

# Reinforcement Learning 3: Policy Methods and IRL

Many slides courtesy of  
Dan Klein, Stuart Russell,  
Andrew Moore or  
Alan Fern

**CS 5300 / CS 6300**  
**Artificial Intelligence**  
**Spring 2010**

Hal Daumé III  
hal@cs.utah.edu

[www.cs.utah.edu/~hal/courses/2010S\\_AI](http://www.cs.utah.edu/~hal/courses/2010S_AI)

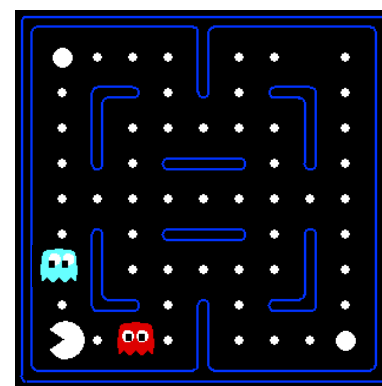
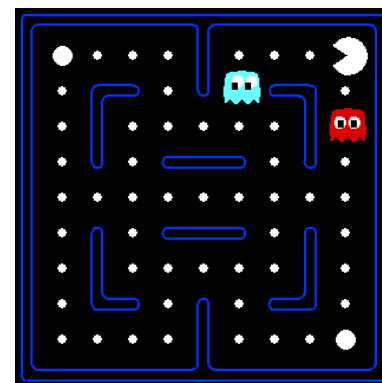
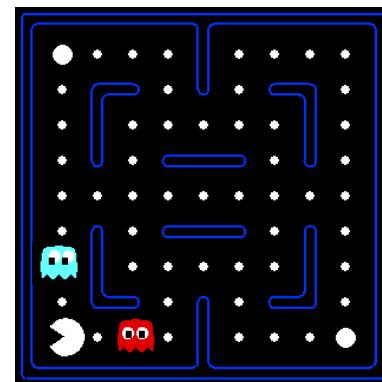
# Announcements

# What's Coming Up...

- Today: Policy methods
- 19 Feb: Final day of RL (inverse RL)
- 24 Feb: Robot motion
- 26 Feb: Probability (preparing for 2<sup>nd</sup> half of the course)
- 03 Mar: Midterm (in class, open book, like HW)
- 05 Mar...: Reasoning under uncertainty

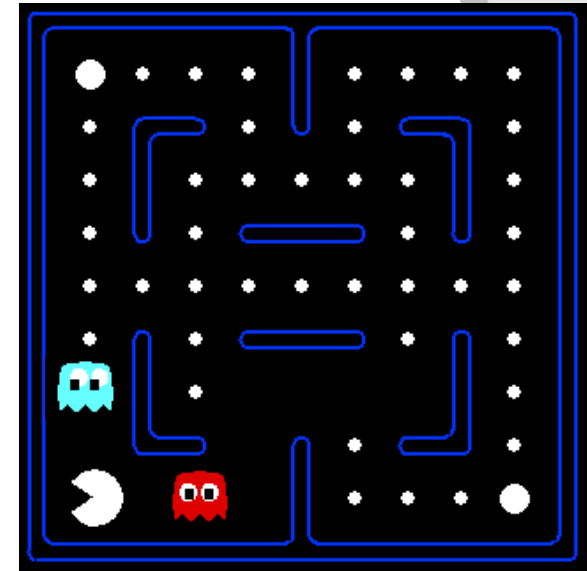
# Example: Pacman

- Let's say we discover through experience that this state is bad:
- In naïve q learning, we know nothing about this state or its q states:
- Or even this one!



# Feature-Based Representations

- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - $1 / (\text{dist to dot})^2$
    - Is Pacman in a tunnel? (0/1)
    - ..... etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



# Linear Feature Functions

- Using a feature representation, we can write a  $q$  function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but be very different in value!

# Function Approximation

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear q-functions:

$$Q(s, a) \leftarrow Q(s, a) + \alpha [\text{error}]$$

$$w_i \leftarrow w_i + \alpha [\text{error}] f_i(s, a)$$

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g. if something unexpectedly bad happens, disprefer all states with that state's features
- Formal justification: online least squares

# Example: Q-Pacman

$$Q(s, a) = 4.0 f_{DOT}(s, a) - 1.0 f_{GST}(s, a)$$

$$f_{DOT}(s, \text{NORTH}) = 0.5$$

$$f_{GST}(s, \text{NORTH}) = 1.0$$

$$Q(s, a) = +1$$

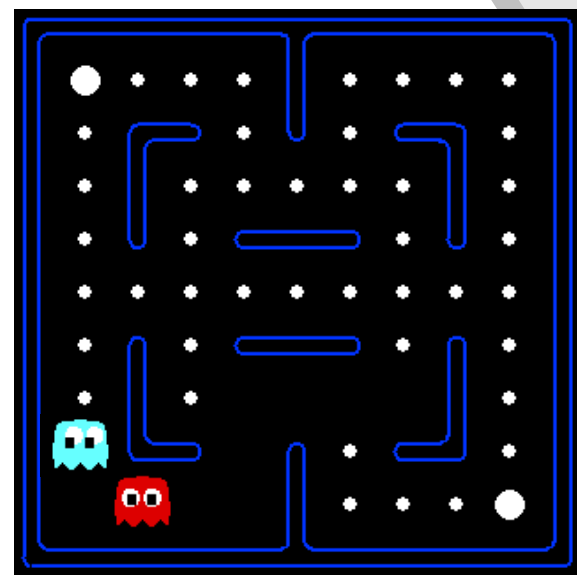
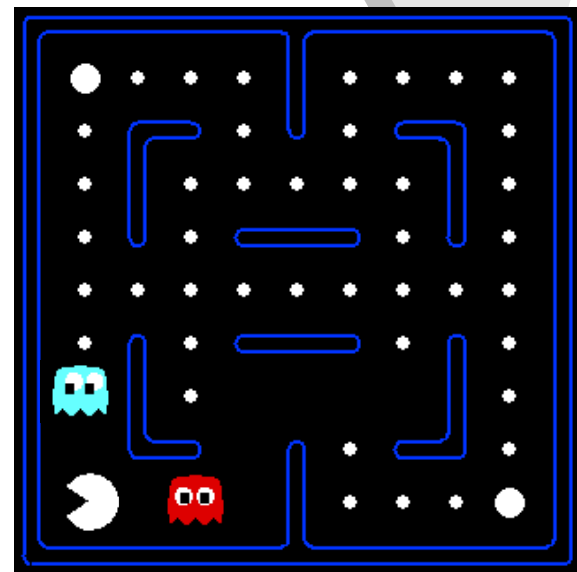
$$R(s, a, s') = -500$$

