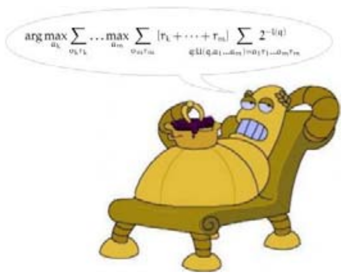
CSE 573: Artificial Intelligence

Reinforcement Learning II

Dan Weld



Many slides adapted from either Alan Fern, Dan Klein, Stuart Russell, Luke Zettlemoyer or Andrew Moore

Today's Outline

- Review Reinforcement Learning
- Review MDPs
  - New MDP Algorithm: Q-value iteration
- Review Q-learning
- Large MDPs
  - Linear function approximation
  - Policy gradient

Applications

- Robotic control
  - helicopter maneuvering, autonomous vehicles
  - Mars rover - path planning, oversubscription planning
  - elevator planning
- Game playing - backgammon, tetris, checkers
- Neuroscience
- Computational Finance, Sequential Auctions
- Assisting elderly in simple tasks
- Spoken dialog management
- Communication Networks – switching, routing, flow control
- War planning, evacuation planning

Demos

- http://inst.eecs.berkeley.edu/~ee128/fa11/videos.html

4

Agent Assets

Transition & Reward Model	Simulator (w/ teleport)	World
Value Iteration Policy Iteration	Monte Carlo Planning	Reinforcement Learning

5

Small vs. Huge MDPs

- First cover RL methods for small MDPs
  - Number of states and actions is reasonably small
    - Eg can represent policy as explicit table
  - These algorithms will inspire more advanced methods
- Later we will cover algorithms for huge MDPs
  - Function Approximation Methods
  - Policy Gradient Methods
  - Least-Squares Policy Iteration

6

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 optimal policy w/o explicitly learning the model
  - useful when model is difficult to represent and/or learn
- We will consider both types of approaches

Comparison

- Model-based approaches:

  Learn T + R

  $|S|^2|A| + |S||A|$ parameters (40,400)

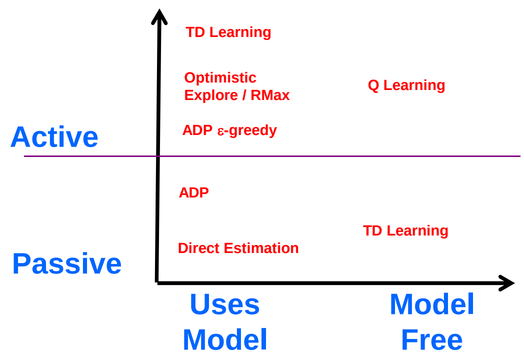
- Model-free approach:

  Learn Q

  $|S||A|$ parameters (400)

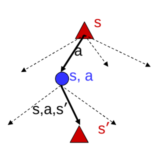
Supposing 100 states, 4 actions…

RL Dimensions



Recap: MDPs

- Markov decision processes:
  - States S
  - Actions A
  - Transitions T(s,a,s′) aka P(s′|s,a)
  - Rewards R(s,a,s′) (and discount γ)
  - Start state s_0 (or distribution P_0)
- Algorithms
  - Value Iteration
  - Q-value iteration
- Quantities:
  - Policy = map from states to actions
  - Utility = sum of discounted future rewards
  - Q-Value = expected utility from a q-state
    - Ie. from a state/action pair



Andrey Markov
(1856-1922)

Bellman Equations

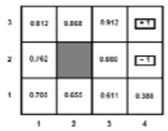
$$V^*(s) = \max_a Q^*(s,a)$$

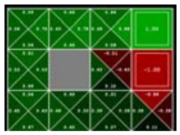
$$Q^*(a,s) = \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma V^*(s')\right]$$

Q-Value Iteration

- Regular Value iteration: find successive approx optimal values
  - Start with $V_0^*(s) = 0$
  - Given V_i^*, calculate the values for all states for depth i+1:

$$V_{i+1}(s) \leftarrow \max_a \ Q_{i+1}(s,a)$$

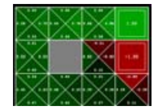




- Storing Q-values is more useful!
  - Start with $Q_0^*(s,a) = 0$
  - Given Q_i^*, calculate the q-values for all q-states for depth i+1:

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \ V_i(s') \right]$$

Q-Value Iteration



Initialize each q-state: $Q_0(s,a) = 0$

Repeat

For all q-states, s,a

Compute $Q_{i+1}(s,a)$ from Q_i by Bellman backup at s,a.

