

Reinforcement Learning

CPS 271
Ron Parr

Limitations of the MDP Framework

- Assumes that transition probabilities known
 - How do we discover these?
 - How do we store them?
 - Big problems have big models
 - Model size is quadratic in state space size
- Assumes reward function is known

Reinforcement Learning

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

RL Schema

- Act



- Perceive next state



- Update value function



- Repeat

Highlights

- Trial and error methods can be mapped to the dynamic programming/MDP framework
- Easy to implement
 - Easy to code
 - Low CPU utilization
- Incremental approach converges in the limit to the same answer as traditional methods
- Can be combined with function approximation – with some caveats
- Has been used successfully in many domains

Model Learning Approach

- Model learning + solving = 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

Temporal Difference Learning

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

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

↑
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:

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

- 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:

$$V^{i+1}(s) = (1 - \alpha) V^i(s) + \alpha V^{temp}(s) \quad 0 < \alpha \leq 1$$

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 $V(s)$

Ensuring Convergence

- Rewards have bounded variance
- $0 \leq \gamma < 1$
- Every state visited infinitely often
- Learning rate decays so that:

$$-\sum_i \alpha_i(s) = \infty$$

$$-\sum_i \alpha_i^2(s) < \infty$$

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.

Value Function Representation

- Fundamental problem remains unsolved:
 - TD 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

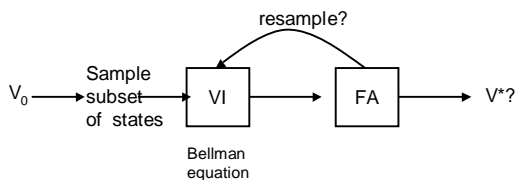
- General problem: Learn function $f(s)$
 - Perceptron
 - Linear regression
 - Neural networks
- Idea: Approximate $f(s)$ with $g(s,w)$
 - g is some easily computable function of s and w
 - Try to find w that minimizes the error in g

Implementing VFA

- Can't represent V as a big vector
- Use (parametric) function approximator
 - Neural network
 - Linear regression (least squares)
 - Nearest neighbor (with interpolation)
- (Typically) sample a subset of the the states
- Use function approximation to “generalize”

Basic Value Function Approximation

Idea: Consider restricted class of value functions
Hope to generalize from subset of states to all



Alternate value iteration with supervised learning

VFA Outline

1. Initialize $V_0(s, \mathbf{w}_0)$, $n=1$
2. Select some $s_0 \dots s_i$
3. For each s_i $\tilde{V}(s_i) = R(s_i) + \gamma \max_a \sum_{s'} P(s'|s, a) V(s', \mathbf{w})$
4. Compute $V_n(s, \mathbf{w}_n)$ by training w on \tilde{v}
5. $n := n+1$
6. Unless $V_{n+1} - V_n < \epsilon$ goto 2

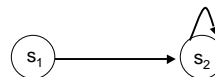
If supervised learning error is “small”,
then V_{final} “close” to V^* .

Key Questions

- Stability
 - Do values converge to some point or region?
- Quality bounds (assuming convergence)
 - How close are we to optimal?

Stability Problem

Problem: Most VFA methods are unstable



No rewards, $\gamma = 0.9$: $V^* = 0$

Example: Bertsekas & Tsitsiklis 1996

Least Squares Approximation

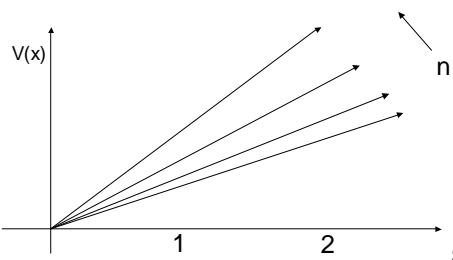
Restrict V to linear functions:



Find θ s.t. $V(s_1) = \theta$, $V(s_2) = 2\theta$

Counterintuitive Result: If we do a least squares fit of θ
 $\theta^{t+1} = 1.08 \theta^t$

Unbounded Growth of V



What Went Wrong?