$$\text{error} = -501$$

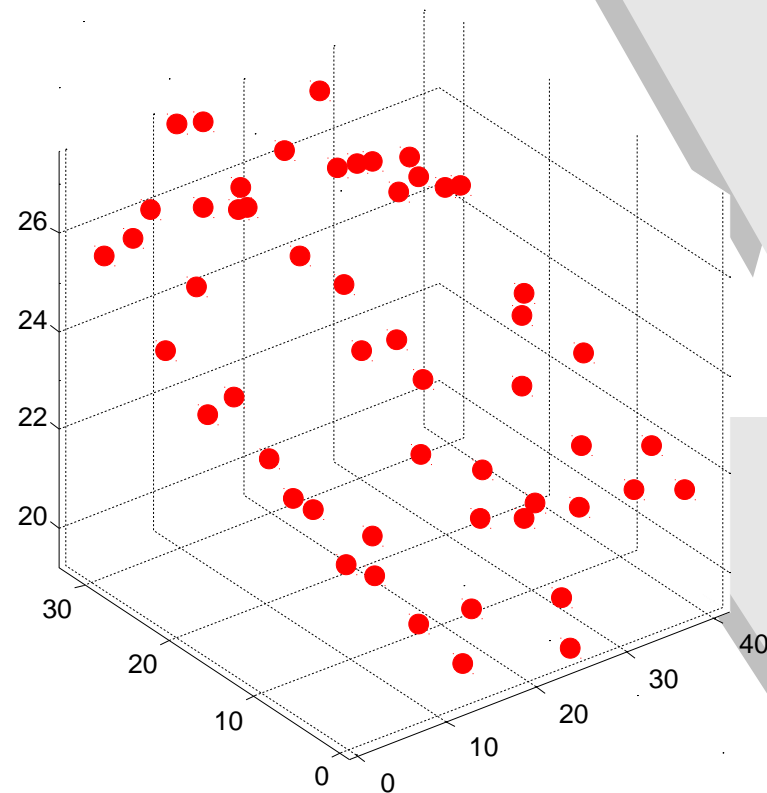
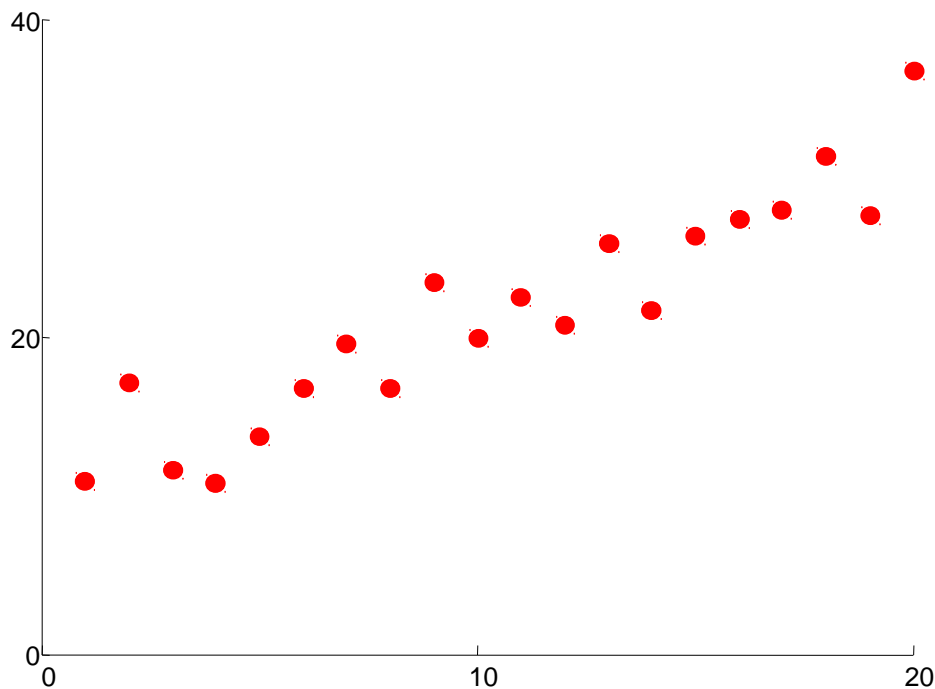
$$w_{DOT} \leftarrow 4.0 + \alpha [-501] 0.5$$

