

Comments on Searle article?

Reading assignment for Wednesday:

1. M. Gasser, Introduction to reinforcement learning (download from class web page)
2. Vehicles: pp. 20-32

No other homework today. Reading questions and exercises to be given out on Wednesday

Heuristic Search

Best-first greedy search:

1. *current state = initial state*
2. Expand *current state*
3. Evaluate offspring states s with heuristic $h(s)$, which estimates cost of path from s to goal state
4. *current state = argmin_s h(s) for $s \in \text{offspring}(\text{current state})$*
5. If *current state* \neq goal state, go to step 2.

Search Terminology

Completeness

- *solution will be found, if it exists*

Optimality

- *least cost solution will be found*

Admissible heuristic h

- $\forall s, h$ never overestimates true cost from state s to goal state

Best first greedy search: Complete? Optimal?

A* Search

Uses evaluation function $f = g + h$

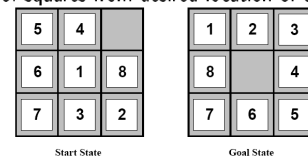
1. g is a cost function
 - Total cost incurred so far from initial state
2. h is an admissible heuristic

Best first search is A* with $g = 0$.

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



$h_1(\text{start state}) =$

$h_2(\text{start state}) =$

Example of A*

A* Algorithm

ComputeShortestPath()

01. while ($\text{argmin}_{s \in \text{OPEN}}(g(s) + h(s, s_{\text{goal}})) \neq s_{\text{goal}}$)
02. remove state s from the front of OPEN;
03. for all $s' \in \text{Succ}(s)$
04. if ($g(s') > g(s) + c(s, s')$)
05. $g(s') = g(s) + c(s, s')$;
06. insert s' into OPEN with value ($g(s') + h(s', s_{\text{goal}})$);

Main()

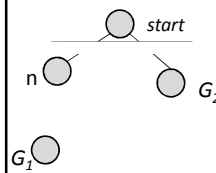
07. for all $s \in S$
08. $g(s) = \infty$;
09. $g(s_{\text{start}}) = 0$;
10. OPEN = \emptyset ;
11. insert s_{start} into OPEN with value ($g(s_{\text{start}}) + h(s_{\text{start}}, s_{\text{goal}})$);
12. ComputeShortestPath();

A* search is optimal and complete

Proof of Optimality of A*

Suppose a suboptimal goal G_2 has been generated and is in the OPEN list.

Let n be an unexpanded node on a shortest path to an optimal goal G_1 .



$f(G_2) = g(G_2)$ since $h(G_2) = 0$

$> g(G_1)$ since G_2 is suboptimal

$f(G_2) > f(n)$ since h is admissible

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Variations of A*

- IDA* (iterative deepening A*)
- ARA* (anytime repairing A*)
- D* (dynamic A*)

Applications?

Learning

- Supervised
- Unsupervised
- Semi-supervised

Semi-Supervised Learning

Consider a learning problem in which a robot has to learn to do tasks in a particular environment.

E.g.,

- Navigate without crashing into anything
- Locate and retrieve an object
- Perform some multi-step manipulation of objects resulting in a desired configuration (e.g., sorting objects)

Robot gets intermittent “rewards” for doing “the right thing”, or “punishment” for doing “the wrong thing”.

Robot has to learn evaluation function for states based on this feedback

Robot has to learn mapping from states to actions (“policy”)

Ideas for this have been inspired by “reinforcement learning” experiments in psychology literature.

Applications of reinforcement learning: A few examples

Learning to play backgammon

Robot arm control (juggling)

Robo-soccer

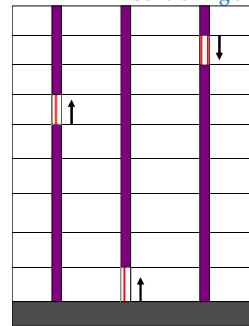
Elevator dispatching

Power systems stability control

Job-shop scheduling

Example

Controlling a bank of elevators



Sensor information:

- State of up and down buttons outside the elevator on each floor (pressed or not-pressed)

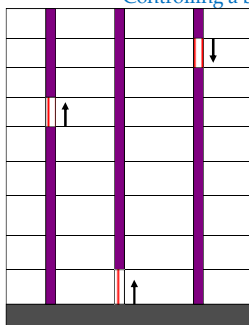
- State of buttons inside elevator (what floor buttons are pressed)

Possible actions:

- Stop
- Go up
- Go down

Example

Controlling a bank of elevators



Constraint: Non-empty elevator must continue in current direction and stop at all floors requested by its passengers before it reverses direction

Control program:

1. At each iteration (e.g., every .5 seconds)
 1. Examine sensor readings and memory contents (“global state” s)
 2. Perform action for each elevator
 3. Receive reward
 - -50 for every floor that has passengers waiting (minimize time spent waiting for elevators)
 - -1 for each elevator that is not stopped (minimize distance travelled by elevators)

Policy: Mapping from space of states to space of actions.
 Could be represented as a look-up table.

