Lecture 18

Introduction to Reinforcement Learning: *Q* Learning

Thursday, October 28, 1999

William H. Hsu
Department of Computing and Information Sciences, KSU
http://www.cis.ksu.edu/~bhsu

Readings:
Sections 13.3-13.4, Mitchell
Sections 20.1-20.2, Russell and Norvig

Lecture Outline

- Readings: Chapter 13, Mitchell; Sections 20.1-20.2, Russell and Norvig
  - Today: Sections 13.1-13.4, Mitchell
  - Review: "Learning to Predict by the Method of Temporal Differences", Sutton
- Suggested Exercises: 13.2, Mitchell; 20.5, 20.6, Russell and Norvig
- Control Learning
  - Control policies that choose optimal actions
  - MDP framework, continued
  - Issues
    - Delayed reward
    - Active learning opportunities
    - Partial observability
    - Reuse requirement
- *Q* Learning
  - Dynamic programming algorithm
  - Deterministic and nondeterministic cases; convergence properties

Control Learning

- Learning to Choose Actions
  - Performance element
    - Applying policy in uncertain environment (last time)
    - Control, optimization objectives: belong to intelligent agent
  - Applications: automation (including mobile robotics), information retrieval
- Examples
  - Robot learning to dock on battery charger
  - Learning to choose actions to optimize factory output
  - Learning to play Backgammon
- Problem Characteristics
  - Delayed reward: loss signal may be episodic (e.g., win-loss at end of game)
  - Opportunity for active exploration: situated learning
  - Possible partially observability of states
  - Possible need to learn multiple tasks with same sensors, effectors (e.g., actuators)

Example: *TD-Gammon*

- Learns to Play Backgammon [Tesauro, 1995]
  - Predecessor: *NeuroGammon* [Tesauro and Sejnowski, 1989]
    - Learned from examples of labelled moves (very tedious for human expert)
    - Result: strong computer player, but not grandmaster-level
  - *TD-Gammon*: first version, 1992 - used reinforcement learning
- Immediate Reward
  - +100 if win
  - -100 if loss
  - 0 for all other states
- Learning in *TD-Gammon*
  - Algorithm: temporal differences [Sutton, 1988] - next time
  - Training: playing 200000 - 1.5 million games against itself (several weeks)
  - Learning curve: improves until ~1.5 million games
  - Result: now approximately equal to best human player (won World Cup of Backgammon in 1992; among top 3 since 1995)

Reinforcement Learning: Problem Definition

- Interactive Model
  - State (may be partially observable), incremental reward presented to agent
  - Agent selects actions based upon (current) policy
  - Taking action puts agent into new state in environment
    - New reward: reinforcement (feedback)
    - Agent uses decision cycle to estimate new state, compute outcome distributions, select new actions
- Reinforcement Learning Problem
  - Given
    - Observation sequence $a_0 : r_0$ $a_1 : r_1$ $a_2 : r_2$
    - Discount factor $\gamma \in [0, 1)$
  - Learn to: choose actions that maximize $r(t) + \gamma r(t + 1) + \gamma^2 r(t + 2) + \ldots$

Quick Review: Markov Decision Processes

- Markov Decision Processes (MDPs)
  - Components
    - Finite set of states S
    - Set of actions A
  - At each time, agent
    - observes state $s(t) \in S$ and chooses action $a(t) \in A$;
    - then receives reward $r(t)$,
    - and state changes to $s(t + 1)$
  - Markov property, *aka* Markov assumption: $s(t + 1) = \delta(t + 1)$ and $r(t) = r(s(t), a(t))$
    - i.e., $r(t)$ and $s(t + 1)$ depend only on *current* state and action
    - Previous history $s(0), s(1), \ldots, s(t - 1)$: irrelevant
    - i.e., $s(t + 1)$ conditionally independent of $s(0), s(1), \ldots, s(t - 1)$ given $s(t)$
    - δ, r: may be nondeterministic; not necessarily known to agent
  - Variants: totally observable (accessible), partially observable (inaccessible)
- Criterion for $a(t)$: Total Reward – Maximum Expected Utility (MEU)

Agent's Learning Task

- Performance Element
  - Execute actions in environment, observe results
  - Learn action policy π : *state* $\rightarrow$ *action* that maximizes expected discounted reward $E[r(t) + \gamma r(t + 1) + \gamma^2 r(t + 2) + \ldots]$ from any starting state in S
  - $\gamma \in [0, 1)$
    - Discount factor on future rewards
    - Expresses preference for rewards sooner rather than later
- Note: Something New!
  - Target function is π : *state* $\rightarrow$ *action*
  - However…
    - We have no training examples of form <*state*, *action*>
    - Training examples are of form <<*state*, *action*>, *reward*>