$$w_{GST} \leftarrow -1.0 + \alpha [-501] 1.0$$

$$Q(s, a) = 3.0 f_{DOT}(s, a) - 3.0 f_{GST}(s, a)$$



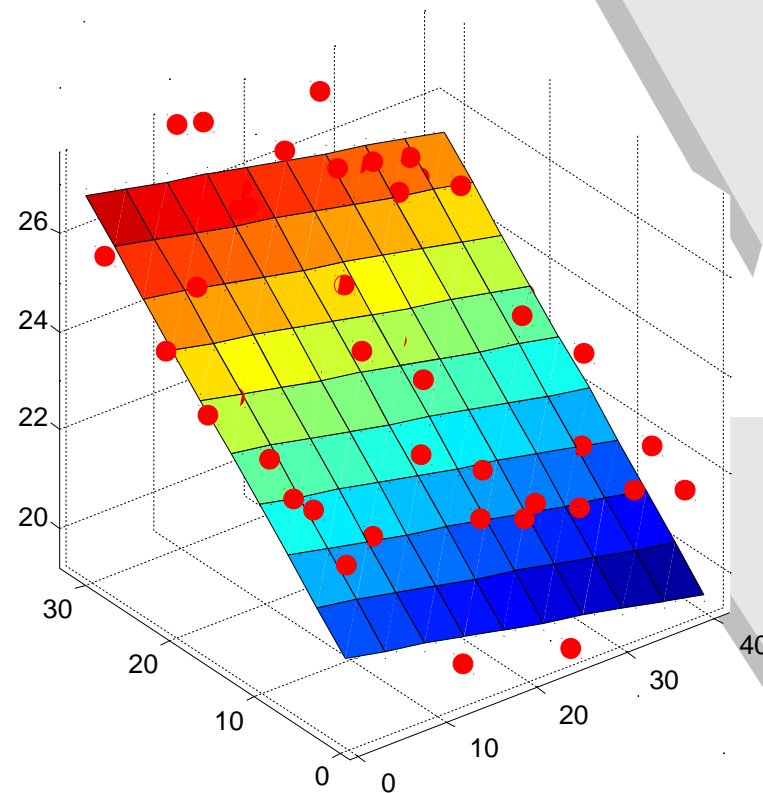
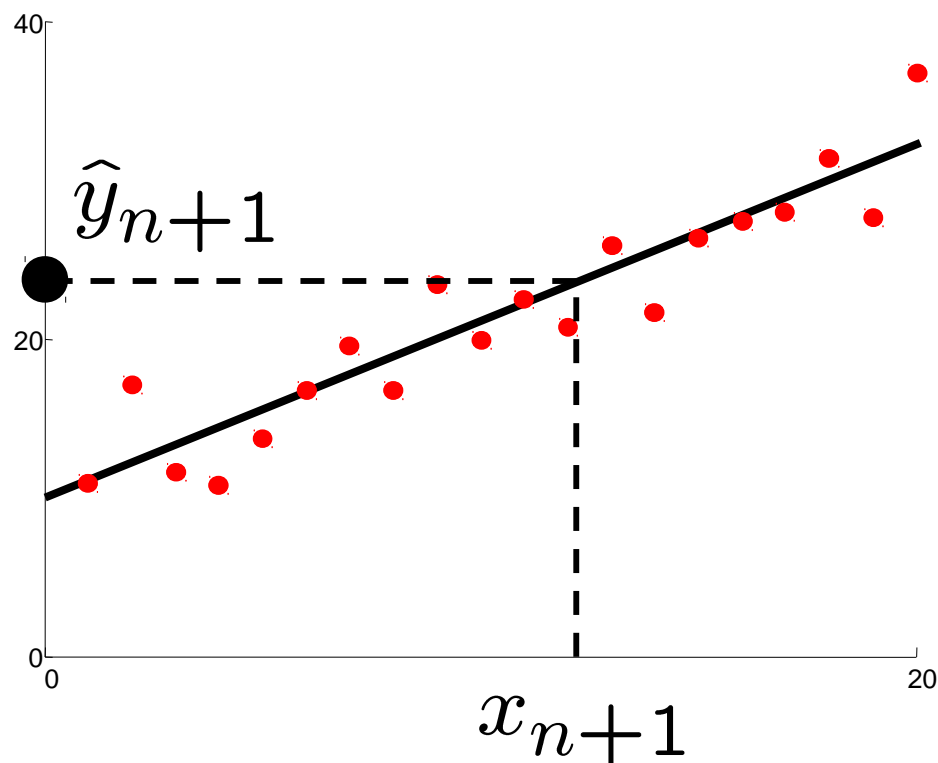
# Linear regression



