

Reinforcement Learning (Lecture 2)

Ron Parr
Duke University

RL Highlights

- Everybody likes to learn from experience
- Use ML techniques to generalize from *relatively small amounts* of experience
- Some notable successes:
 - Backgammon
 - Flying a helicopter upside down
- Sutton's seminal RL paper is 40th most cited paper in computer science (Citeseer 9/04)



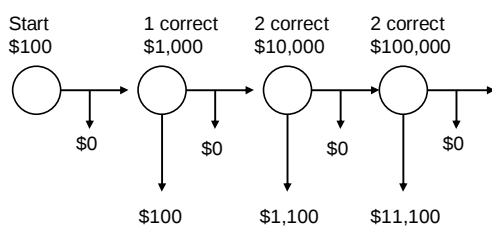
Comparison w/Other Kinds of Learning

- Learning often viewed as:
 - Classification (supervised), or
 - Model learning (unsupervised)
- RL is between these (delayed signal)
- What the last thing that happens before an accident? 🌀

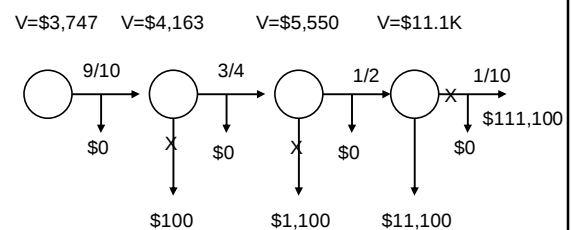
Overview

- Review of value determination
- Motivation for RL
- Algorithms for RL
 - Overview
 - TD
 - Q-learning
 - Approximation

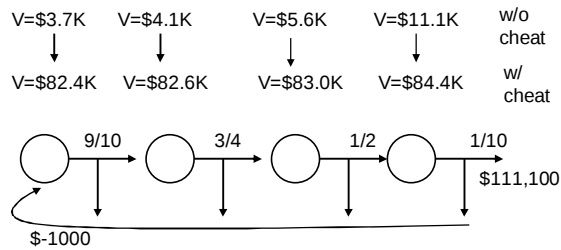
Recall Our Game Show



Optimal Policy w/o Cheating



Cheat until you win policy



Solving for Values

$$\mathbf{V} = \gamma \mathbf{P}_{\pi} \mathbf{V} + \mathbf{R}$$

For moderate numbers of states we can solve this system exactly:

$$\mathbf{V} = (\mathbf{I} - \gamma \mathbf{P}_{\pi})^{-1} \mathbf{R}$$

Guaranteed invertible because $\mathbf{P}_{\pi}$ has spectral radius < 1

Iteratively Solving for Values

$$\mathbf{V} = \gamma \mathbf{P}_{\pi} \mathbf{V} + \mathbf{R}$$

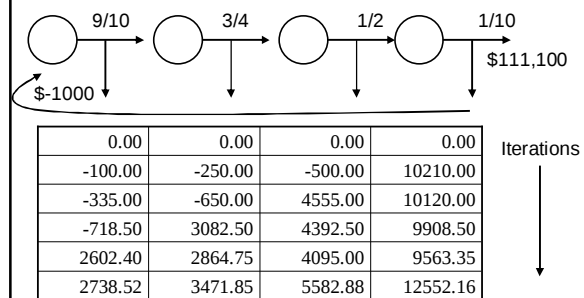
For larger numbers of states we can solve this system indirectly:

$$\mathbf{V}^{i+1} = \gamma \mathbf{P}_{\pi} \mathbf{V}^i + \mathbf{R}$$

Guaranteed convergent because $\mathbf{P}_{\pi}$ has spectral radius < 1 for $\gamma < 1$

Convergence not guaranteed for $\gamma = 1$

Iterative Policy Evaluation



Iterations Contd.

i=0	0.00	0.00	0.00	0.00
i=1	-100.00	-250.00	-500.00	10210.00
i=2	-335.00	-650.00	4555.00	10120.00
i=3	-718.50	3082.50	4392.50	9908.50
i=4	2602.40	2864.75	4095.00	9563.35
i=5	2738.52	3471.85	5582.88	12552.16
i=20	15697.49	16688.07	18396.47	23621.43
i=100	56740.99	57190.86	58074.31	60999.20
i=200	74658.96	74872.93	75399.39	77318.76
i=1000	82469.80	82580.93	82951.31	84432.82
i=10000	82470.37	82581.48	82951.85	84433.33

Note: Slow convergence b/c $\gamma = 1$

Overview

- Review of value determination
- Motivation for RL
- Algorithms for RL
 - Overview
 - TD
 - Q-learning
 - Approximation

Why We Need RL

- Where do we get transition probabilities?
- How do we store them?
 - Big problems have big models
 - Model size is quadratic in state space size
- Where do we get the reward function?