Value Function

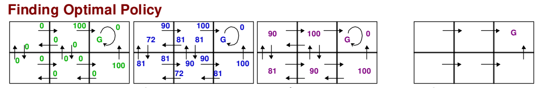
- First Learning Scenario: Deterministic Worlds
  - Agent considers adopting policy π from policy space Π
  - For each possible policy $\pi \in \Pi$, can define an evaluation function over states:

$$V^\pi(s) \equiv r(t) + \gamma r(t+1) + \gamma^2 r(t+1) + \ldots$$
$$\equiv \sum_{i=0}^{\infty} \gamma^i r(t+i)$$

  where $r(t)$, $r(t + 1)$, $r(t + 2)$, … are generated by following policy π starting at state s
  - Restated, task is to learn optimal policy π^*

$$\pi^* \equiv \arg\max_{\pi} V^\pi(s), \forall s$$

- Finding Optimal Policy



r(*state*, *action*) immediate reward values | *Q*(*state*, *action*) values | V^*(*state*) values | One optimal policy

What to Learn

- Idea
  - Might have agent try to learn evaluation function V^{π^*} (abbreviated V^*)
  - Could then perform lookahead search to choose best action from any state s, because:

$$\pi^*(s) \equiv \arg\max_a \left[r(s, a) + V^*(\delta(s, a))\right]$$

- Problem with Idea
  - Works well if agent knows
    - δ : *state* $\times$ *action* $\rightarrow$ *state*
    - r : *state* $\times$ *action* $\rightarrow \mathbb{R}$
  - When agent doesn't know δ and r, cannot choose actions this way

Q Function

- Solution Approach
  - Define new function very similar to V^*

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

  - If agent learns Q, it can choose optimal action even without knowing δ!
- Using Learned Q
  - Q: evaluation function to be learned by agent
  - Apply Q to select action
    - Idealized, computed policy (this is your brain without Q-learning):

$$\pi^*(s) \equiv \arg\max_a \left[r(s, a) + V^*(\delta(s, a))\right]$$

    - Approximated policy (this is your brain with Q-learning):

$$\pi^*(s) \equiv \arg\max_a Q(s, a)$$

Training Rule to Learn *Q*

- Developing Recurrence Equation for Q
  - Note: Q and V^* closely related

$$V^*(s) = \arg\max_{a'} Q(s, a')$$

  - Allows us to write Q recursively as

$$Q(s(t), a(t)) = r(s(t), a(t)) + \gamma V^*(\delta(s(t), a(t)))$$
$$= r(s(t), a(t)) + \gamma \max_{a'} Q(s(t+1), a')$$

  - Nice! Let $\hat{Q}$ denote learner's current approximation to Q
- Training Rule

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$$

  - s': state resulting from applying action a in state s
  - (Deterministic) transition function δ made implicit
  - Dynamic programming: iterate over table of possible a' values

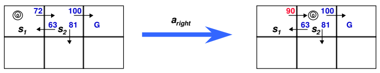
Q Learning for Deterministic Worlds

- (Nonterminating) Procedure for Situated Agent
- Procedure *Q-Learning-Deterministic* (*Reinforcement-Stream*)
  - *Reinforcement-Stream*: consists of <<*state*, *action*>, *reward*> tuples
  - FOR each <*s*, *a*> DO
    - Initialize table entry $\hat{Q}(s, a) \leftarrow 0$
  - Observe current state s
  - WHILE (true) DO
    - Select action a and execute it
    - Receive immediate reward r
    - Observe new state s'
    - Update table entry for $\hat{Q}(s, a)$ as follows

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$$

    - Move: record transition from s to s'

Updating the *Q* Estimate



Initial state: s_1 Next state: s_2

- **Example: Propagating Credit (*Q*) for Candidate Action**
  - ◎ glyph: denotes mobile robot
  - Initial state: s_1 (upper left)
  - Let discount factor γ be 0.9
  - *Q* estimate

$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \cdot \max\{63, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

- **Property**
  - If rewards nonnegative, then $\hat{Q}$ increases monotonically between 0 and true Q
  - $r(s, a) \geq 0 \Rightarrow \forall\, s, a, n\,.\, \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a) \wedge \forall\, s, a, n\,.\, 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$