Given examples  $(x_i, y_i)_{i=1 \dots n}$

Predict  $y_{n+1}$  given a new point  $x_{n+1}$

# Linear regression



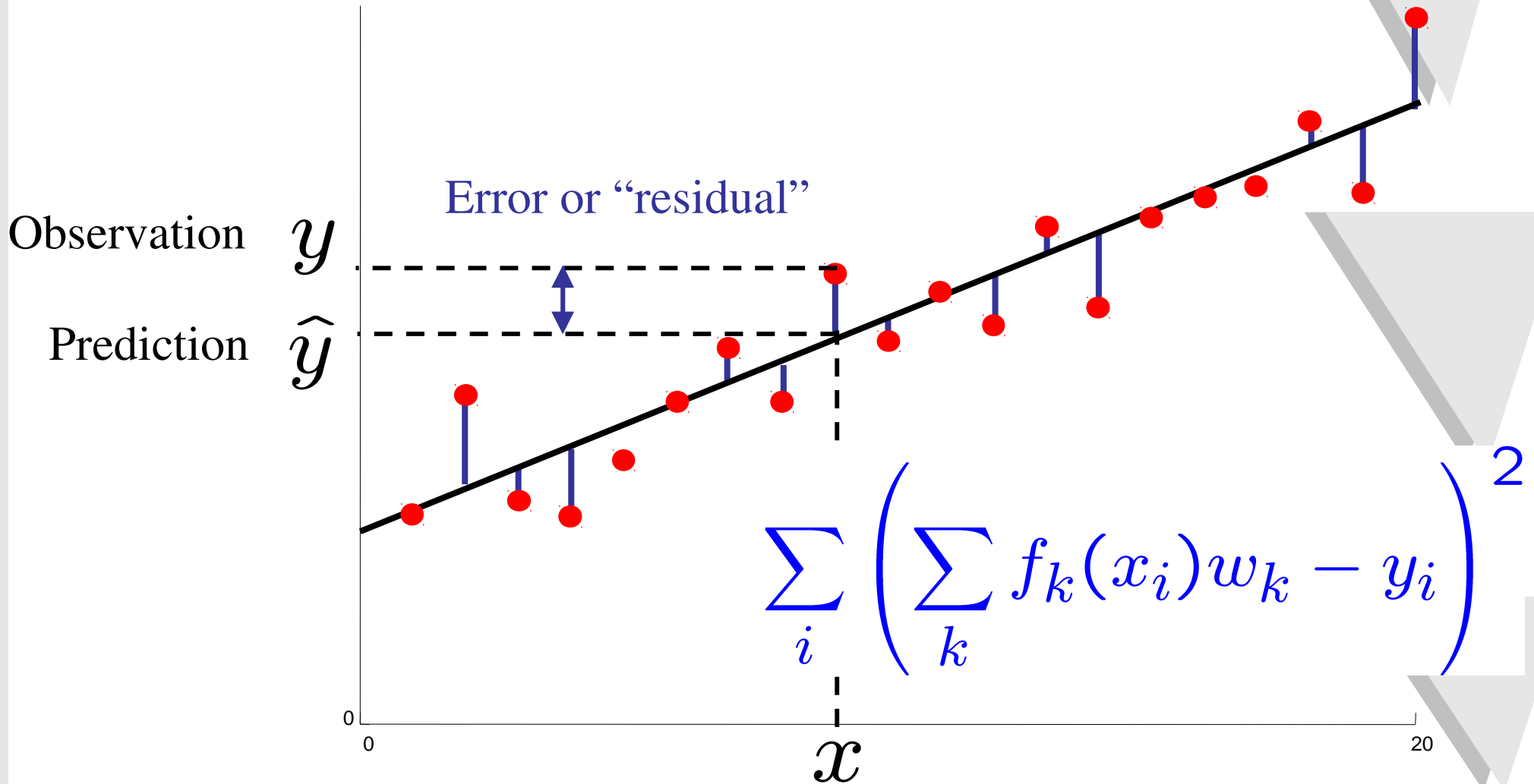
Prediction

$$\hat{y}_i = w_0 + w_1 x_i$$

Prediction

$$\hat{y}_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2}$$

# Ordinary Least Squares (OLS)



# Minimizing Error

$$E(w) = \frac{1}{2} \sum_i \left( \sum_k f_k(x_i) w_k - y_i \right)^2$$

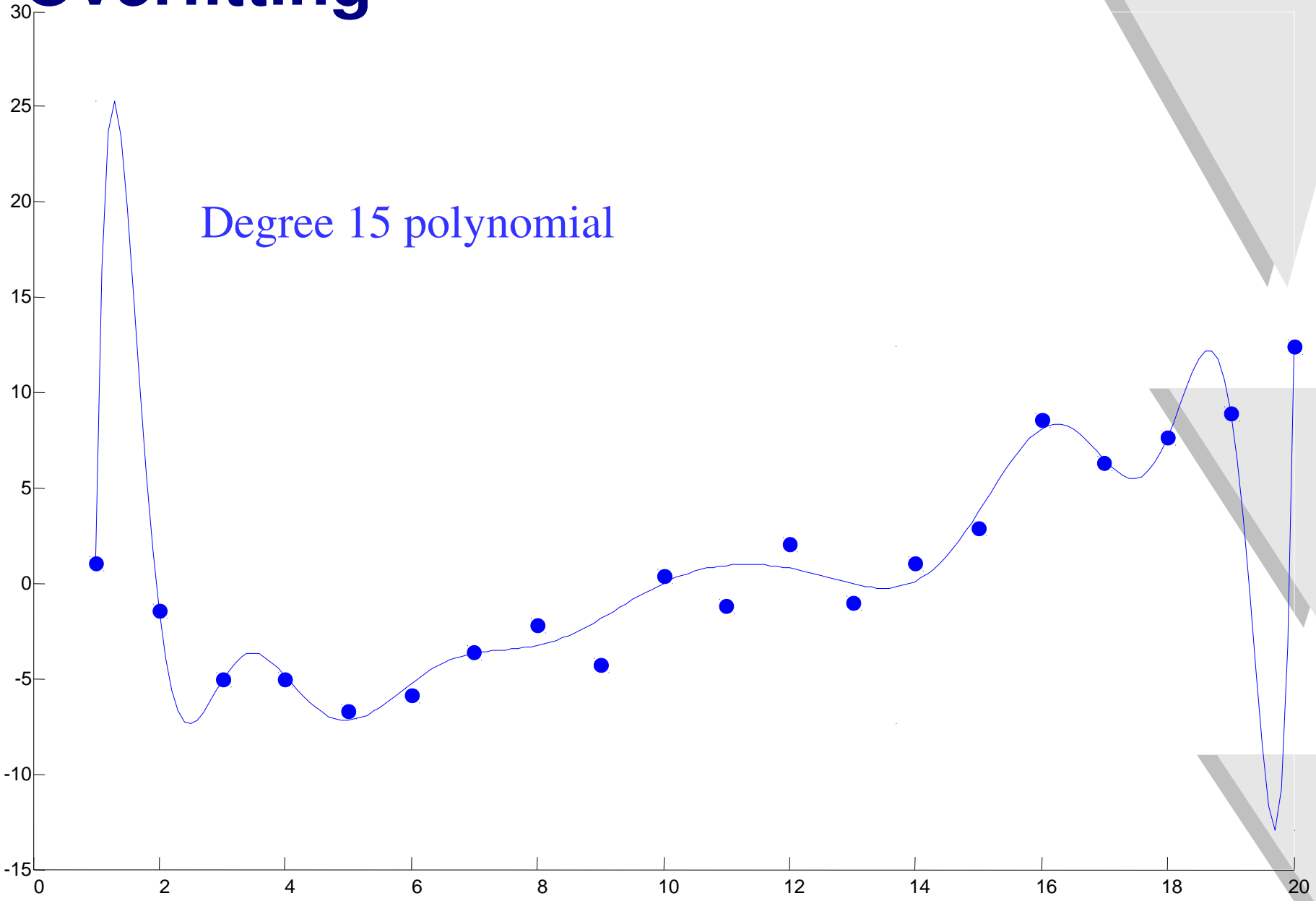
$$\frac{\partial E}{\partial w_m} = \sum_i \left( \sum_k f_k(x_i) w_k - y_i \right) f_m(x_i)$$

$$E \leftarrow E + \alpha \sum_i \left( \sum_k f_k(x_i) w_k - y_i \right) f_m(x_i)$$

Value update explained:

$$w_i \leftarrow w_i + \alpha [\text{error}] f_i(s, a)$$

# Overfitting



# RL via Policy Gradient Search

- So far all of our RL techniques have tried to learn an exact or approximate utility function or Q-function
  - I.e. learn the optimal “value” of being in a state, or taking an action from a state.
- Value functions can often be much more complex to represent than the corresponding policy
  - Do we really care about knowing  $Q(s, \text{left}) = 0.3554$ ,  $Q(s, \text{right}) = 0.533$
  - Or just that “right is better than left in state  $s$ ”
- Motivates searching directly in a parameterized policy space

# Policy Search

- Simplest policy search:
  - Start with an initial linear value function or q-function
  - Nudge each feature weight up and down and see if your policy is better than before
  
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical

# Aside: Gradient Ascent

- Given a function  $f(\theta_1, \dots, \theta_n)$  of  $n$  real values  $\theta = (\theta_1, \dots, \theta_n)$  suppose we want to maximize  $f$  with respect to  $\theta$
- A common approach to doing this is gradient ascent
- The gradient of  $f$  at point  $\theta$ , denoted by  $\nabla_{\theta} f(\theta)$ , is an  $n$ -dimensional vector that points in the direction where  $f$  increases most steeply at point  $\theta$
- Vector calculus tells us that  $\nabla_{\theta} f(\theta)$  is just a vector of partial derivatives

