Machine Learning

Machine Learning: Jordan Boyd-Graber
University of Maryland
REINFORCEMENT LEARNING

Slides adapted from Tom Mitchell and Peter Abeel

Control Learning

Consider learning to choose actions, e.g.,

- Roomba learning to dock on battery charger
- Learning to choose actions to optimize factory output
- Learning to play Backgammon

Note several problem characteristics:

- Delayed reward
- Opportunity for active exploration
- Possibility that state only partially observable
- Possible need to learn multiple tasks with same sensors/effectors

One Example: TD-Gammon

[Tesauro, 1995]

Learn to play Backgammon

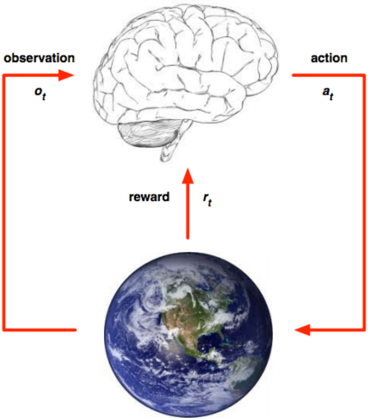
Immediate reward

- +100 if win
- -100 if lose
- 0 for all other states

Trained by playing 1.5 million games against itself

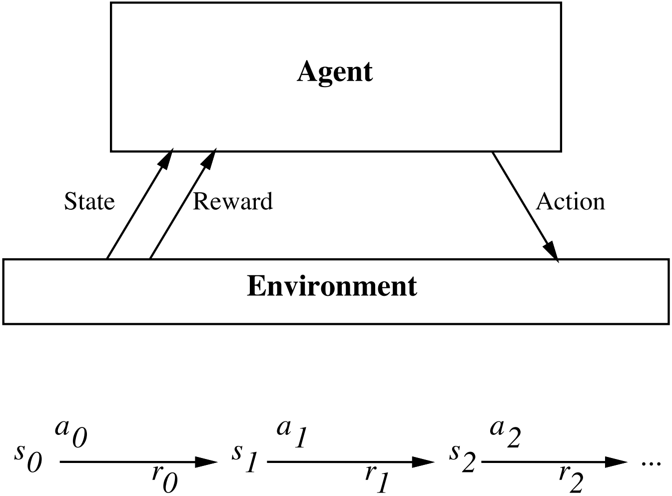
Now approximately equal to best human player

Reinforcement Learning Problem



- At each step t the agent:
  - Executes action a_t
  - Receives observation o_t
  - Receives scalar reward r_t
- The environment:
  - Receives action a_t
  - Emits observation o_{t+1}
  - Emits scalar reward r_{t+1}

Reinforcement Learning Problem



Markov Decision Processes

Assume

- finite set of states S
- set of actions A
- at each discrete time agent observes state $s_t \in S$ and chooses action $a_t \in A$
- then receives immediate reward r_t
- and state changes to s_{t+1}
- Markov assumption: $s_{t+1} = \delta(s_t, a_t)$ and $r_t = r(s_t, a_t)$
  - i.e., r_t and s_{t+1} depend only on current state and action
  - functions δ and r may be nondeterministic
  - functions δ and r not necessarily known to agent

State

- Experience is a sequence of observations, actions, rewards

$$o_1, r_1, a_1, \ldots, a_{t1}, o_t, r_t \tag{1}$$

- The state is a summary of experience

$$s_t = f(o_1, r_1, a_1, \ldots, a_{t1}, o_t, r_t) \tag{2}$$

- In a fully observed environment

$$s_t = f(o_t) \tag{3}$$

Agent's Learning Task

Execute actions in environment, observe results, and

- learn action policy $\pi : S \rightarrow A$ that maximizes

$$\mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

  from any starting state in S
- here $0 \leq \gamma < 1$ is the discount factor for future rewards

Note something new:

- Target function is $\pi : S \rightarrow A$
- but we have no training examples of form $\langle s, a \rangle$
- training examples are of form $\langle \langle s, a \rangle, r \rangle$

What makes an RL agent?

- Policy: agent's behaviour function
- Value function: how good is each state and/or action
- Model: agent's representation of the environment

Policy

- A policy is the agent's behavior
  - It is a map from state to action:
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a \mid s) = p(a \mid s)$

Value Function

To begin, consider deterministic worlds ...