Convergence

- **Claim**
  - $\hat{Q}$ converges to Q
  - Scenario: deterministic world where <*s*, *a*> are observed (visited) infinitely often
- **Proof**
  - Define full interval: interval during which each <*s*, *a*> is visited
  - During each full interval, largest error in $\hat{Q}$ table is reduced by factor of γ
  - Let $\hat{Q}_n$ be table after n updates and Δ_n be the maximum error in $\hat{Q}_n$; that is,

$$\Delta_n = \max_{s,a} \left|\hat{Q}_n(s, a) - Q(s, a)\right|$$

  - For any table entry $\hat{Q}_n(s, a)$, updated error in revised estimate $\hat{Q}_{n+1}(s, a)$ is

$$\begin{aligned}\left|\hat{Q}_{n+1}(s, a) - Q(s, a)\right| &= \left|r + \gamma \max_{a'} \hat{Q}_n(s', a') - \left(r + \gamma \max_{a'} Q_n(s', a')\right)\right| \\ &= \gamma \left|\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')\right| \\ &\leq \gamma \max_{a'} \left|\hat{Q}_n(s', a') - Q(s', a')\right| \leq \gamma \max_{s'', a'} \left|\hat{Q}_n(s'', a') - Q(s'', a')\right|\end{aligned}$$

$$\left|\hat{Q}_{n+1}(s, a) - Q(s, a)\right| \leq \gamma \Delta_n$$

  - Note: used general fact, $\left|\max_a f_1(a) - \max_a f_2(a)\right| \leq \max_a \left|f_1(a) - f_2(a)\right|$

Nondeterministic Case

- **Second Learning Scenario: Nondeterministic World (Nondeterministic MDP)**
  - What if reward and next state are nondeterministically selected?
  - i.e., *reward function and transition function are nondeterministic*
  - Nondeterminism may express many kinds of uncertainty
    - Inherent uncertainty in dynamics of world
    - Effector exceptions (qualifications), side effects (ramifications)
- **Solution Approach**
  - Redefine *V*, *Q* in terms of expected values

$$\begin{aligned}V^{\pi}(s) &\equiv E\left[r(t) + \gamma r(t+1) + \gamma^2 r(t+1) + \ldots\right] \\ &\equiv E\left[\sum_{i=0}^{\infty} \gamma^i r(t+i)\right]\end{aligned}$$

  - Introduce decay factor; retain some of previous Q value
  - Compare: momentum term in ANN learning

Nondeterministic Case: Generalizing *Q* Learning

- **Q Learning Generalizes to Nondeterministic Worlds**
  - Alter training rule to

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n\left[r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a')\right]$$

  - Decaying weighted average

$$\alpha_n = \frac{1}{1 + visits_n(s, a)}$$

  - $visits_n(s, a)$: total number of times <*s*, *a*> has been visited by iteration *n*, inclusive
  - *r*: observed reward (may also be stochastically determined)
- **Can Still Prove Convergence of $\hat{Q}$ to *Q* [Watkins and Dayan, 1992]**
- **Intuitive Idea**
  - $\alpha \in [0, 1]$: discounts estimate by number of visits, *prevents oscillation*
  - Tradeoff
    - More gradual revisions to $\hat{}$
    - Able to deal with stochastic environment: $P(s' \mid s, a)$, $P(r \mid s, s', a)$

Terminology

- **Reinforcement Learning (RL)**
  - RL: learning to choose optimal actions from <*state*, *reward*, *action*> observations
  - Scenarios
    - Delayed reward: reinforcement is deferred until end of episode
    - Active learning: agent can control collection of experience
    - Partial observability: may only be able to observe rewards (*must infer state*)
    - Reuse requirement: sensors, effectors may be required for multiple tasks
- **Markov Decision Processes (MDPs)**
  - Markovity (*aka* Markov property, Markov assumption): CI assumption on states over time
  - Maximum expected utility (MEU): maximum expected *total reward* (under additive decomposition assumption)
- ***Q* Learning**
  - Action-value function $Q : state \times action \rightarrow value$
  - *Q* learning: training rule and dynamic programming algorithm for RL

Summary Points

- **Control Learning**
  - Learning policies from <*state*, *reward*, *action*> observations
  - Objective: choose optimal actions given new percepts and incremental rewards
  - Issues
    - Delayed reward
    - Active learning opportunities
    - Partial observability
    - Reuse of sensors, effectors
- ***Q* Learning**
  - Action-value function $Q : state \times action \rightarrow value$ (expected utility)
  - Training rule
  - Dynamic programming algorithm
  - *Q* learning for deterministic worlds
  - Convergence to true *Q*
  - Generalizing *Q* learning to nondeterministic worlds
- **Next Week: More Reinforcement Learning (Temporal Differences)**