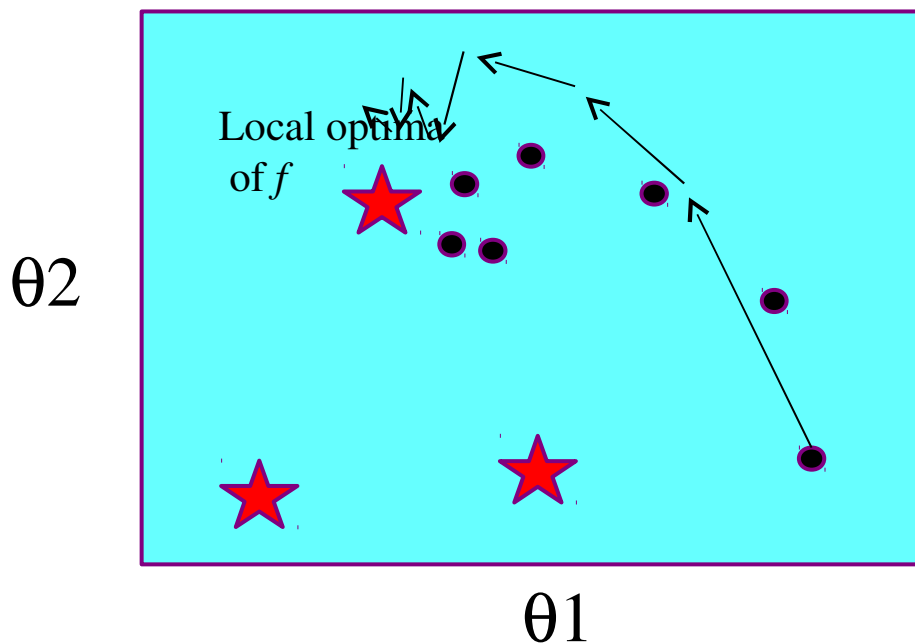
$$\nabla_{\theta} f(\theta) = \left[ \frac{\partial f(\theta)}{\partial \theta_1}, \dots, \frac{\partial f(\theta)}{\partial \theta_n} \right]$$

where 
$$\frac{\partial f(\theta)}{\partial \theta_i} = \lim_{\varepsilon \rightarrow 0} \frac{f(\theta_1, \dots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \dots, \theta_n) - f(\theta)}{\varepsilon}$$

# Aside: Gradient Ascent

- Gradient ascent iteratively follows the gradient direction starting at some initial point
- Initialize  $\theta$  to a random value
- Repeat until stopping condition

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} f(\theta)$$



With proper decay of learning rate gradient descent is guaranteed to converge to local optima.

# RL via Policy Gradient Ascent

- This general approach has the following components
  1. Select a space of parameterized policies:
  2. Compute the gradient of the value function of the policy wrt parameters
  3. Move parameters in the direction of the gradient
  4. Repeat these steps until we reach a local maxima
  
- So we must answer the following questions:
  - How should we represent parameterized policies?
  - How can we compute the gradient?

# Parameterized Policies

- One example of a space of parametric policies is:

$$\pi_{\theta}(s) = \arg \max_a \hat{Q}_{\theta}(s, a)$$

where  $\hat{Q}_{\theta}(s, a)$  may be a linear function, e.g.

$$\hat{Q}_{\theta}(s, a) = \theta_0 + \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \dots + \theta_n f_n(s, a)$$

- The goal is to learn parameters  $\theta$  that give a good policy
- Note that it is not important that  $\hat{Q}_{\theta}(s, a)$  be close to the actual Q-function  $\hat{Q}_{\theta}(s, a)$ 
  - Rather we only require  $\hat{Q}_{\theta}(s, a)$  is good at ranking actions in order of goodness

# Policy Gradient Ascent

- Let  $\rho(\theta)$  be the expected value of policy  $\pi_\theta$ .
  - $\rho(\theta)$  is just the expected discounted total reward for a trajectory of  $\pi_\theta$ .
  - For simplicity assume each trajectory starts at a single initial state.
- Our objective is to find a  $\theta$  that maximizes  $\rho(\theta)$
- Policy gradient ascent tells us to iteratively update parameters via:  $\theta \leftarrow \theta + \alpha \nabla_\theta \rho(\theta)$
- **Problem:**  $\rho(\theta)$  is generally very complex and it is rare that we can compute a closed form for the gradient of  $\rho(\theta)$ .
- We will instead estimate the gradient based on experience

# Gradient Estimation

- **Concern**: Computing or estimating the gradient of discontinuous functions can be problematic.

- For our example parametric policy

$$\pi_{\theta}(s) = \arg \max_a \hat{Q}_{\theta}(s, a)$$

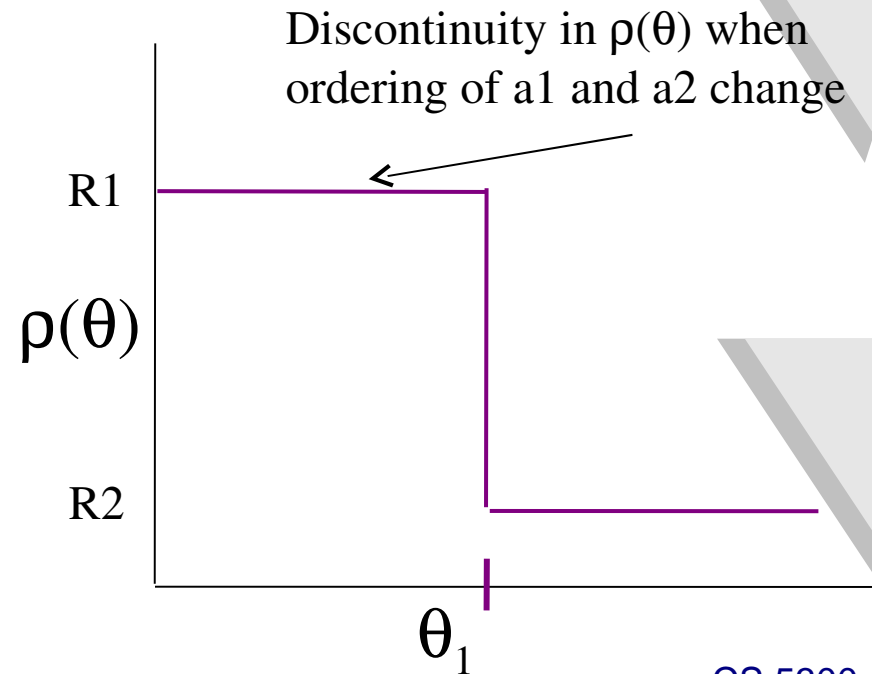
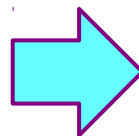
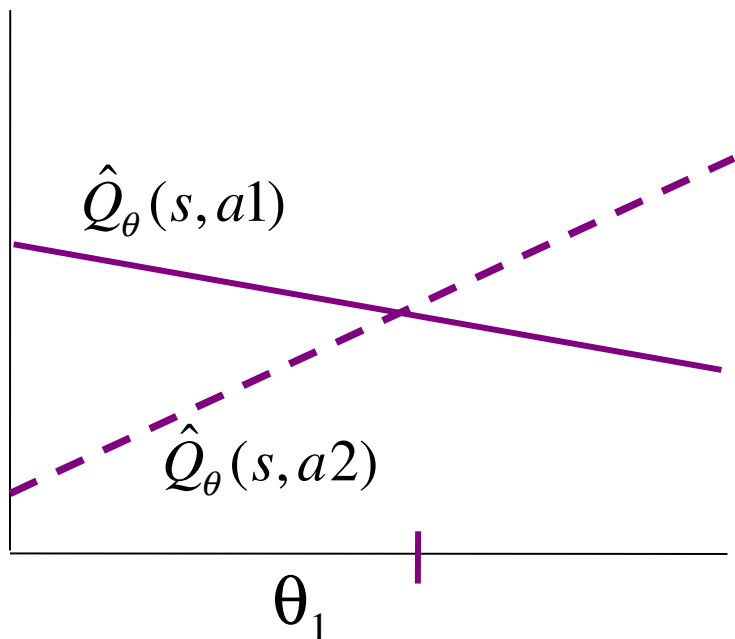
is  $\rho(\theta)$  continuous?

