CSE 473: Artificial Intelligence
Spring 2012

Reinforcement Learning

Dan Weld

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

Today's Outline

- Reinforcement Learning
  - Passive Learning
  - TD Updates
  - Q-value iteration
  - Q-learning
  - Linear function approximation

Reinforcement Learning

- Still have an MDP
  - Still looking for policy π
- New twist: don't know **T** or **R**
  - Don't know what actions do
  - Nor which states are good!
- Must actually try out actions to learn

Formalizing the reinforcement learning problem

- Given a set of states **S** and actions **A**
- Interact with world at each time step *t*:
  - world gives state $\mathbf{s}_t$ and reward r_t
  - you give next action $\mathbf{a}_t$
- **Goal**: (quickly) learn policy that (approximately) maximizes long-term expected discounted reward

The "Credit Assignment" Problem

I'm in state 43,	reward = 0,	action = 2
" " " 39,	" = 0,	" = 4
" " " 22,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 13,	" = 0,	" = 2
" " " 54,	" = 0,	" = 2
" " " 26,	" = **100**,	

Yippee! I got to a state with a big reward!

But which of my actions along the way actually helped me get there??

This is the Credit Assignment problem.

Exploration-Exploitation tradeoff

- You have visited part of the state space and found a reward of 100
  - is this the best you can hope for???
- **Exploitation**: should I stick with what I know and find a good policy w.r.t. this knowledge?
  - at risk of missing out on a better reward somewhere
- **Exploration**: should I look for states w/ more reward?
  - at risk of wasting time & getting some negative reward

6

Two main reinforcement learning approaches

- Model-based approaches:
  - explore environment & learn model, T=P(**s'**|**s**,**a**) and R(**s**,**a**), (almost) everywhere
  - use model to plan policy, MDP-style
  - approach leads to strongest theoretical results
  - often works well when state-space is manageable
- Model-free approach:
  - don't learn a model; learn value function or policy directly
  - weaker theoretical results
  - often works better when state space is large

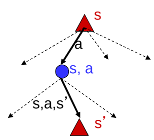
Passive *vs.* Active learning

- Passive learning
  - The agent has a ***fixed policy***
  - Tries to ***learn utilities of states*** by observing world go by
  - Analogous to policy evaluation
    - Often serves as a component of active learning algorithms
    - Often inspires active learning algorithms
- Active learning
  - Agent tries to ***find a good policy*** by acting in the world
  - Analogous to solving the underlying MDP
    - But without first being given the MDP model

8

Recap: MDPs

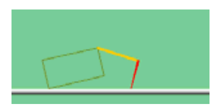
- Markov decision processes:
  - States S
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount ©)
  - Start state s_0 (or distribution P_0)



- Quantities:
  - Policy = map from states to actions
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state
  - Q-Values = expected future utility from a q-state
    - Ie. A state/action pair

Reinforcement Learning

- Reinforcement learning:
  - Still have an MDP:
    - A set of states s ◘ S
    - A set of actions (per state) A
    - A model T(s,a,s')
    - A reward function R(s,a,s')
  - Still looking for a policy □(s)
  - New twist: don't know T or R
    - I.e. don't know which states are good or what the actions do
    - Must actually try actions and states out to learn



What is it doing?

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

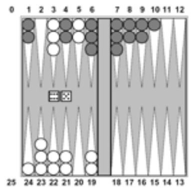
Example: Animal Learning

- RL studied experimentally for more than 60 years in psychology
  - Rewards: food, pain, hunger, drugs, etc.
  - Mechanisms and sophistication debated

- Example: foraging
  - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
  - Bees have a direct neural connection from nectar intake measurement to motor planning area

Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to V(s) using a neural network
- Combined with depth 3 search, one of the top 3 players in the world



- You could imagine training Pacman this way…
- … but it’s tricky! (It’s also P3)

Extreme Driving

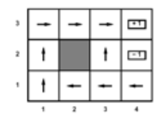
http://www.youtube.com/watch?v=gzI54rm9m1Q

14

Other 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

Passive Learning



- Simplified task
  - You don’t know the transitions T(s,a,s’)
  - You don’t know the rewards R(s,a,s’)
  - You are given a policy π(s)
  - Goal: learn the state values (and maybe the model)
  - I.e., policy evaluation
- In this case:
  - Learner “along for the ride”
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We’ll get to the active case soon
  - This is NOT offline planning!

