Reinforcement Learning

Active Learning

Alan Fern

* Based in part on slides by Daniel Weld

Active Reinforcement Learning

- So far, we've assumed agent ***has*** a policy
  - We just learned how good it is

- Now, suppose agent must learn a good policy (ideally optimal)
  - While acting in uncertain world

Naïve Model-Based Approach

1. Act Randomly for a (long) time
   - Or systematically explore all possible actions
2. Learn
   - Transition function
   - Reward function
3. Apply value/policy iteration to get policy
4. Follow resulting policy thereafter.

Will this work? Yes (if we do step 1 long enough and there are no “dead-ends”)

Any problems? We will act randomly for a long time before exploiting what we know.

Revision of Naïve Approach

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work?

No. Can get stuck in local minima. (depends on initialization)

What can be done?

Exploration versus Exploitation

- Two reasons to take an action in RL
  - **Exploitation**: To try to get reward. We exploit our current knowledge to get a payoff.
  - **Exploration**: Get more information about the world. How do we know if there is not a pot of gold around the corner.
- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic intuition behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

ADP-based (model-based) RL

1. Start with initial model
2. Solve for optimal policy given current model
   (using value or policy iteration)
3. Take action according to an explore/exploit policy
   (explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the explore/exploit policy

Will this work? Depends on the explore/exploit policy.

Any ideas?

Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s') V(s')$$

  - where V is current optimal value function estimate (based on current model), and R, T are current estimates of model
  - Q(s,a) is the expected value of taking action a in state s and then getting the estimated value V(s') of the next state s'

- Want an exploration policy that is greedy in the limit of infinite exploration (GLIE)
  - Guarantees convergence

Explore/Exploit Policies

- GLIE Policy 1
  - On time step t select random action with probability p(t) and greedy action with probability 1-p(t)
  - p(t) = 1/t will lead to convergence, but can be slow

- In practice it is common to simply set p(t) to a small constant ε (e.g. ε=0.1)
  - Called ε-greedy exploration
  - Just as we saw for bandits
  - ε usually set to small value (compared to 0.5) so the trajectories we learn from are mostly based on exploitation behavior

Explore/Exploit Policies

- GLIE Policy 2: Boltzmann Exploration
  - Select action a with probability,

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

  - T is the temperature. Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically start with large T and decrease with time

The Impact of Temperature

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

- Suppose we have two actions and that Q(s,a1) = 1, Q(s,a2) = 2

- T=10 gives Pr(a1 | s) = 0.48, Pr(a2 | s) = 0.52
  - Almost equal probability, so will explore

- T= 1 gives Pr(a1 | s) = 0.27, Pr(a2 | s) = 0.73
  - Probabilities more skewed, so explore a1 less

- T = 0.25 gives Pr(a1 | s) = 0.02, Pr(a2 | s) = 0.98
  - Almost always exploit a2

Alternative Model-Based Approach: Optimistic Exploration

- There is a class of RL algorithms based on the idea of optimistic exploration.
- Basically, if the agent has not explored a state “enough”, then it acts as if that state provides maximum reward
  - So actions will be selected to try and reach such states
- Many of the theoretical results are based on this idea
  - We’ll only touch on the theory
- We’ll cover two views of essentially the same approach
  - The first view from your book
  - The second view that is more standard in the literature

Alternative Model-Based Approach: Optimistic Exploration (view 1)

1. Start with initial model
2. Solve for “optimistic policy”
   (uses optimistic variant of value iteration)
   (inflates value of actions leading to unexplored regions)
3. Take greedy action according to optimistic policy
4. Update estimated model
5. Goto 2

Basically act as if all “unexplored” state-action pairs are maximally rewarding.

Optimistic Value Iteration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) \leftarrow R(s) + \beta \max_a \sum_{s'} T(s,a,s')V(s')$$

  - Optimistic variant adjusts update to make actions that lead to unexplored regions look good

- **Optimistic VI:** assigns highest possible value V^{max} to any state-action pair that has not been explored enough
  - Maximum value is when we get maximum reward forever

$$V^{\max} = \sum_{t=0}^{\infty} \beta^t R^{\max} = \frac{R^{\max}}{1-\beta}$$

- What do we mean by "explored enough"?
  - $N(s,a) > N_e$, where $N(s,a)$ is number of times action a has been tried in state s and N_e is a user selected parameter