- No.
  - There are values of  $\theta$  where arbitrarily small changes, cause the policy to change.
  - Since different policies can have different values this means that changing  $\theta$  can cause discontinuous jump of  $\rho(\theta)$ .

# Example: Discontinuous $\rho(\theta)$

$$\pi_{\theta}(s) = \arg \max_a \hat{Q}_{\theta}(s, a) = \theta_1 f_1(s, a)$$

- Consider a problem with initial state  $s$  and two actions  $a_1$  and  $a_2$ 
  - $a_1$  leads to a very large terminal reward  $R_1$
  - $a_2$  leads to a very small terminal reward  $R_2$
- Fixing  $\theta_2$  to a constant we can plot the ranking assigned to each action by  $Q$  and the corresponding value  $\rho(\theta)$



# Probabilistic Policies

- We would like to avoid policies that drastically change with small parameter changes, leading to discontinuities
- A **probabilistic policy**  $\pi_\theta$  takes a state as input and returns a distribution over actions
  - Given a state  $s$   $\pi_\theta(s,a)$  returns the probability that  $\pi_\theta$  selects action  $a$  in  $s$
- Note that  $\rho(\theta)$  is still well defined for probabilistic policies
  - Now uncertainty of trajectories comes from environment and policy
  - Importantly if  $\pi_\theta(s,a)$  is continuous relative to changing  $\theta$  then  $\rho(\theta)$  is also continuous relative to changing  $\theta$
- A common form for probabilistic policies is the **softmax function** or **Boltzmann exploration function**

$$\pi_\theta(s, a) = \Pr(a | s) = \frac{\exp(\hat{Q}_\theta(s, a))}{\sum_{a' \in A} \exp(\hat{Q}_\theta(s, a'))}$$

# Empirical Gradient Estimation

- Our first approach to estimating  $\nabla_{\theta} \rho(\theta)$  is to simply compute empirical gradient estimates
- Recall that  $\theta = (\theta_1, \dots, \theta_n)$  and 
$$\nabla_{\theta} \rho(\theta) = \left[ \frac{\partial \rho(\theta)}{\partial \theta_1}, \dots, \frac{\partial \rho(\theta)}{\partial \theta_n} \right]$$

so we can compute the gradient by empirically estimating each partial derivative

$$\frac{\partial \rho(\theta)}{\partial \theta_i} = \lim_{\varepsilon \rightarrow 0} \frac{\rho(\theta_1, \dots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \dots, \theta_n) - \rho(\theta)}{\varepsilon}$$

- So for small  $\varepsilon$  we can estimate the partial derivatives by

$$\frac{\rho(\theta_1, \dots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \dots, \theta_n) - \rho(\theta)}{\varepsilon}$$

- This requires estimating  $n+1$  values:

$$\rho(\theta), \quad \left\{ \rho(\theta_1, \dots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \dots, \theta_n) \mid i = 1, \dots, n \right\}$$

# Empirical Gradient Estimation

- How do we estimate the quantities

$$\rho(\theta), \quad \left\{ \rho(\theta_1, \dots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \dots, \theta_n) \mid i = 1, \dots, n \right\}$$

- For each set of parameters, simply execute the policy for N trials/episodes and average the values achieved across the trials
- This requires a total of  $N(n+1)$  episodes to get gradient estimate
  - For stochastic environments and policies the value of N must be relatively large to get good estimates of the true value
  - Often we want to use a relatively large number of parameters
  - Often it is expensive to run episodes of the policy
- So while this can work well in many situations, it is often not a practical approach computationally

# Gradient Estimation: Single Step Problems

- For stochastic policies it is possible to estimate  $\nabla_{\theta} \rho(\theta)$  directly from trajectories of just the current policy  $\pi_{\theta}$ 
  - Idea: take advantage of the fact that we know the functional form of the policy

- First consider the simplified case where all trials have length 1

- For simplicity assume each trajectory starts at a single initial state and reward only depends on action choice

- $\rho(\theta)$  is just the expected reward of action selected by  $\pi_{\theta}$ .

$$\rho(\theta) = \sum_a \pi_{\theta}(s_o, a) R(a)$$

where  $s_o$  is the initial state and  $R(a)$  is reward of action  $a$

- The gradient of this becomes

$$\nabla_{\theta} \rho(\theta) = \nabla_{\theta} \sum_a \pi_{\theta}(s_o, a) R(a) = \sum_a (\nabla_{\theta} \pi_{\theta}(s_o, a)) R(a)$$

- How can we estimate this by just observing the execution of  $\pi_{\theta}$ ?

# Gradient Estimation: Single Step Problems

➤ Rewriting 
$$\begin{aligned} \nabla_{\theta} \rho(\theta) &= \sum_a (\nabla_{\theta} \pi_{\theta}(s_o, a)) R(a) \\ &= \sum_a \pi_{\theta}(s_o, a) \frac{(\nabla_{\theta} \pi_{\theta}(s_o, a))}{\pi_{\theta}(s_o, a)} R(a) \\ &= \sum_a \pi_{\theta}(s_o, a) \underbrace{\nabla_{\theta} \log(\pi_{\theta}(s_o, a))}_{g(s_o, a)} R(a) \end{aligned}$$

can get closed form  $g(s_o, a)$

➤ The gradient is just the expected value of  $g(s_o, a)R(a)$  over execution trials of  $\pi_{\theta}$

➤ Can estimate by executing  $\pi_{\theta}$  for  $N$  trials and

$$\nabla_{\theta} \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^N g(s_o, a_j) R(a_j)$$

$a_j$  is action selected by policy on  $j$ 'th episode

➤ Only requires executing  $\pi_{\theta}$  for a number of trials that need not depend on the number

# Gradient Estimation: General Case

- So for the case of a length 1 trajectories we got:

$$\nabla_{\theta} \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^N g(s_o, a_j) R(a)$$

- For the general case where trajectories have length greater than one and reward depends on state we can do some work and get:

$$\nabla_{\theta} \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^N \sum_{t=1}^{T_j} g(s_{j,t}, a_{j,t}) R_j(s_{j,t})$$

length of trial j

Observed total reward in trial j from step t to end

- $s_{jt}$  is t'th state of j'th episode,  $a_{jt}$  is t'th action of episode j
- The derivation of this is straightforward but messy.