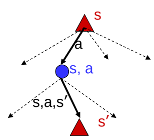
Until $\max_{s,a} |Q_{i+1}(s,a) - Q_i(s,a)| < \varepsilon$

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \ V_i(s') \right]$$

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a') \right]$$

Reinforcement Learning

- Markov decision processes:
  - States S
  - Actions A
  - Transitions T(s,a,s') aka P(s'|s,a)
  - Rewards R(s,a,s') (and discount γ)
  - Start state s_0 (or distribution P_0)



- Algorithms
  - Q-value iteration → Q-learning

Recap: Sampling Expectations

- Want to compute an expectation weighted by P(x):

$$E[f(x)] = \sum_x P(x) f(x)$$

- Model-based: estimate P(x) from samples, compute expectation

$$x_i \sim P(x)$$

$$\hat{P}(x) = \text{count}(x)/k \qquad E[f(x)] \approx \sum_x \hat{P}(x) f(x)$$

- Model-free: estimate expectation directly from samples

$$x_i \sim P(x) \qquad E[f(x)] \approx \frac{1}{k} \sum_i f(x_i)$$

- Why does this work? Because samples appear with the right frequencies!

Recap: Exp. Moving Average

- Exponential moving average
  - Makes recent samples more important

$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots}$$

  - Forgets about the past (distant past values were wrong anyway)
  - Easy to compute from the running average

$$\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Decreasing learning rate can give converging averages

Q-Learning Update

- Q-Learning = ***sample-based*** Q-value iteration

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right]$$

- How learn Q*(s,a) values?
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha) \left[sample \right]$$

Q-Learning Update

- Alternatively….

  difference = sample – Q(s, a)

  $Q(s,a) \leftarrow Q(s,a) + \alpha\,$[difference]

- How learn Q*(s,a) values?
  - Receive a sample (s,a,s′,r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

    $sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$
  - Incorporate the new estimate into a running average:

    $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$

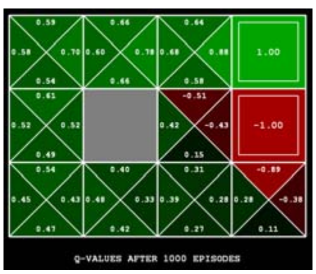
Exploration / Exploitation

- *ε greedy*
  - Every time step, flip a coin: with probability ε , act randomly
  - With probability 1- ε, act according to current policy
- Exploration function
  - Explore areas whose badness is not (yet) established
  - Takes a value estimate and a count, and returns an ***optimistic utility***, e.g. $f(u,n) = u + k/n$ (exact form not important)
  - Exploration policy $\pi(s')$=

$\max_{a'} Q_i(s',a')$ vs. $\max_{a'} f(Q_i(s',a'), N(s',a'))$

Q-Learning Final Solution

- Q-learning produces tables of q-values:

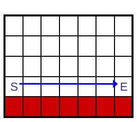


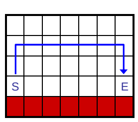
Q-Learning: ε Greedy

QuickTime™ and a
H.264 decompressor
are needed to see this picture.

