Machine Learning 10-701

Tom M. Mitchell
Machine Learning Department
Carnegie Mellon University

April 26, 2011

Today:
• Learning of control policies
• Markov Decision Processes
• Temporal difference learning
• Q learning

Readings:
• Mitchell, chapter 13
• Kaelbling, et al., Reinforcement Learning: A Survey

Thanks to Aarti Singh for several slides

Reinforcement Learning

[Sutton and Barto 1981; Samuel 1957; ...]

\[ V^*(s) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...] \]
Reinforcement Learning: Backgammon

[Tesauro, 1995]

Learning task:
• chose move at arbitrary board states

Training signal:
• final win or loss

Training:
• played 300,000 games against itself

Algorithm:
• reinforcement learning + neural network

Result:
• World-class Backgammon player

Outline

• Learning control strategies
  – Credit assignment and delayed reward
  – Discounted rewards

• Markov Decision Processes
  – Solving a known MDP

• Online learning of control strategies
  – When next-state function is known: value function \( V'(s) \)
  – When next-state function unknown: learning \( Q'(s,a) \)

• Role in modeling reward learning in animals
The task: learn a policy \( \pi : S \rightarrow A \) for choosing actions that maximizes

\[
E'[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots] \quad 0 < \gamma \leq 1
\]

for every possible starting state \( s_0 \).
Reinforcement Learning Task for Autonomous Agent

Execute actions in environment, observe results, and
• Learn control policy $\pi: S \rightarrow A$ that maximizes $\sum_{t=0}^{\infty} \gamma^t E[r_t]$ from every state $s \in S$

Example: Robot grid world, deterministic reward $r(s,a)$

$\gamma \leq 1$  
$\gamma > 1$  
$\gamma = 0$  
$\gamma = 1$

$r(s, a)$ (immediate reward)

Yikes!!
• Function to be learned is $\pi: S \rightarrow A$
• But training examples are not of the form $<s, a>$
• They are instead of the form $<s, a>, r>
Value Function for each Policy

- Given a policy $\pi : S \rightarrow A$, define
  
  $$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

  assuming action sequence chosen according to $\pi$, starting at state $s$

- Then we want the optimal policy $\pi^*$ where
  
  $$\pi^* = \arg \max_\pi V^\pi(s), \quad (\forall s)$$

- For any MDP, such a policy exists!
- We'll abbreviate $V^{\pi^*}(s)$ as $V^*(s)$
- Note if we have $V^*(s)$ and $P(s_{t+1}|s_t, a)$, we can compute $\pi^*(s)$

Value Function – what are the $V^\pi(s)$ values?

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

Suppose $\mathcal{M}$ is shown by circled action from each state
Suppose $\gamma = 0.9$

$$r(s, a) \text{ (immediate reward)}$$

Tom Mitchell, April 2011
Value Function – what are the $V^*(s)$ values?

$$V^\pi(s) = E[ \sum_{t=0}^{\infty} \gamma^t r_t]$$

$r(s, a)$ (immediate reward)
Recursive definition for $V^*(S)$

$$V^*(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right]$$

assuming actions are chosen according to the optimal policy, $\pi^*$

$$V^*(s_1) = E[r(s_1, a_1)] + E[\gamma r(s_2, a_2)] + E[\gamma^2 r(s_3, a_3)] + \ldots$$

$$V^*(s_1) = E[r(s_1, a_1)] + \gamma E_{s_2|s_1, a_1}[V^*(s_2)]$$

$$V^*(s) = E[r(s, \pi^*(s))] + \gamma E_{s'|s, \pi^*(s)}[V^*(s')]$$

Value Iteration for learning $V^*$: assumes $P(S_{t+1}|S_t, A)$ known

Initialize $V(s)$ arbitrarily

Loop until policy good enough

Loop for $s$ in $S$

Loop for $a$ in $A$

$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V(s')$$

$$V(s) \leftarrow \max_a Q(s, a)$$

End loop

End loop

$V(s)$ converges to $V^*(s)$

Dynamic programming
Value Iteration

Interestingly, value iteration works even if we randomly traverse the environment instead of looping through each state and action methodically

- but we must still visit each state infinitely often on an infinite run
- For details: [Bertsekas 1989]
- Implications: online learning as agent randomly roams

If max (over states) difference between two successive value function estimates is less than $\epsilon$, then the value of the greedy policy differs from the optimal policy by no more than

$$\frac{2\epsilon \gamma}{(1 - \gamma)}$$

So far: learning optimal policy when we know $P(s_t | s_{t-1}, a_{t-1})$

What if we don’t?
Q learning

Define new function, closely related to $V^*$

\[
V^*(s) = E[r(s, \pi^*(s)) + \gamma E_{s'\mid s,\pi^*(s)}[V^*(s')]]
\]

\[
Q(s, a) = E[r(s, a) + \gamma E_{s'\mid s, a}[V^*(s')]]
\]

If agent knows $Q(s, a)$, it can choose optimal action without knowing $P(s_{t+1} \mid s_t, a)$!

\[
\pi^*(s) = \arg \max_a Q(s, a) \quad V^*(s) = \max_a Q(s, a)
\]

And, it can learn $Q$ without knowing $P(s_{t+1} \mid s_t, a)$

Immediate rewards $r(s, a)$

State values $V^*(s)$

State-action values $Q^*(s, a)$

$V^*(s) = E[r(s, \pi^*(s)) + \gamma E_{s'\mid s,\pi^*(s)}[V^*(s')]]$

Bellman equation.