RL Framework

- Learn by “trial and error”
- No assumptions about model
- No assumptions about reward function
- Assumes:
 - True state is known at all times
 - Immediate reward is known
 - Discount is known

RL Schema

- Act
- Perceive results
- Update something
- Repeat



RL for Our Game Show

- Problem: We don't prob of answering correctly
- Solution:
 - Buy the home version of the game
 - Practice on the home game to refine our strategy
 - Deploy strategy when we play the real game

Model Learning Approach

- Learn model, solve = Certainty Equivalence
- How to learn a model:
 - Take action a in state s , observe s'
 - Take action a in state s , n times
 - Observe s' m times
 - $P(s'|s,a) = m/n$
 - Fill in transition matrix for each action
 - Compute avg. reward for each state
- Solve learned model as an MDP

Limitations of Model Learning

- Partitions learning, solution into two phases
- Model may be large (hard to visit every state lots of times)
 - Note: Can't completely get around this problem...
- Model storage is expensive
- Model manipulation is expensive

Overview

- Review of value determination
- Motivation for RL
- Algorithms for RL
 - Overview
 - TD
 - Q-learning
 - Approximation

Temporal Difference Learning

- One of the first RL algorithms
- Learn the value of a *fixed* policy (no optimization; just prediction)
- Recall iterative value determination:

$$V^{i+1}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^i(s')$$

Problem: We don't know this.

First Idea: Monte Carlo Sampling

- Assume that we have a black box:



- Count the number of times we see each s'
 - Estimate $P(s'|s)$ for each s'
 - Essentially learns a mini-model for state s
 - Can think of as numerical integration
- Problem: The world doesn't work this way

Next Idea

- Remember Value Determination:

$$V^{i+1}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^i(s')$$

- Compute an update *as if the observed s' and r were the only possible outcomes*:

$$V^{temp}(s) = r + \gamma V^i(s')$$

- Make a small update in this direction:

$$V^{i+1}(s) = (1 - \alpha) V^i(s) + \alpha V^{temp}(s)$$

$$0 < \alpha \leq 1$$

Idea: Value Function Soup

Suppose: $\alpha = 0.1$

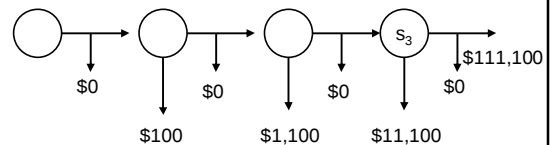
- Upon observing s' :
- Discard 10% of soup
 - Refill with $V^{temp}(s)$
 - Stir
 - Repeat



One vat for each state

$$V^{i+1}(s) = (1 - \alpha) V^i(s) + \alpha V^{temp}(s)$$

Example: Home Version of Game



Suppose we guess: $V(s_3) = 15K$
We play and get the question wrong

$$V^{temp} = 0$$

$$V(s_3) = (1 - \alpha) 15K + \alpha 0$$

Convergence?

- Why doesn't this oscillate?
 - e.g. consider some low probability s' with a very high (or low) reward value



- This could still cause a big jump in $V(s)$

Convergence Intuitions

- Need heavy machinery from stochastic process theory to prove convergence
- Main ideas:
 - Iterative value determination converges
 - Updates approximate value determination
 - Samples approximate expectation

$$V^{i+1}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^i(s')$$

Ensuring Convergence

- Rewards have bounded variance
- $0 \leq \gamma < 1$
- Every state visited infinitely often
- Learning rate decays so that:
 - $\sum_t \alpha_t(s) = \infty$
 - $\sum_t \alpha_t^2(s) < \infty$

These conditions are jointly *sufficient* to ensure convergence in the limit with probability 1.

How Strong is This?

- Bounded variance of rewards: easy
- Discount: standard
- Visiting every state infinitely often: Hmmm...
- Learning rate: Often leads to slow learning
- Convergence in the limit: Weak
 - Hard to say anything stronger w/o knowing the mixing rate of the process
 - Mixing rate can be low; hard to know a priori
- Convergence w.p. 1: Not a problem.