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - … but not decrease it too quickly!
  - Not too sensitive to how you select actions (!)
- Neat property: off-policy learning
  - learn optimal policy without following it (some caveats)



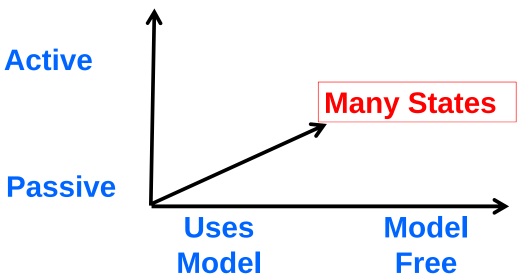


Q-Learning – Small Problem

- Doesn't work
- In realistic situations, we can't possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we need to ***generalize***:
  - Learn about a few states from experience
  - Generalize that experience to new, ***similar*** states

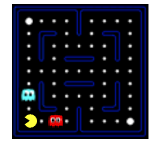
  (Fundamental idea in machine learning)

RL Dimensions

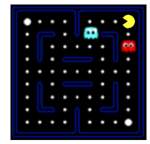


Example: Pacman

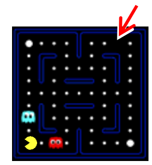
- Let's say we discover through experience that this state is bad:



- In naïve Q learning, we know nothing about related states and their Q values:



- Or even this third one!



Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - 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 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - 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 linear combination of a few weights:

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

- Advantage: our experience is summed up in a ***few*** powerful numbers

  $|S|^2|A|$? $|S||A|$?

- Disadvantage: states may share features but actually be very different in value!

Function Approximation

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

- Q-learning with linear q-functions:

$$\textit{transition} = (s, a, r, s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

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

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$ Approximate Q's

- 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

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

$a = \text{NORTH}$
$r = -500$

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

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

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

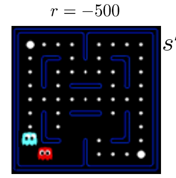
$$R(s,a,s') = -500$$

$$\textit{correction} = -501$$

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

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

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



Q-learning with Linear Approximators

1. Start with initial parameter values
2. Take action a according to an explore/exploit policy (should converge to greedy policy, i.e. GLIE) transitioning from s to s'
3. Perform TD update for each parameter

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s, a)$$

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

4. Goto 2

•Q-learning can diverge. Converges under some conditions.

31

Q-learning, no features, 50 learning trials:

QuickTime™ and a
GIF decompressor
are needed to see this picture.

Q-learning, no features, 1000 learning trials:

QuickTime™ and a
GIF decompressor
are needed to see this picture.

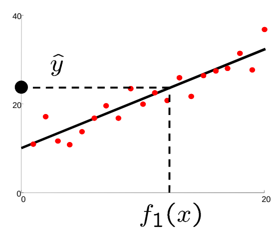
Q-learning, simple features, 50 learning trials:

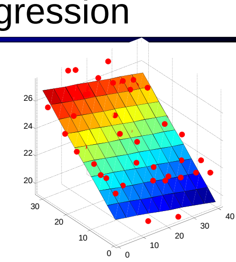
QuickTime™ and a
GIF decompressor
are needed to see this picture.

Why Does This Work?

35

Linear Regression



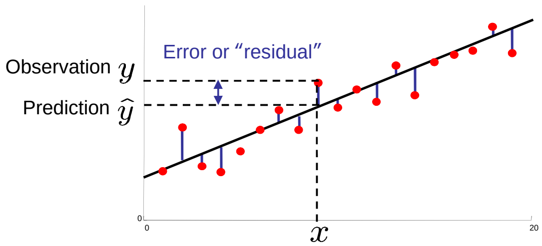


Prediction
$\hat{y} = w_0 + w_1 f_1(x)$

Prediction
$\hat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$

Ordinary Least Squares (OLS)

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i)\right)^2$$



Minimizing Error

Imagine we had only one point x with features f(x):

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\, \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

Approximate q update:

"target" "prediction"

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

Recap: Linear Function Approximation

- Define a set of state features f1(s), …, fn(s)
  - The features are used as our representation of states
  - States with similar feature values will be considered to be similar
- Often represent V*(s)* with linear approximation

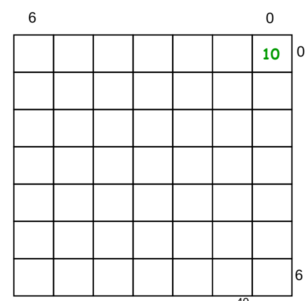
$$\hat{V}_\theta(s) = \theta_0 + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$$

- Approx. accuracy is fundamentally limited by the features
- Can we always define features for perfect approximation?
  - Yes! Assign each state an indicator feature. (I.e. i'th feature is 1 iff i'th state is present and θ_i represents value of i'th state)
  - No! This requires far too many features, gives no generalization.