# Gradient Estimation: General Case

- How can we interpret this gradient expression:

$$g(s, a) = \nabla_{\theta} \log(\pi_{\theta}(s, a))$$

Direction to move parameters in order to increase the probability that policy selects  $a_t$  in state  $s_t$

$$\nabla_{\theta} \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^N \sum_{t=1}^{T_j} g(s_{j,t}, a_{j,t}) R_j(s_{j,t})$$

Total reward observed after taking  $a_t$  in state  $s_t$

- So the overall gradient is a reward weighted combination of individual gradient directions
- Intuitively this increases probability of actions in states that were followed by large rewards and decreases the probability of actions in states that were followed by negative rewards

# Basic Policy Gradient Algorithm

➤ Repeat until stopping condition

5 Execute  $\pi_\theta$  for N trajectories while storing the state, action, reward sequences

5 
$$\nabla_\theta \leftarrow \frac{1}{N} \sum_{j=1}^N \sum_{t=1}^{T_j} g(s_{j,t}, a_{j,t}) R_j(s_{j,t})$$

5 
$$\theta \leftarrow \theta + \alpha \nabla_\theta$$

➤ One disadvantage of this approach is the small number of updates per amount of experience

□ Also requires a notion of trajectory rather than an infinite sequence of experience

➤ Online policy gradient algorithms perform updates after each step in environment (often learn faster)



# Gradient Estimation: General Case

- Both algorithms require computation of

$$g(s, a) = \nabla_{\theta} \log(\pi_{\theta}(s, a))$$

- For the Boltzmann distribution with linear approximation we have:

$$\pi_{\theta}(s, a) = \frac{\exp(\hat{Q}_{\theta}(s, a))}{\sum_{a' \in A} \exp(\hat{Q}_{\theta}(s, a'))}$$

where

$$\hat{Q}_{\theta}(s, a) = \theta_0 + \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \dots + \theta_n f_n(s, a)$$

- Here the partial derivatives needed for  $g(s, a)$  are:

$$\frac{\partial \log(\pi_{\theta}(s, a))}{\partial \theta_i} = f_i(s, a) - \sum_{a'} \pi_{\theta}(s, a') f_i(s, a')$$

# Inverse RL: Motivation

- Given: (1) measurements of an agent's behavior over time, in a variety of circumstances, (2) if needed, measurements of the sensory inputs to that agent; (3) if available, a model of the environment.
- Determine: the reward function being optimized.

# Why?

- Reason #1: Computational models for animal and human learning.
- “In examining animal and human behavior we must consider the reward function as an unknown to be ascertained through empirical investigation.”
- Particularly true of multiattribute reward functions (e.g. Bee foraging: amount of nectar vs. flight time vs. risk from wind/predators)

# Why?

- Reason #2: Agent construction.
- “An agent designer [...] may only have a very rough idea of the reward function whose optimization would generate 'desirable' behavior.”
- e.g. “Driving well”
- Apprenticeship learning: Recovering expert's underlying reward function more “parsimonious” than learning expert's policy?

# Applications in multi-agent systems

- In multi-agent adversarial games, learning opponents' reward functions that guide their actions to devise strategies against them.
- In mechanism design, learning each agent's reward function from histories to manipulate its actions.
- and more?

# IRL: Finite State Space

$$(P_{a_1} - P_a)(I - \gamma P_{a_1})^{-1} R \geq 0, \forall a \in A \setminus a_1$$

There are many solutions of  $R$  that satisfy the inequality (e.g.  $R = 0$ ), which one might be the best solution?

1. Make deviation from  $\pi$  as costly as possible:

$$\sum_{s \in S} (Q^\pi(s, a_1) - \max_{a \in A \setminus a_1} Q^\pi(s, a))$$

2. Make reward function as simple as possible

# IRL from Sample Trajectories

- If  $\pi$  is only accessible through a set of sampled trajectories (e.g. driving demo)
- Assume we start from a dummy state  $s_0$  (whose next state distribution is according to  $D$ ).
- In the case that reward  $R = \phi_i$  trajectory state sequence  $(s_0, s_1, s_2, \dots)$ :

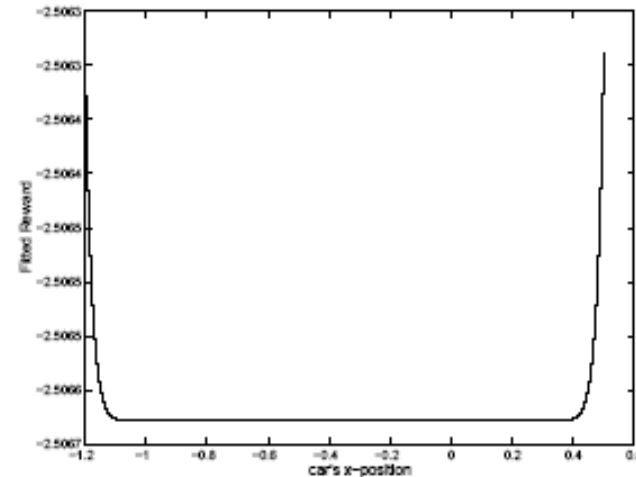
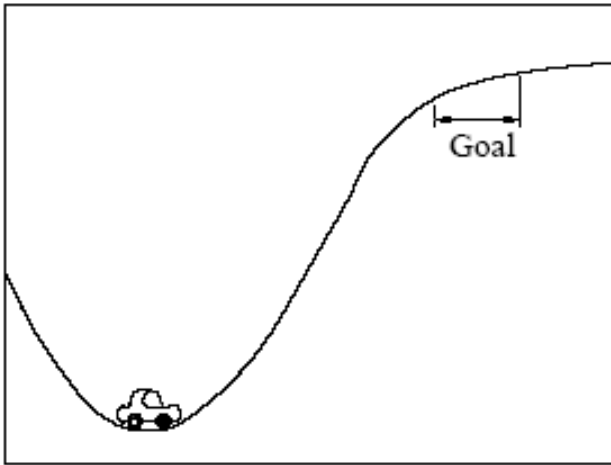
$$V_i^{\pi}(s_0) = \phi_i(s_0) + \gamma \phi_i(s_1) + \gamma^2 \phi_i(s_2) + \dots$$