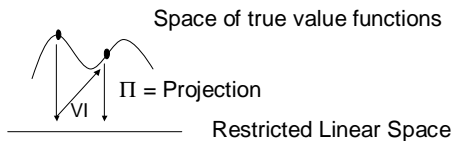
- VI reduces error in maximum norm
- Least squares (= projection) non-expansive in L_2
- May increase maximum norm distance
- Grows max norm error at faster rate than VI shrinks it
- And we didn't even use sampling!
- Seems like bad news for neural networks...
- Success depends on
 - sampling distribution
 - pairing approximator and problem
- Good feature engineering can succeed (TD-Gammon)

Types of Success with VFA

- Empirical
 - Use domain knowledge to choose family of functions carefully
 - Tricky...
- Theoretical
 - Establish a posteriori bounds in case of convergence (weak)
 - Pick special function approximation methods that permit a priori bounds

Success Stories - Linear TD

- [Tsitsiklis & Van Roy 96, Bratdke & Barto 96]
- Start with a set of basis functions
- Restrict V to linear space spanned by bases
- Do *weighted* projection



N.B. linear is still expressive due to basis functions

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 basis functions

$$g(s) = \sum_{i=1}^k w_i h_i(s)$$

- Example: Space of quadratic functions:

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

- Orthogonal projection minimizes sum of squared errors

Formal Properties

- Use to evaluate policies *only*
- Converges w.p. 1
- Error measured w.r.t. stationary distribution
- Frequently visited states have low error
- Infrequent states can have high error

$$\|V^* - \hat{V}\|_{\rho} \leq \frac{\|V^* - \Pi_{\rho} V^*\|_{\rho}}{\sqrt{1 - k^2}}$$

Linear Methods

- Applications
 - Inventory control: Van Roy et al.
 - Packet routing: Marbach et al.
 - Used by Morgan Stanley to value options
- No guarantees when combined with policy improvement
 - Can produce *bad* policies for trivial problems [Koller & Parr 99]
 - Modified for better PI: LSPI [Lagoudakis & Parr 01]
- Can be done symbolically [Koller & Parr 00]
- Issues
 - Selection of basis functions
 - Mixing rate of process - affects κ , speed

Success Story: Averagers [Gordon 95, and others...]

- Pick set, $Y = y_1, \dots, y_l$ of representative states
- Perform VI on Y
- For x not in Y , $V(x) = \sum_i \beta_i J(y_i)$

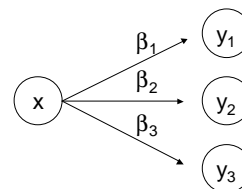
$$0 \leq \beta_i \leq 1$$

$$\sum_i \beta_i = 1$$

- Averagers are non expansions in max norm
- Converge to within $1/(1-\gamma)$ factor of "best"

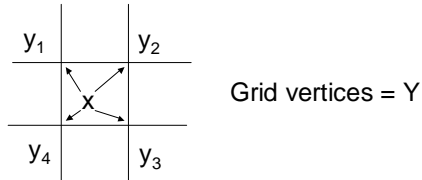
Interpretation of Averagers

$$V(x) = \sum_i \beta_i J(y_i)$$



Interpretation of Averagers II

Averagers Interpolate:



Neural Networks

- s = input into neural network
- w = weights of neural network
- $g(s,w)$ = output of network
- Try to minimize

$$E = \sum_s (f(s) - g(s, w))^2$$

- Compute gradient of error wrt weights

$$\frac{\partial E}{\partial w}$$

- Adjust weights in direction that minimizes error

Combining NNs with TD

- Recall TD:

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

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

- Compute error function:

$$E = (\hat{V}^i(s, w) - \hat{V}^{temp}(s, w))^2$$

- Update:

$$w^{i+1} = w^i - \alpha \frac{\partial E}{\partial w}$$

$$= w^i + 2\alpha [\hat{V}^{temp}(s, w) - \hat{V}(s, w)] \frac{\partial \hat{V}(s, w)}{\partial w}$$

Is this right???

Gradient-based Updates

$$w^{i+1} = w^i - \alpha \frac{\partial E}{\partial w}$$

$$= w^i + 2\alpha [\hat{V}^{temp}(s, w) - \hat{V}(s, w)] \frac{\partial \hat{V}(s, w)}{\partial w}$$

- Constant factor absorbed into learning rate
- Table-updates are a special case
- Perceptron, linear regression are special cases
- Converges for linear architectures (Tsitsiklis & Van Roy)

Using TD for Control

- Recall value iteration:

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

- 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

- How do you 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
- Linear function approximation doesn't provably converge for optimization (but is still used successfully in many cases!)

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:

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

- Can split this into two functions:

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

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

Q-learning

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

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

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

Q-learning Properties

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

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

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

Q-learning Properties

- Can't prove convergence with function approximation
- Introduces exploration vs. exploitation dilemma