39

Example

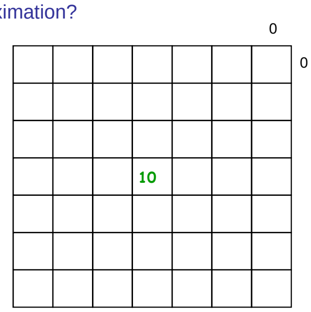
- Grid with no obstacles, deterministic actions U/D/L/R, no discounting, -1 reward everywhere except +10 at goal
- Features for state s=(x,y): f1(s)=x, f2(s)=y (just 2 features)
- V(s) = θ_0 + θ_1 x + θ_2 y
- Is there a good linear approximation?
  - Yes.
  - θ_0 =10, θ_1 = -1, θ_2 = -1
  - (note upper right is origin)
- V(s) = 10 - x - y subtracts Manhattan dist. from goal reward



40

But What If We Change Reward?

- V(s) = θ_0 + θ_1 x + θ_2 y
- Is there a good linear approximation?
  - No.

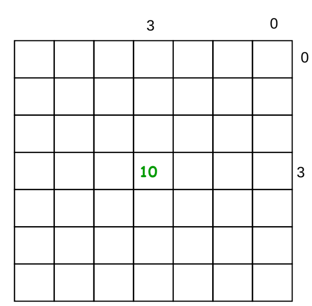


41

But What If…

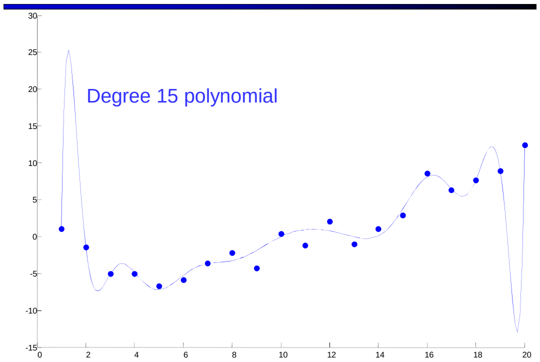
- V(s) = θ_0 + θ_1 x + θ_2 y + θ_3 z

- Include new feature z
  - z= |3-x| + |3-y|
  - z is dist. to goal location
- Does this allow a good linear approx?
  - θ_0 =10, θ_1 = θ_2 = 0, θ_0 = -1



42

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
  - Learn optimal "value" of being in a state, or taking an action from state.
- Value functions can often be much more complex to represent than the corresponding policy
  - Do we really care about knowing Q(s,left) = 0.3554, Q(s,right) = 0.533
  - Or just that "right is better than left in state s"
- Motivates searching directly in a parameterized policy space
  - Bypass learning value function and "directly" optimize the value of a policy

44

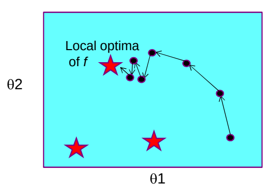
Aside: Gradient Ascent

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

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

where

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



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

45

RL via Policy Gradient Ascent

- The policy gradient approach has the following schema:
  1. Select a space of parameterized policies
  2. Compute the gradient of the value of current policy wrt parameters
  3. Move parameters in the direction of the gradient
  4. Repeat these steps until we reach a local maxima
  5. Possibly also add in tricks for dealing with bad local maxima (e.g. random restarts)
- So we must answer the following questions:
  - How should we represent parameterized policies?
  - How can we compute the gradient?

46

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.

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

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

47

Policy Gradient Ascent

- 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 $\rho(\theta)$ even if we have an exact model of the system.
- **Key idea:** estimate the gradient based on experience

48