For each possible policy π the agent might adopt, we can define an evaluation function over states

$$\begin{aligned} V^{\pi}(s) &\equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \\ &\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned}$$

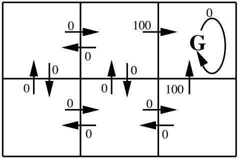
where $r_t, r_{t+1}, \ldots$ are from following policy π starting at state s

Q-learning

Restated, the task is to learn the optimal policy π^*

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

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



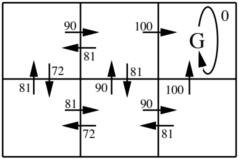
- $Q(s,a)$ values
- One optimal policy

Q-learning

Restated, the task is to learn the optimal policy π^*

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

- $r(s,a)$ (immediate reward) values
- $Q(s,a)$ values



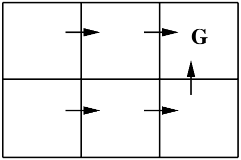
- One optimal policy

Q-learning

Restated, the task is to learn the optimal policy π^*

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

- $r(s,a)$ (immediate reward) values
- $Q(s,a)$ values
- One optimal policy



What to Learn

We might try to have agent learn the evaluation function V^{π^*} (which we write as V^*)
It could then do a lookahead search to choose best action from any state s because

$$\pi^*(s) = \arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

A problem:

- This works well if agent knows $\delta : S \times A \rightarrow S$, and $r : S \times A \rightarrow \Re$
- But when it doesn't, it can't choose actions this way

Q Function

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!

$$\pi^*(s) = \arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

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

Q is the evaluation function the agent will learn

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

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t))) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

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

Value Function

- A value function is a prediction of future reward: "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ (future rewards mean less than immediate)

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s,a \right] \tag{4}$$

A Value Function is Great!

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a) \tag{5}$$

- If you know the value function, you can derive policy

$$\pi^* = \arg\max_{a} Q(s,a) \tag{6}$$

Q Learning for Deterministic Worlds

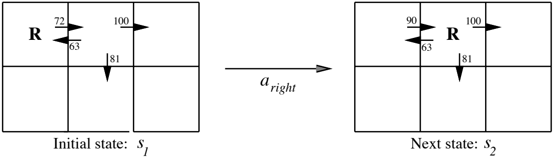
For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
Observe current state s
Do forever:

- 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 $\hat{Q}$



$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$
$$\leftarrow 0 + 0.9 \ \max\{63, 81, 100\} = 90$$

if rewards non-negative, then

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

and

$$(\forall s, a, n) \ \ 0 \leq \hat{Q}_n(s,a) \leq Q(s,a)$$

$\hat{Q}$ converges to Q.

Nondeterministic Case

What if reward and next state are non-deterministic?
We redefine V, Q by taking expected values

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

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

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:

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

Temporal Difference Learning

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

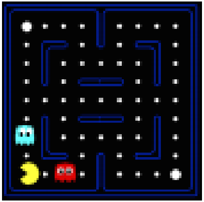
Equivalent expression:

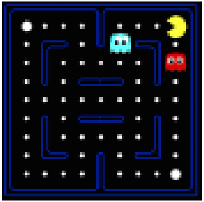
$$\begin{aligned} Q^{\lambda}(s_t, a_t) &= r_t + \gamma[\quad (1-\lambda)\max_a \hat{Q}(s_t, a_t) \\ &\qquad\qquad + \lambda\, Q^{\lambda}(s_{t+1}, a_{t+1})] \end{aligned}$$

TD(λ) 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

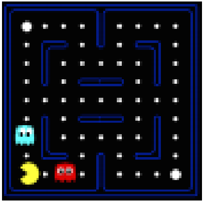
What if the number of states is huge and/or structured?

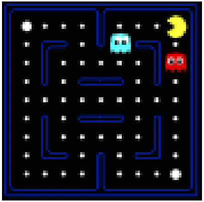




- Let's say we discover that state is bad
- In Q learning, we know nothing about similar states

What if the number of states is huge and/or structured?





- Let's say we discover that state is bad
- In Q learning, we know nothing about similar states
- Solution: Feature-based Representation
  - Distance to closest ghost
  - Distance to closest dot
  - Number of ghosts
  - Is Pacman in a tunnel?