Reinforcement Learning

3/19/18
Known MDPs

MDPs model worlds with discrete states and actions, occasional rewards, and random transitions.

If the MDP is known, we have several planning algs:
• value iteration
• MCTS
• LAO*

All of these algorithms require the agent to have a complete model of the world (it has access to the full MDP).
What if we don’t know the MDP?

• We might not know all the states.
• We might not know the transition probabilities.
• We might not know the rewards.

• The only way to figure it out is to explore.

• We now need two things:
  • A policy to use while exploring.
  • A way to devise an optimal policy from observations.
Known vs. Unknown MDPs

If we know the full MDP:
• All states and actions
• All transition probabilities
• All rewards

We can use value iteration or other algorithms to find an optimal policy before we start acting.

If we don’t know the MDP:
• Missing states
  • Generally know actions
• Missing transition probabilities
• Missing rewards

We need to try out various actions and learn from the results. This is called RL: Reinforcement Learning.
Temporal Difference (TD) Learning

• Used to estimate value from observations.

• Can estimate state values or state-action values.
  • $V(s)$ or $Q(s,a)$

Key idea:

• Update estimates based on experience, using the difference between successive observations.
TD Learning Update

Update estimates based on experience, using the difference between successive observations.

Update rule:  
\[ V(s) = \alpha [R(s) + \gamma V(s')] + (1 - \alpha)V(s) \]

Equivalently:  
\[ V(s) + = \alpha [R(s) + \gamma V(s') - V(s)] \]

(temporal difference)
Where does this come from?

\[ A_k = \frac{v_1 + \ldots + v_k}{k} \]

\[ = \left( 1 - \frac{1}{k} \right) A_{k-1} + \frac{v_k}{k} \]

\[ = (1 - \alpha_k) A_{k-1} + \alpha_k \cdot v_k \]

\[ \alpha_k = \frac{1}{k} \]
TD Learning Convergence

\[(1 - \alpha_k)V_{old} + \alpha_k \cdot V_{new}\]

• If \(\alpha_k\) declines over time, this will converge to correct value estimates.

• Why might we prefer \(\alpha_k > \frac{1}{k}\)?
Demo
How does TD learning work on MDPs?

TD learning maintains no model of the environment.
  • It never learns transition probabilities.

Yet TD learning converges to correct value estimates. **Why?**

Consider how values will be modified...
  • when all values are initially 0.
  • when s’ has a high value.
  • when s’ has a low value.
  • when discount is close to 1.
  • when discount is close to 0.
  • over many, many runs.

\[
V(s) + = \alpha \left[ R(s) + \gamma V(s') - V(s) \right]
\]
Q-learning

Key idea: TD learning on (state, action) pairs.

- $Q(s,a)$ is the expected value of doing action $a$ in state $s$.
- Store $Q$ values in a table; update them incrementally.

Update rule:

$$Q(s, a) = \alpha \left[ R(s) + \gamma \left( \max_{a'} Q(s', a') \right) \right] + (1 - \alpha) Q(s, a)$$

Equivalently:

$$Q(s, a) += \alpha \left[ R(s) + \gamma \left( \max_{a'} Q(s', a') \right) - Q(s, a) \right]$$
Exploration Policy *During* Q-Learning

What policy should we follow while we’re learning (before we have good value estimates)?

- We want to explore: try out each action enough times that we have a good estimate of its value.

- We want to exploit: we update other Q-values based on the best action, so we want a good estimate of the value of the best action.

We need a policy that handles this tradeoff.
- One option: $\varepsilon$-greedy