Optimal policy: Policy that maximizes expected sum of rewards received over lifetime of system

Formalization

This is a Markov decision processes:

1. Agent L only knows current state and actions available from that state.

Agent L can

1. perceive a set S of distinct states of an environment
2. perform a set A of actions.

Let r be a reward function, unknown to L , such that

$$r : S \times A \rightarrow \mathfrak{R} \quad r(s_t, a_t) = r_t$$

Let δ be a state transition function, unknown to L , such that

$$\delta : S \times A \rightarrow S \quad \delta(s_t, a_t) = s_{t+1}$$

Goal is for L to learn policy $\pi, \pi : S \rightarrow A$

$$\pi(s_t) = a_t$$

such that π maximizes cumulative reward.

Example: Robby the robot

Sensors: N,S,E,W,C(current)
Example policy:

Actions:
 Move N
 Move S
 Move E
 Move W
 Stay put
 Try to pick up can

Rewards/Penalties (points):
 Picks up can: 10
 Tries to pick up can on empty site: -1
 Crashes into wall: -5

0	c					c	c	c		c
1	c		c		c	c				
2	c		c	c	c				c	c
3	c			c		c				c
4		c	c	c			c	c	c	c
5	c				c	c		c		c
6	c		c	c						c
7			c	c	c	c	c	c	c	
8	c	c			c		c			
9	c	c				c				

What is cumulative reward of this policy?

Let $V^\pi(s_t)$ denote the "cumulative value" of π starting from initial state s_t :

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$= \sum_{j=0}^{\infty} \gamma^j r_{t+j}$$

where $0 \leq \gamma < 1$ is a constant that determines the relative value of delayed versus immediate rewards.

$V^\pi(s_t)$ is called the "value function" for policy π

Note that rewards received i times steps into the future are discounted exponentially (by a factor of γ).

If $\gamma = 0$, we only care about immediate reward.

The closer γ is to 1, the more we care about future rewards, relative to immediate reward.

	0	1	2	3	4	5	6	7	8	9
R	C					C	C	C		C
0	C				C	C				
1	C		C		C	C				
2	C		C	C	C				C	C
3	C			C			C			C
4		C	C	C			C	C	C	C
5	C				C	C		C		C
6	C		C	C						C
7			C	C	C					
8	C	C			C					
9	C	C								

$$V^\pi(s_0) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$

Let $\gamma = 0.8$

Precise specification of learning task:

We require that the agent learn policy π that maximizes $V^\pi(s)$ for all states s .

We call such a policy an *optimal policy*, and denote it by π^* :

$$\pi^* = \operatorname{argmax}_\pi V^\pi(s), \forall s$$

To simplify notation, let

$$V^*(s) \equiv V^{\pi^*}(s)$$

Q Learning

Hard to learn $\pi^* : S \rightarrow A$ directly, since we don't have training data of form (s, a) .

Only training data available to learner is sequence of rewards:

$$r(s_0, a_0), r(s_1, a_1), \dots$$

So what should the learner L learn?

Q Learning, continued

One possibility: learn evaluation function $V^*(s)$.

Then L would know what action to take next:

- L should prefer state s_1 over s_2 whenever $V^*(s_1) > V^*(s_2)$
- Optimal action a in state s is the one that maximizes sum of $r(s,a)$ (immediate reward) and V^* of the immediate successor state, discounted by γ :

$$\pi^*(s) = \operatorname{argmax}_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

	0	1	2	3	4	5	6	7	8	9
R	C					C	C	C		C
0	C				C	C				
1	C		C		C	C				
2	C		C	C	C				C	C
3	C			C			C			C
4		C	C	C			C	C	C	C
5	C				C	C		C		C
6	C		C	C						C
7			C	C	C	C	C	C	C	
8	C	C								
9	C	C				C				C

$$\pi^*(s) = \operatorname{argmax}_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

Q Learning, continued

However, using V^* to obtain optimal policy π^* requires perfect knowledge of δ and r , which we earlier said are unknown to L .

