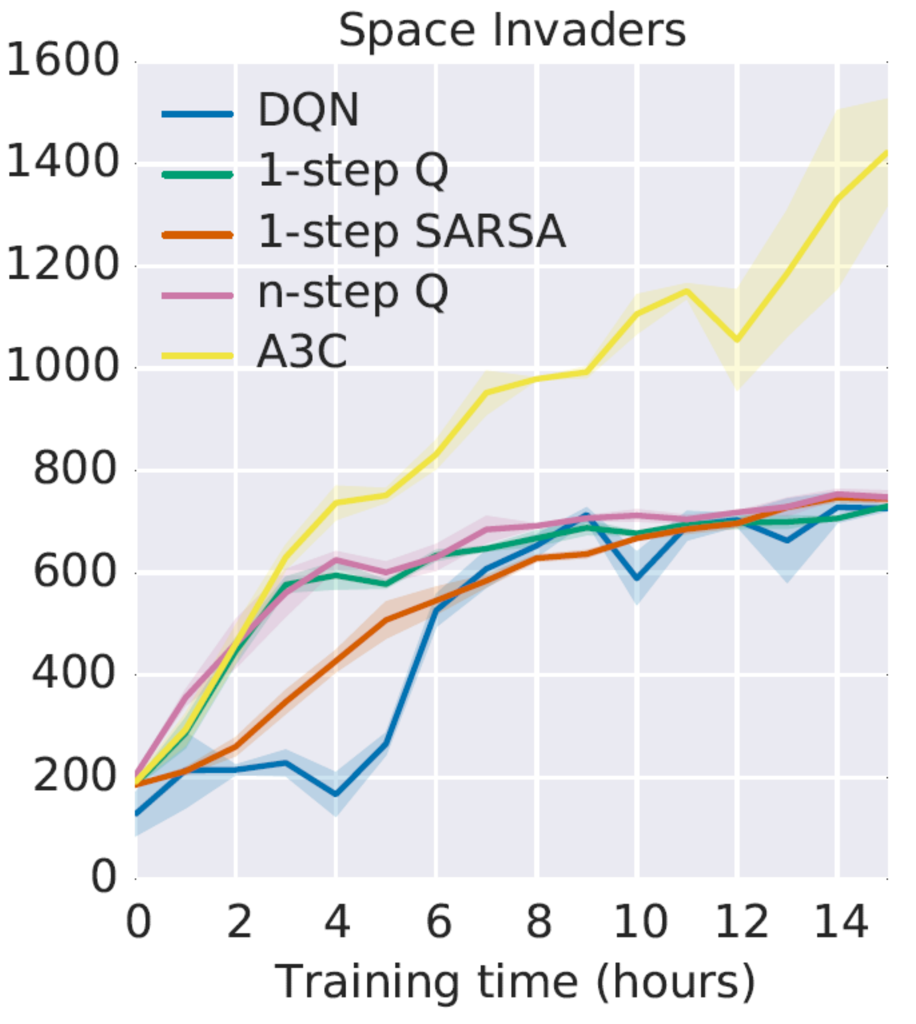
- The gradient updates within a single trajectory are highly correlated
- Combining multiple parallel threads gives a more independent (and therefore, more stable) gradient estimate

- Both the policy and the value function are trained via gradient ascent steps

Experiments on Atari Gams

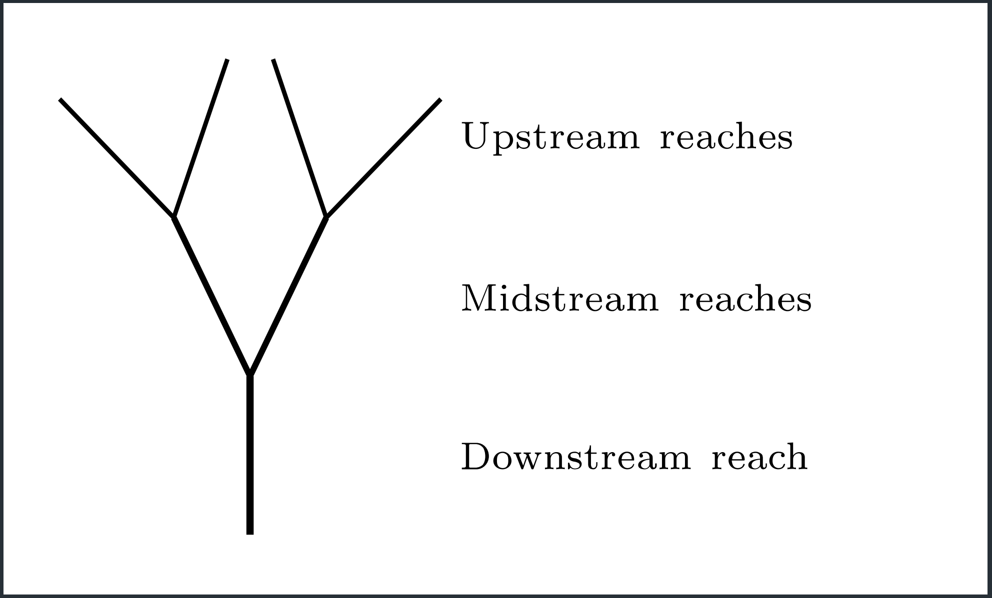


Space Invaders



Tamarisk Study

- Small network with one “slot” per edge
- Budget: 2 edges treated per time step
- For each (s, a), we invoked the simulator thousands of times to estimate $P(s'|s, a)$. We used a Clopper-Pearson confidence interval on each outcome probability and sampled until the width of the confidence interval was less than 0.01
- Then we applied value iteration to compute π^*
- Then we manually analyzed the resulting policy



Tamarisk Results

- Both (eradicate + plant native) action was performed primarily in the midstream reaches
  - Prevents invader from establishing there
  - Provides native seeds for downstream reach
  - It is not actually a barrier
- Planting native trees in the upstream reaches was sometimes chosen.
  - Serves as a source of native seeds
  - Upstream propagation of invasive seeds is rare, so it does not have much preventative effect
- If there is no upstream propagation at all, then the optimal policy uses eradication starting upstream and sweeping downstream
  - Eradication is permanent under these conditions
- If there are exogenous arrivals, then eradication by itself is weak, because an exogenous invasive seed can undo the eradication easily
  - Optimal policy focuses on planting natives everywhere, starting upstream
- In general, the optimal policy is quite complex, even for this simple river system

Summary

- Many ecosystem management problems can be formulated as Markov Decision Problems
- Known MDP (known $T(s, a, s')$): use value iteration to compute the optimal policy
- Simulator MDP or real world MDP
  - If the state and action spaces are small enough, use Q learning with a tabular representation for $Q(s, a)$
  - Else perform policy search
    - Policy Search via Bayesian Optimization
    - Policy Search via Policy Gradient Methods