

Chapter 21

Reinforcement Learning

By Timothy Durnan

Reinforcement

- Feedback about what is good and what is bad
- Reinforcement Learning

Three Types of Designs

- Utility-Based Agent
- Q-learning
- Reflex Agent

Passive Learning

- In state s , always executes action $\pi(s)$
- Similar to the policy evaluation and policy iteration from ch17
- Main Difference: Agent doesn't know the transition model $T(s,a,s')$
- Also doesn't know $R(s)$, the reward for each state
- Uses multiple trials, updating its policy to reflect the expected utility $U^{\pi}(s)$

Direct Utility Estimation

- Utility of current state is expressed as expected total reward from that state onward
- Each trial provides a sample of this value for each state
- Reduces reinforcement learning to inductive learning
- Utilities of states are NOT independent
- Converges very slowly

Adaptive Dynamic Programming Agent

- Learns the transition model as it goes along, solving MDP using dynamic programming
- Process is easy, since environment is fully observable (reduces to a supervised learning problem)
- Provides a nice standard against which to measure other algorithms
- Somewhat intractable for large state spaces (eg, backgammon)

Temporal Difference Learning

- Best of both worlds (Almost)
- Use the observed transitions to adjust the values of the observed states so that they agree with the constraint equations
- When a transition occurs from s to s' , we apply (21.3) to $U^\pi(s)$
- Uses the difference in utilities between successive states

Active Reinforcement Learning

- Must decide what actions to take, rather than relying on a fixed policy
- Trivially* easy to always find the optimal policy's recommendation
- *Trivially, in this instance, must be taken with a grain of salt

Problems

- If we always take the optimal policy, we will *never learn the true utilities*
- Basically just a "greedy agent"
- Very seldom converges to the optimal policy
- Therefore we need something else

Exploration

- By improving the model, the agent will receive greater rewards in the future (theory of information value, Ch16)
- Need a trade-off between exploitation and exploration
- GLIE: Greedy in the Limit of Infinite Exploration
- Try each action in each state an unbounded number of times

GLIE

- Try a random action a fraction $1/t$ of the time and follow greedy policy otherwise
- Converges, but slowly
- Weight actions that agent hasn't tried very often, while avoiding actions that have low utility
- $U^+(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U^+(s') , N(a,s)$

Learning These Functions

- Note that the TD update function doesn't change
- TD will converge to same values as ADP as training sequences \rightarrow infinity
- alternative to TD, called Q-learning
- Learns action-value representation instead of learning utilities
- $U(s) = \max_a Q(a, s)$
- *A TD agent that learns a Q-function does not need a model for either learning or action selection*

Q-Learning Agent

function Q-Learning-Agent(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r'

static: Q , a table of action values index by state and action

N_{sa} , a table of frequencies for state and action

s, a, r , the previous state, action, and reward, initially null

if s is not null **then do**

 increment $N_{sa}[s, a]$

$Q[a, s] \leftarrow Q[a, s] + \alpha(N_{sa}[s, a]) (r + \gamma \max_{a'} Q[a', s] - Q[a, s])$

if TERMINAL?[s] **then** $s, a, r \leftarrow$ null

else $s, a, r \leftarrow s', \operatorname{argmax}_{a'} f(Q[a', s'], N_{sa}[s', a'], r')$

return a

Generalization

- Chess, backgammon, have incredible state spaces (10^{50} to 10^{120} states)
- Absurd to try and visit all of these states in order to learn how to play the game
- Use function approximation
- Bonus: Allows the agent to generalize from states it has visited to states it has not visited

Linear vs Nonlinear

- Linear function approximation only requires linearity in the the parameters -- the features themselves can be arbitrary nonlinear functions of the state variables
- Can be shown to converge to the "closest possible" approximation
- All bets are off when nonlinear functions (like neural networks) are used
- Very delicate art

Applications to Game-Playing

- Arthur Samuel (1959, 1967) checker-playing
- Interesting methodology of using observed rewards
- Gerald Tesauro's TD-Gammon (1992)
- Reward was only given at the terminal states
- Able to achieve world-champion status

Applications to Robot Control

- “Cart-pole” problem, aka “inverted pendulum”
- Over two thousand papers published on this seemingly simple problem

Summary

- Model-based design, using a model T and utility function U
- Model-free design, using action-value function Q
- Reflex design, using policy π

Summary cont.

- Utilities:
 - Direct utility estimation
 - Adaptive dynamic programming
 - Temporal-difference
- Q-functions can be combined with ADP or TD
- No model necessary in either learning or action-selection for TD

Summary cont.

- Exploitation vs exploration
- Exact solution infeasible, but simple heuristics do a reasonable job
- Large state spaces necessitate an approximation function

Final Words

- Potential for eliminating hand coding of control strategies
- Therefore, one of the most active areas of machine learning
- Original idea proposed by Turing (1948, 1950)