Using TD for Control

- Recall value iteration:

$$V^{i+1}(s) = \max_a R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^i(s')$$

- Why not pick the maximizing a and then do:

$$V^{i+1}(s) = (1 - \alpha) V^i(s) + \alpha V^{temp}(s')$$

- s' is the observed next state after taking action a

Problems

- Pick the best action w/o model?
- Must visit every state infinitely often
 - What if a good policy doesn't do this?
- Learning is done “on policy”
 - Taking random actions to make sure that all states are visited will cause problems

Q-Learning Overview

- Want to maintain good properties of TD
- Learns good policies and optimal value function, not just the value of a fixed policy
- Simple modification to TD that learns the optimal policy regardless of how you act! (mostly)

Q-learning

- Recall value iteration:

$$V^{i+1}(s) = \max_a R(s, a) + \gamma \sum_{s'} P(s' | s, a) V^i(s')$$

- Can split this into two functions:

$$Q^{i+1}(s, a) = R(s, a) + \gamma \sum_{s'} P(s' | s, a) V^i(s')$$

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

Q-learning

- Store Q values instead of a value function
- Makes selection of best action easy
- Update rule:

$$Q^{temp}(s, a) = r + \gamma \max_{a'} Q^i(s', a')$$

$$Q^{i+1}(s, a) = (1 - \alpha) Q^i(s, a) + \alpha Q^{temp}(s, a)$$

Q-learning Properties

- Converges under same conditions as TD
- Still must visit every state infinitely often
- Separates policy you are currently following from value function learning:

$$Q^{temp}(s, a) = r + \gamma \max_{a'} Q^i(s', a')$$

$$Q^{i+1}(s, a) = (1 - \alpha) Q^i(s, a) + \alpha Q^{temp}(s, a)$$

Value Function Representation

- Fundamental problem remains unsolved:
 - TD/Q learning solves model-learning problem, but
 - Large models still have large value functions
 - Too expensive to store these functions
 - Impossible to visit every state in large models
- Function approximation
 - Use machine learning methods to generalize
 - Avoid the need to visit every state

Function Approximation

- General problem: Learn function $f(s)$
 - Linear regression
 - Perceptron
 - Neural networks
- Idea: Approximate $f(s)$ with $g(s, w)$
 - g is some easily computable function of s and w
 - Try to find w that minimizes the error in g

Linear Regression

- Define a set of basis functions (vectors)

$$h_1(s), h_2(s), \dots, h_k(s)$$

- Approximate f with a weighted combination of these

$$g(s) = \sum_{j=1}^k w_j h_j(s)$$

- Example: Space of quadratic functions:

$$h_1(s) = 1, h_2(s) = s, h_3(s) = s^2$$

- Orthogonal projection minimizes SSE

Updates with Approximation

- Recall regular TD update:

$$V^{i+1}(s) = (1 - \alpha)V^i(s) + \alpha V^{temp}(s)$$

- With function approximation:

$$V(s) \approx V(s, \theta)$$

Vector
operations

- Update:

$$\theta^{i+1} = (1 - \alpha)\theta^i + \alpha V^{temp}(s) \nabla_{\theta} V(s, \theta)$$

For linear value functions

- Gradient is trivial:

$$V(s, \theta) = \sum_{j=1}^k \theta_j h_j(s)$$

$$\nabla_{\theta_j} V(s, \theta) = h_j(s)$$

Individual
components

- Update is trivial:

$$\theta_j^{i+1} = (1 - \alpha)\theta_j^i + \alpha V^{temp}(s) h_j(s)$$

Properties of approximate RL

- Table-updates are a special case
- Can be combined with Q-learning
- Convergence not guaranteed
 - Ordinary neural nets converge to local opt
 - NN + RL convergence not guaranteed
 - Chasing a moving target
 - Errors can compound
- Success requires very well chosen features

Other Approaches

- TD, Q-learning approximate value iteration
- Typically use parameterized V
- Can also approximate policy iteration
 - Parameterized space of policies
 - Estimate values from samples
 - Update policy parameters to improve performance

How'd They Do That???

- Backgammon (Tesauro)
 - Neural network value function approximation
 - TD sufficient (known model)
 - Carefully selected inputs to neural network
 - About 1 million games played against self
- Helicopter (Ng et al.)
 - Approximate policy iteration
 - Constrained policy space
 - Trained on a simulator

Swept under the rug...

- Difficulty of finding good features
- Partial observability
- Exploration vs. Exploitation

Conclusions

- Reinforcement learning solves an MDP
- Converges for exact value function representation
- Can be combined with approximation methods
- Good results require good features