Detour: 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

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

- Model-free: estimate expectation directly from samples

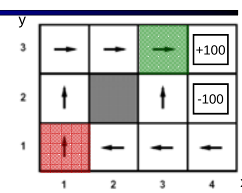
$$x_i \sim P(x) \qquad E[f(x)] \approx \frac{1}{k} \sum_i f(x_i)$$

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

Simple Case: Direct Estimation

- Episodes:

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



γ= 1, R = -1

V(1,1) ~ (92 + -106) / 2 = -7

V(3,3) ~ (99 + 97 + -102) / 3 = 31.3

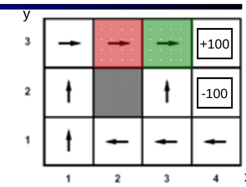
Model-Based Learning

- Idea:
  - Learn the model empirically (rather than values)
  - Solve the MDP as if the learned model were correct
  - Better than direct estimation?
- Empirical model learning
  - Simplest case:
    - Count outcomes for each s,a
    - Normalize to give estimate of T(s,a,s')
    - Discover R(s,a,s') the first time we experience (s,a,s')
  - More complex learners are possible (e.g. if we know that all squares have related action outcomes, e.g. "stationary noise")

Example: Model-Based Learning

- Episodes:

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



© = 1

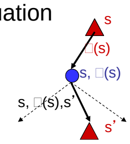
T(<3,3>, right, <4,3>) = 1 / 3

T(<2,3>, right, <3,3>) = 2 / 2

Towards Better Model-free Learning

Review: Model-Based Policy Evaluation

- Simplified Bellman updates to calculate V for a ***fixed policy***:
  - New V is expected one-step-look-ahead using current V
  - Unfortunately, need T and R



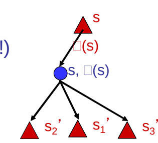
$$V_0^\pi(s) = 0$$

$$V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_i^\pi(s')]$$

Sample Avg to Replace Expectation?

$$V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_i^\pi(s')]$$

- Who needs T and R? Approximate the expectation with samples (drawn from T!)



$$sample_1 = R(s,\pi(s),s_1') + \gamma V_i^\pi(s_1')$$
$$sample_2 = R(s,\pi(s),s_2') + \gamma V_i^\pi(s_2')$$
$$\dots$$
$$sample_k = R(s,\pi(s),s_k') + \gamma V_i^\pi(s_k')$$
$$V_{i+1}^\pi(s) \leftarrow \frac{1}{k}\sum_i sample_i$$

Detour: Exp. Moving Average

- Exponential moving average
  - Makes recent samples more important

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

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

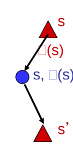
$$\bar{x}_n = (1-\alpha)\cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Decreasing learning rate can give converging averages

Model-Free Learning

$$V^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

- Big idea: why bother learning T?
  - Update V each time we experience a transition
- "Temporal difference learning" (TD)
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!



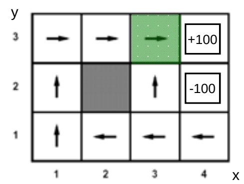
$$sample = R(s,\pi(s),s') + \gamma V^\pi(s')$$
$$V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + (\alpha)sample$$
$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$$

TD Policy Evaluation

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

(1,1) up -1
(1,2) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(3,3) right -1
(4,3) exit +100
(done)

(1,1) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(4,2) exit -100
(done)



Updates for V(<3,3>):

V(<3,3>) = 0.5*0 + 0.5*[-1 + 1*0] = -0.5

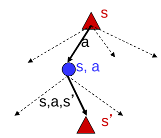
V(<3,3>) = 0.5*-0.5 + 0.5*[-1+1*100] = 49.475

V(<3,3>) = 0.5*49.475 + 0.5*[-1 + 1*-0.75]

Take $\gamma = 1$, $\alpha = 0.5$, $V_0(<4,3>)=100$, $V_0(<4,2>)=-100$, $V_0 = 0$ otherwise

Problems with TD Value Learning

- TD value leaning is model-free for policy evaluation (passive learning)
- However, if we want to turn our value estimates into a policy, we're sunk:



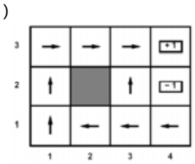
$$\pi(s) = \arg\max_a Q^*(s,a)$$

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

- Idea: learn Q-values directly
- Makes action selection model-free too!

Active Learning

- Full reinforcement learning
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You can choose any actions you like
  - Goal: learn the optimal policy
  - … what value iteration did!



- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Detour: Q-Value Iteration

- 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:

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

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

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

Q-Learning Update

- Q-Learning: sample-based Q-value iteration

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

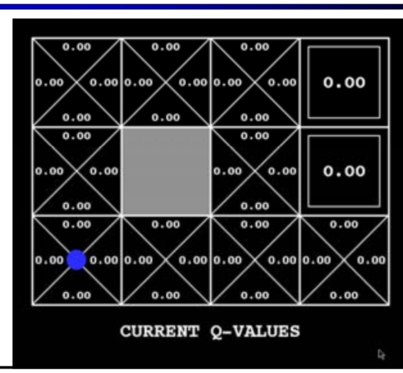
- Learn Q*(s,a) values
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

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

Q-Learning: Fixed Policy



Exploration / Exploitation

- Several schemes for action selection
  - Simplest: random actions (*ε* ***greedy***)
    - Every time step, flip a coin
    - With probability ε , act randomly
    - With probability 1- ε, act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: ***exploration functions***

Q-Learning: ε Greedy

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

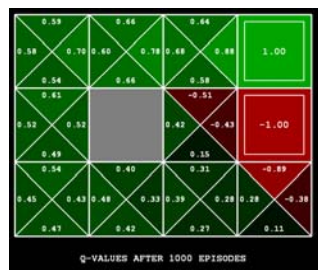
Exploration Functions

- When to explore
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established
- Exploration function
  - 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 π(s')=

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

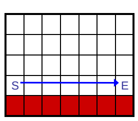
Q-Learning Final Solution

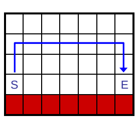
- Q-learning produces tables of q-values:



Q-Learning 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

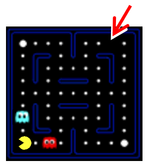
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we' ll see it over and over again

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:

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

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

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

Todo

- Add 446

40

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:

$$transition\ = (s,a,r,s')$$

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

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

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

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

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

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

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

$$correction = -501$$

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

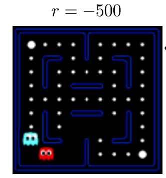
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

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

s

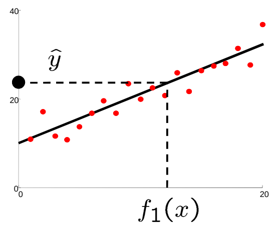
$a = \text{NORTH}$

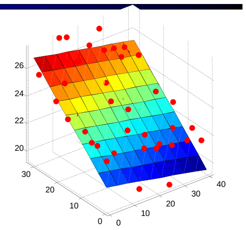
$r = -500$



s'

Linear Regression



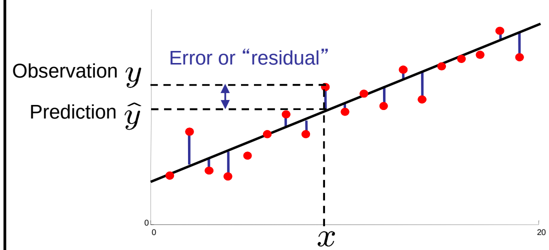


Prediction
$\hat{y} = w_0 + w_1 f_1(x)$

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):

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

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

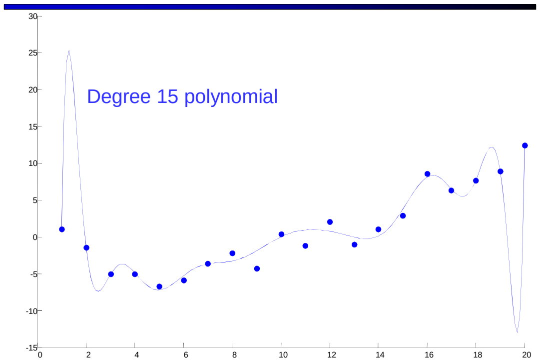
$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

Approximate q update:

"target" "prediction"

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

Overfitting



Which Algorithm?

Q-learning, no features, 50 learning trials:

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

Which Algorithm?

Q-learning, no features, 1000 learning trials:

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

Which Algorithm?

Q-learning, simple features, 50 learning trials:

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

Policy Search*

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

Policy Search*

- Problem: often the feature-based policies that work well aren't the ones that approximate V / Q best
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - We'll see this distinction between modeling and prediction again later in the course
- Solution: learn the policy that maximizes rewards rather than the value that predicts rewards
- This is the idea behind policy search, such as what controlled the upside-down helicopter

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

Policy Search*

- Advanced policy search:
  - Write a stochastic (soft) policy:

$$\pi_w(s) \propto e^{\sum_i w_i f_i(s,a)}$$

  - Turns out you can efficiently approximate the derivative of the returns with respect to the parameters w (details in the book, optional material)
  - Take uphill steps, recalculate derivatives, etc.