Optimistic Value Iteration

$$V(s) \leftarrow R(s) + \beta \max_a \sum_{s'} T(s,a,s')V(s')$$ Standard VI

- Optimistic value iteration computes an optimistic value function V^+ using following updates

$$V^+(s) \leftarrow R(s) + \beta \max_a \begin{cases} V^{\max}, & N(s,a) < N_e \\ \sum_{s'} T(s,a,s')V^+(s'), & N(s,a) \geq N_e \end{cases}$$

- The agent will behave initially as if there were wonderful rewards scattered all over around– **optimistic** .
- But after actions are tried enough times we will perform standard “non-optimistic” value iteration

Optimistic Exploration (view 1)

1. Start with initial model
2. Solve for optimistic policy using optimistic value iteration
3. Take greedy action according to optimistic policy
4. Update estimated model; Goto 2

Can any guarantees be made for the algorithm?

- If N_e is large enough and all state-action pairs are explored that many times, then the model will be accurate and lead to close to optimal policy
- But, perhaps some state-action pairs will never be explored enough or it will take a very long time to do so
- Optimistic exploration is equivalent to another algorithm, Rmax, which has been proven to efficiently converge

Alternative View of Optimistic Exploration: The Rmax Algorithm

1. Start with an optimistic model
   (assign largest possible reward to “unexplored states”)
   (actions from “unexplored states” only self transition)
2. Solve for optimal policy in optimistic model (standard VI)
3. Take greedy action according to policy
4. Update optimistic estimated model
   (if a state becomes “known” then use its true statistics)
5. Goto 2

Agent always acts greedily according to a model that assumes all “unexplored” states are maximally rewarding

Rmax: Optimistic Model

- Keep track of number of times a state-action pair is tried
  - A state is “unknown” if some action is not explored enough from it

- If $N(s,a) < N_e$ then $T(s,a,s)=1$ and $R(s) = Rmax$ in optimistic model,
- Otherwise $T(s,a,s')$ and $R(s)$ are based on estimates obtained from the N_e experiences (the estimate of true model)
  - For large enough N_e these will be accurate estimates

- An optimal policy for this optimistic model will try to reach unexplored states (those with unexplored actions) since it can stay at those states and accumulate maximum reward
- Never explicitly explores. Is always greedy, but with respect to an optimistic outlook.

Optimistic Exploration

- Rmax is equivalent to optimistic exploration via optimistic VI
  - Convince yourself of this.

- Is Rmax provably efficient?
  - If the model is ever completely learned (i.e. N(s,a) > N_e, for all (s,a), then the policy will be near optimal)
  - Recent results show that this will happen “quickly”

- **PAC Guarantee (Roughly speaking):** *There is a value of N_e (depending on n,m, and Rmax), such that with high probability the Rmax algorithm will select at most a polynomial number of action with value less than ε of optimal.*

- RL can be solved in poly-time in n, m, and Rmax!

TD-based Active RL

1. Start with initial value function
2. Take action from explore/exploit policy giving new state s' (should converge to greedy policy, i.e. GLIE)
3. Update estimated model
4. Perform TD update

   $$V(s) \leftarrow V(s) + \alpha(R(s) + \beta V(s') - V(s))$$

   V(s) is new estimate of optimal value function at state s.
5. Goto 2

Just like TD for passive RL, but we follow explore/exploit policy

Given the usual assumptions about learning rate and GLIE, TD will converge to an optimal value function!

TD-based Active RL

1. Start with initial value function
2. Take action from explore/exploit policy giving new state s' (should converge to greedy policy, i.e. GLIE)
3. Update estimated model
4. Perform TD update

$$V(s) \leftarrow V(s) + \alpha(R(s) + \beta V(s') - V(s))$$

   V(s) is new estimate of optimal value function at state s.

5. Goto 2

Requires an estimated model. Why?

To compute the explore/exploit policy.

TD-Based Active Learning

- Explore/Exploit policy requires computing Q(s,a) for the exploit part of the policy
  - Computing Q(s,a) requires T and R in addition to V
- Thus TD-learning must still maintain an estimated model for action selection
- It is computationally more efficient at each step compared to Rmax (i.e. optimistic exploration)
  - TD-update vs. Value Iteration
  - But model requires much more memory than value function
- Can we get a model-fee variant?

Q-Learning: Model-Free RL

- Instead of learning the optimal value function V, directly learn the optimal Q function.
  - Recall Q(s,a) is the expected value of taking action a in state s and then following the optimal policy thereafter
- Given the Q function we can act optimally by selecting action greedily according to Q(s,a) without a model
- The optimal Q-function satisfies $V(s) = \max_{a'} Q(s,a')$ which gives:

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s')V(s')$$

$$= R(s) + \beta \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

How can we learn the Q-function directly?