Consider first the case where $P(s' \mid s, a)$ is deterministic
Training Rule to Learn $Q$

Note $Q$ and $V^*$ closely related:

\[ V^*(s) = \max_{a'} Q(s, a') \]

Which 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 $Q$. Consider training rule

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

where $s'$ is the state resulting from applying action $a$ in state $s$

---

$Q$ Learning for Deterministic Worlds

For each $s, a$ initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Observe current state $s$

Do forever:

- Select an action $a$ and execute it
- Receive immediate reward $r$
- Observe the new state $s'$
- Update the table entry for $\hat{Q}(s, a)$ as follows:
  \[ \hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a') \]
- $s \leftarrow s'$
Updating $\hat{Q}$

\[
\hat{Q}(s_1, a_{\text{right}}) = r + \gamma \max_{a'} \hat{Q}(s_2, a')
\]

\[
= 0 + 0.9 \max\{63, 81, 100\}
\]

\[
= 90
\]

notice if rewards non-negative, then

\[
(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)
\]

and

\[
(\forall s, a, n) \quad 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)
\]

\[
\hat{Q}
\]

converges to $Q$ Consider case of deterministic world where see each $(s, a)$ visited infinitely often.

Proof: Define a full interval to be an interval during which each $(s, a)$ is visited. During each full interval the largest error in $\hat{Q}$ table is reduced by factor of $\gamma$. Discount factor

Let $Q_n$ be table after $n$ updates, and $\Delta_n$ be the maximum error in $\hat{Q}_n$; that is

\[
\Delta_n = \max_s \max_a |\hat{Q}_n(s, a) - Q(s, a)|
\]

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

\[
|\hat{Q}_{n+1}(s, a) - Q(s, a)| = |(r + \gamma \max_{a'} \hat{Q}_n(s', a')) - (r + \gamma \max_{a'} \hat{Q}_n(s', a'))|
\]

\[
= |(r + \gamma \max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a'))|
\]

\[
\leq \gamma |\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')|
\]

\[
\leq \gamma \max_{a'} |\hat{Q}_n(s', a') - Q(s', a')|
\]

\[
|\hat{Q}_{n+1}(s, a) - Q(s, a)| \leq \gamma \Delta_n
\]
Nondeterministic Case

$Q$ learning generalizes to nondeterministic worlds

Alter training rule to

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

where

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$$

Can still prove convergence of $\hat{Q}$ to $Q$ [Watkins and Dayan, 1992]

Temporal Difference Learning

$Q$ learning: reduce discrepancy between successive $Q$ estimates

One step time difference:

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

Why not two steps?

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)$$

Or $n$?

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)$$

Blend all of these:

$$Q^\lambda(s_t, a_t) \equiv (1 - \lambda) [Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t)]$$
Temporal Difference Learning

\[ Q^\lambda(s_t, a_t) \equiv (1 - \lambda) \left[ Q^1(s_t, a_t) + \lambda Q^2(s_t, a_t) + \lambda^2 Q^3(s_t, a_t) \right] \]

Equivalent expression:

\[ Q^\lambda(s_t, a_t) = r_t + \gamma \left[ (1 - \lambda) \max_a \hat{Q}^\lambda(s_t, a_t) + \lambda Q^\lambda(s_{t+1}, a_{t+1}) \right] \]

TD(\lambda) algorithm uses above training rule

- Sometimes converges faster than Q learning
- converges for learning \( V^* \) for any \( 0 \leq \lambda \leq 1 \) (Dayan, 1992)
- Tesauro’s TD-Gammon uses this algorithm

---

MDP’s and RL: What You Should Know

- Learning to choose optimal actions \( A \)
- From delayed reward
- By learning evaluation functions like \( V(S), Q(S,A) \)

Key ideas:

- If next state function \( S_t \times A_t \rightarrow S_{t+1} \) is known
  - can use dynamic programming to learn \( V(S) \)
  - once learned, choose action \( A_t \) that maximizes \( V(S_{t+1}) \)
- If next state function \( S_t \times A_t \rightarrow S_{t+1} \) unknown
  - learn \( Q(S_t, A_t) = E[V(S_{t+1})] \)
  - to learn, sample \( S_t \times A_t \rightarrow S_{t+1} \) in actual world
  - once learned, choose action \( A_t \) that maximizes \( Q(S_t, A_t) \)
MDPs and Reinforcement Learning: Further Issues

- What strategy for choosing actions will optimize
  - learning rate? (explore uninvestigated states)
  - obtained reward? (exploit what you know so far)

- Partially observable Markov Decision Processes
  - state is not fully observable
  - maintain probability distribution over possible states you’re in

- Convergence guarantee with function approximators?
  - our proof assumed a tabular representation for Q, V
  - some types of function approximators still converge (e.g., nearest neighbor) [Gordon, 1999]

- Correspondence to human learning?