Reinforcement Learning

Ron Parr
CPS 570

RL Highlights

- Everybody likes to learn from experience
- Use ML techniques to generalize from *relatively small amounts* of experience
- Some notable successes:
  - Playing Atari games
  - Aerobatic helicopter maneuvers
  - Go

From Andrew Ng's home page

- Sutton's seminal RL paper is 178th most cited ref. in computer science (Citeseerx 11/16); Sutton & Barto RL Book is the 7th most cited

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

Solving for Values

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

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

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

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

Iteratively Solving for Values

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

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

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

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

Convergence not guaranteed for γ=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 for Our Game Show

- Problem: Don’t know 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
- 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
  - TD
  - Q-learning
  - Approximation

Temporal Differences

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

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

Problem: We don't know this.

Temporal Difference Learning

- Remember Value Determination:

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

- 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:

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

$$0 < \alpha \leq 1$$

Note: we have dropped the π subscripts

Idea: Value Function Soup

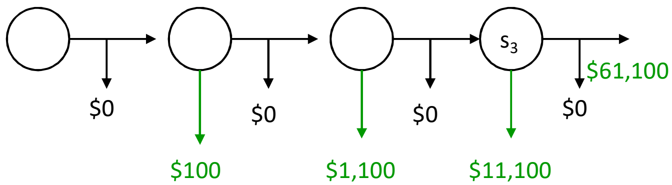
Suppose: $\alpha = 0.1$

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 our current estimate: $V(s_3)$=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
  - TD 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
- $0 \le \gamma < 1$
- Every state visited infinitely often
- Learning rate decays so that:
  - $\sum_i^\infty \alpha_i(s) = \infty$
  - $\sum_i^\infty \alpha_i^2(s) < \infty$

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:

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

- 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:

$$Q^{i+1}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^i(s')$$
$$V^{i+1}(s) = \max_a Q^{t+1}(s,a)$$

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:

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

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

Note: If there is only one action possible in each state, then Q-learning and TD-learning are identical

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
  - Neural networks
  - State aggregation (violates Markov property)
- Idea: Approximate f(s) with $g(s,\theta)$
  - g is some easily computable function of s and θ
  - Try to find θ that minimizes the error in g

Linear Regression review

- Define a set of basis functions (vectors)

$$\phi_1(s),\phi_2(s)\ldots\phi_k(s)$$

- Approximate f with a weighted combination of these

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

- Example: Space of quadratic functions:

$$\phi_1(s)=1,\phi_2(s)=s,\phi_3(s)=s^2$$

- 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:

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

- Update:

Vector operations

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

Neural networks are a special case of this.

For linear value functions

- Gradient is trivial:

$$V(s,w)=\sum_{j=1}^{k} w_j\phi_j(s)$$

$$\nabla_{w_j} V(s,w)=\phi_j(s)$$

- Update is trivial:

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

Properties of approximate RL

- Exact case (tabular representation) = special case
- Can be combined with Q-learning
- Convergence not guaranteed
  - Policy evaluation with linear function approximation converges if samples are drawn "on policy"
  - In general, convergence is not guaranteed
    - Chasing a moving target
    - Errors can compound
- Success has often required very well chosen features, but this may be changing (deep RL)

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