Q-Learning: Model-Free RL

Bellman constraints on optimal Q-function:

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s') \max_{a'} Q(s,a')$$

- We can perform updates after each action just like in TD.
  - After taking action a in state s and reaching state s' do: (note that we directly observe reward R(s))

$$Q(s,a) \leftarrow Q(s,a) + \alpha \underbrace{(R(s) + \beta \max_{a'} Q(s',a')}_{\text{(noisy) sample of Q-value based on next state}} - Q(s,a))$$

Q-Learning

1. Start with initial Q-function (e.g. all zeros)
2. Take action from explore/exploit policy giving new state s' (should converge to greedy policy, i.e. GLIE)
3. Perform TD update

   $$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \beta \max_{a'} Q(s',a') - Q(s,a))$$

   Q(s,a) is current estimate of optimal Q-function.
4. Goto 2

- Does not require model since we learn Q directly!
- Uses explicit |S|x|A| table to represent Q
- Explore/exploit policy directly uses Q-values
  - E.g. use Boltzmann exploration.
  - Book uses exploration function for exploration (Figure 21.8)

Q-Learning: Speedup for Goal-Based Problems

- **Goal-Based Problem:** receive big reward in goal state and then transition to terminal state
  - Parking domain is goal based
- Consider initializing Q(s,a) to zeros and then observing the following sequence of (state, reward, action) triples
  - (s0,0,a0) (s1,0,a1) (s2,10,a2) (terminal,0)
- The sequence of Q-value updates would result in: Q(s0,a0) = 0, Q(s1,a1) =0, Q(s2,a2)=10
- So nothing was learned at s0 and s1
  - Next time this trajectory is observed we will get non-zero for Q(s1,a1) but still Q(s0,a0)=0

Q-Learning: Speedup for Goal-Based Problems

- From the example we see that it can take many learning trials for the final reward to “back propagate” to early state-action pairs

- Two approaches for addressing this problem:
  1. **Trajectory replay**: store each trajectory and do several iterations of Q-updates on each one
  2. **Reverse updates:** store trajectory and do Q-updates in reverse order

- In our example (with learning rate and discount factor equal to 1 for ease of illustration) reverse updates would give
  - Q(s2,a2) = 10, Q(s1,a1) = 10, Q(s0,a0)=10

Q-Learning: Suggestions for Q-Learning Assignment

- A very simple exploration strategy is ε-greedy exploration (generally called "epsilon greedy")
  - Select a "small" value for e (perhaps 0.1)
  - On each step:
    - With probability ε select a random action, and with probability 1- ε select a greedy action
- But it might be interesting to play with exploration a bit (e.g. compare to a decreasing exploration rate)
- You can use a discount factor of one or close to 1.

Active Reinforcement Learning Summary

- Methods
  - ADP
  - Temporal Difference Learning
  - Q-learning
- All converge to optimal policy assuming a GLIE exploration strategy
  - Optimistic exploration with ADP can be shown to converge in polynomial time with high probability
- All methods assume the world is not too dangerous (no cliffs to fall off during exploration)
- So far we have assumed small state spaces

ADP vs. TD vs. Q

- Different opinions.
- (my opinion) When state space is small then this is not such an important issue.
- Computation Time
  - ADP-based methods use more computation time per step
- Memory Usage
  - ADP-based methods uses $O(mn^2)$ memory
  - Active TD-learning uses $O(mn^2)$ memory (must store model)
  - Q-learning uses $O(mn)$ memory for Q-table
- Learning efficiency (performance per unit experience)
  - ADP-based methods make more efficient use of experience by storing a model that summarizes the history and then reasoning about the model (e.g. via value iteration or policy iteration)

What about large state spaces?

- One approach is to map original state space S to a much smaller state space S' via some lossy hashing function.
  - Ideally "similar" states in S are mapped to the same state in S'
- Then do learning over S' instead of S.
  - Note that the world may not look Markovian when viewed through the lens of S', so convergence results may not apply
  - But, still the approach can work if a good enough S' is engineered (requires careful design), e.g.
  - *Empirical Evaluation of a Reinforcement Learning Spoken Dialogue System.* With S. Singh, D. Litman, M. Walker. *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000
- We will now study three other approaches for dealing with large state-spaces
  - Value function approximation
  - Policy gradient methods
  - Least Squares Policy Iteration