$$V^{\pi}(s_0) = \alpha_1 V_1^{\pi}(s_0) + \dots + \alpha_d V_d^{\pi}(s_0)$$

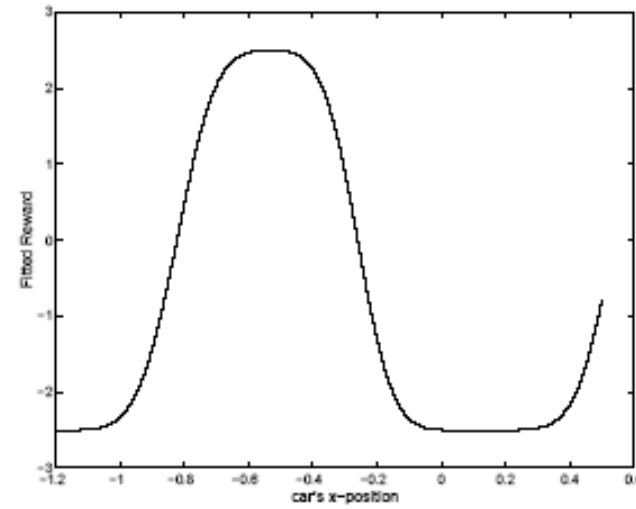
# Mountain Car Experiment #2

- Goal is in bottom of valley
- Car starts... not sure. Top of hill?
- Reward is 1 in the goal area, 0 elsewhere
- **$\gamma = 0.99$**
- State = car's  $x$ -position & velocity (continuous!)
- Function approx. class: all linear combinations of 26 evenly spaced Gaussian-shaped basis functions

# Mountain Car Results



#1



#2

# Apprenticeship Learning via IRL

- For  $t = 1, 2, \dots$ 
  - **Inverse RL step:**
  - Estimate expert's reward function  $R(s) = w^T \phi(s)$  such that under  $R(s)$  the expert performs better than all previously found policies  $\{\pi_i\}$ .
    - **RL step:**
    - Compute optimal policy  $\pi_t$  for
    - the estimated reward  $w$ .

# Gridworld Experiment

- 128 x 128 grid world divided into 64 regions, each of size 16 x 16 (“macrocells”).
- A small number of macrocells have positive rewards.
- For each macrocell, there is one feature  $\Phi_i(s)$  indicating whether that state  $s$  is in macrocell  $i$
- *Algorithm was also run on the subset of features  $\Phi_i(s)$  that correspond to non-zero rewards.*

# Gridworld Results

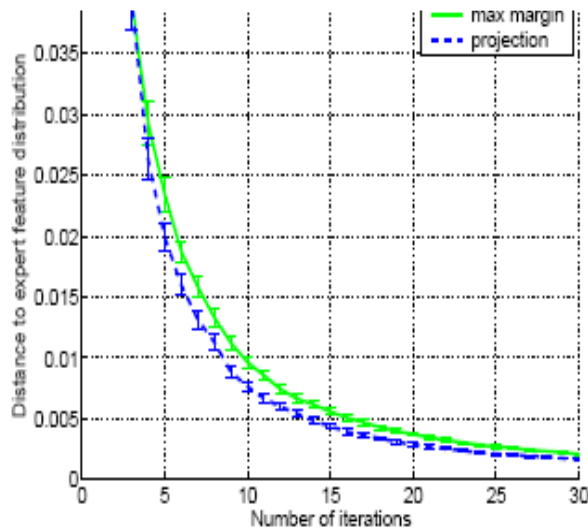
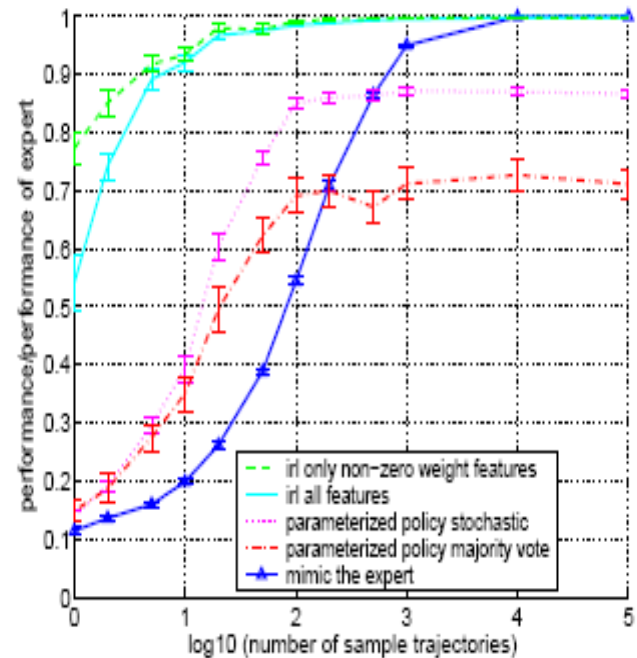


Figure 3. A comparison of the convergence speeds of the max-margin and projection versions of the algorithm on a 128x128 grid. Euclidean distance to the expert’s feature expectations is plotted as a function of the number of iterations. We rescaled the feature expectations by  $(1 - \gamma)$  such that they are in  $[0, 1]^k$ . The plot shows averages over 40 runs, with 1 s.e. errorbars.

Distance to expert vs. # Iterations



Performance vs. # Trajectories

# Car Driving Experiment

- No explicit reward function at all!
- Expert demonstrates proper policy via 2 min. of driving time on simulator (1200 data points).
- 5 different “driver types” tried.
- Features: which lane the car is in, distance to closest car in current lane.
- Algorithm run for 30 iterations, policy hand-picked.
- Movie Time! (Expert left, IRL right)

# Additional References and Pointers

- Policy gradient and policy search:
  - Kakade and Langford, ICML 2002
  - Bagnell, Kakade, Ng and Schneider, NIPS 2003
  - Bhatnagar, Sutton, Ghavamzadeh and Lee, NIPS 2007
  - Peters and Schaal, NN 2008
  
- Apprenticeship learning:
  - Ng and Russell, ICML 2000
  - Abbeel and Ng, ICML 2004
  - Ratliff, Bagnell and Zinkevich, ICML 2006
  - Daume, Langford and Marcu, MLJ 2008
  - Wingate and Singh, NIPS 2008
  
- Plus lots of other stuff by these guys, Rich Sutton, Andy Barto, Michael Kearns, and many others...