Reinforcement Learning

Ron Parr
CPS 271

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 88th most cited ref. in computer science (Citeseerx 10/09); Sutton & Barto RL Book is the 14th 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
Recall Our Game Show

Start
$100

1 correct
$1,000

2 correct
$10,000

2 correct
$100,000

Optimal Policy w/o Cheating

V=$3,750  V=$4,166  V=$5,555  V=$11.1k

9/10
$0

3/4
$0

1/2
$0

1/10
$0

$100

$1,100

$11,100

$111,100
Cheat until you win policy

V=$3,749  V=$4,166  V=$5,555  V=$11.11K  w/o cheat
\[
\begin{align*}
V &= 32.47K \\
V &= 32.58K \\
V &= 32.95R \\
V &= 34.43K
\end{align*}
\]

Solving for Values

\[ V_\pi = \gamma P_\pi V_\pi + R_\pi \]

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

\[ V_\pi = (I - \gamma P_\pi)^{-1} R \]

Guaranteed invertible because \( \gamma P_\pi \) has spectral radius <1
Iteratively Solving for Values

\[ V_\pi = \gamma P_\pi V + R \]

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

\[ V_{\pi, i+1} = \gamma P_\pi V_{\pi, i} + R \]

Guaranteed convergent because \( \gamma P_\pi \)
has spectral radius < 1 for \( \gamma < 1 \)

Convergence not guaranteed for \( \gamma = 1 \)

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 know probability 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
  – \( P(s'|s,a) = \frac{m}{n} \)
  – 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:

\[
V_{\pi^{i+1}}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s' | s, \pi(s)) V_{\pi^i}(s')
\]

Problem: We don’t know this.
Temporal Difference Learning

- Remember Value Determination:
  \[ V^{i+1}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'| s, \pi(s)) V^i(s') \]

- Compute an update as if the observed \( s' \) and \( r \) were the only possible outcomes:
  \[ V^{\text{temp}}(s) = r + \gamma V^i(s') \]

- Make a small update in this direction:
  \[ V^{i+1}(s) = (1 - \alpha)V^i(s) + \alpha V^{\text{temp}}(s) \]
  \[ 0 < \alpha \leq 1 \]

Example: Home Version of Game

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

\[ V^{\text{temp}}=0 \]
\[ V(s_3) = (1-\alpha)15K + \alpha0 \]
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
- $0 \leq \gamma < 1$
- Every state visited infinitely often
- Learning rate decays so that:
  - $\sum_{i=0}^{\infty} \alpha_i(s) = \infty$
  - $\sum_{i=0}^{\infty} \alpha^2_i(s) < \infty$

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: Hmmmm...
- 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
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:

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

– \( s' \) 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:

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

- Can split this into two functions:

\[ Q_{i+1}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)V'(s') \]

\[ V_{i+1}(s) = \max_a Q_{i+1}(s,a) \]
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

Properties of approximate RL

- Table-updates are a special case
- Can be combined with Q-learning

- Convergence not guaranteed
  - Policy evaluation with linear function approximation converges if samples are drawn “on policy”
  - 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
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