Q Learning, continued

Alternative: learn evaluation function $Q: S \times A \rightarrow \mathfrak{R}$:

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

$Q(s, a)$ is the maximum discounted cumulative reward that can be achieved starting from state s and applying action a as the first action.

In other words, $Q(s, a)$ is the immediate reward gained when action a is taken from state s , plus the value of following the optimal policy thereafter.

Q Learning, continued

Note that

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

(Why is this true?)

Q Learning, continued

What's the difference between learning Q and learning V^* ?

Recapping what we have discussed:

$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

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

so,

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

Q Learning, continued

In short, if L can learn Q function, it can find optimal policy without knowledge of δ and r !

Since

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

all L needs to do is consider each available action a in current state s and choose the action that maximizes $Q(s, a)$.

This is similar to how some computer game-playing algorithms work, e.g., chess.

	0	1	2	3	4	5	6	7	8	9
0	c					c	c	c		c
1	c		c		c	c				
2	c		c	c	c				c	c
3	c			c			c			c
4		c	c	c			c	c	c	c
5	c				c	c		c		c
6	c		c	c						c
7			c	c	c	c	c	c	c	
8	c	c								
9	c	c				c				c

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

Q Learning, continued

How does a learner learn this magical Q function?

Recall that

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

Thus we can write $Q(s, a)$ as follows:

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

Now, using this recursive definition, we can estimate $Q(s, a)$ via iterative approximation.

How to learn Q

Let \hat{Q} denote L 's current hypothesis—an estimate for target function Q .

\hat{Q} is represented by table, whose entries are $\hat{Q}(s, a)$

Assume deterministic rewards and actions.

Choose $0 \leq \gamma < 1$.

Q learning algorithm

1. For each (s, a) , initialize $\hat{Q}(s, a)$ to be zero.
2. Observe the current state s .
3. Do forever:
 - Select an action a and execute it.
 - Receive immediate reward r
 - Learn:
 - Observe the new state s'
 - Update the table entry for $\hat{Q}(s, a)$ as follows:

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

- $s \leftarrow s'$

	0	1	2	3	4	5	6	7	8	9
0	C					C	C	C		C
1	C		C		C	C				
2	C		C	C	C				C	C
3	C			C			C			C
4		C	C	C			C	C	C	C
5	C				C	C		C		C
6	C		C	C						C
7			C	C	C	C	C	C	C	
8	C	C								
9	C	C					C			C

Example of Q learning

Convergence theorem for Q-learning:

Let $\hat{Q}_n(s, a)$ denote the agent's hypothesis, $\hat{Q}(s, a)$, after the n th update in the Q-learning algorithm.

If each state-action pair is visited infinitely often, then

$$\hat{Q}(s, a) \text{ converges to } Q(s, a) \text{ as } n \rightarrow \infty.$$

How to choose actions?

Naïve strategy: at each time step, choose action that maximizes $\hat{Q}(s, a)$

This **exploits** current \hat{Q} but doesn't further **explore** the state-action space (in case \hat{Q} is way off).

Also, convergence theorem assumes that, in the limit, each state-action transition occurs infinitely often.

Common in Q learning to use probabilistic approach:

$$P(a_i | s) = \frac{e^{\hat{Q}(s, a_i)/T}}{\sum_j e^{\hat{Q}(s, a_j)/T}}, \quad T > 0$$

This balances exploitation and exploration in a tunable way:

1. high T : more exploration (more random)
2. low T : more exploitation (more deterministic)

Can start with high T , and decrease it as \hat{Q} improves.

Representation of $\hat{Q}(s,a)$

Note that in all of the above discussion, $\hat{Q}(s,a)$ was assumed to be a look-up table, with a distinct table entry for each distinct (s,a) pair.

More commonly, $\hat{Q}(s,a)$ is represented as a function (e.g., as a neural network), and the function is estimated (e.g., through back-propagation).

Recap on Reinforcement Learning

$r : S \times A \rightarrow \mathfrak{R}$ (reward function)

$\delta : S \times A \rightarrow S$ (transition function)

$\pi : S \rightarrow A$ (policy)

$$V^\pi(s) = r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{i+i} \quad (\text{cumulative value of } \pi)$$

$$\pi^* = \operatorname{argmax}_{\pi} V^\pi(s), \quad \forall s \quad (\text{optimal policy})$$

$$V^*(s) \equiv V^{\pi^*}(s) \quad (\text{cumulative value of optimal policy})$$

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) \quad \text{Evaluation function}$$