

Q - Learning

Sargur N. Srihari

srihari@cedar.buffalo.edu

Topics in Q -Learning

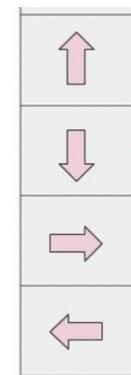
- Overview
- 1. The Q Function
- 2. An algorithm for learning Q
- 3. An illustrative example
- 4. Convergence
- 5. Experimental strategies
- 6. Updating sequence

Task of Reinforcement Learning

States s



Actions a



Current state s_t	$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$
Selected action a_t	
Reward r_t	0
Next state s_{t+1}	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$

$$\delta(s_t, a_t) = s_{t+1}$$
$$r(s_t, a_t) = r_t$$

Task of agent is to learn a policy $\pi: S \rightarrow A$

Agent's Task is to learn π

- The agent has to learn a policy π that maximizes $V^\pi(s)$ for all states s

- Where

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

- We will call such a policy an optimal policy π^* :

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

- We denote the value function $V^{\pi^*}(s)$ by $V^*(s)$
- It gives the maximum discounted cumulative reward that the agent can obtain starting from state s

Role of an Evaluation Function

- How can an agent learn an optimal policy π^* for an arbitrary environment?
- It is difficult to learn function $\pi^* : S \rightarrow A$ directly
 - Because available training data does not provide training examples of the form $\langle s, a \rangle$
 - Instead the only information available is the sequence of immediate rewards $r(s_i, a_i)$ for $i=0,1,2,\dots$
- Easier to learn a numerical evaluation function defined over states and actions
 - And implement optimal policy in term of the evaluation function

Optimal action for a state

- An evaluation function is $V^{\pi^*}(s)$ denoted as $V^*(s)$
 - where

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

$$\pi^* = \arg \max_{\pi} V^\pi(s), (\forall s)$$
 - Agent prefers s_1 over state s_2 whenever $V^*(s_1) > V^*(s_2)$
 - i.e., cumulative feature reward is greater from s_1
- Optimal action in state s is the action a
 - One that maximizes the sum of the immediate reward $r(s, a)$ plus the value V^* of the immediate successor state discounted by γ

$$\pi^*(s) = \arg \max_{\pi} [r(s, a) + \gamma V^*(\delta(s, a))]$$
 - Where $\delta(s, a)$ is state resulting from applying action a to s

Perfect knowledge of δ and r

- Agent can acquire optimal policy by learning V^* provided it has perfect knowledge of
 - Immediate reward function r , and
 - State transition function δ
 - It can then use equation
$$\pi^*(s) = \arg \max_{\pi} [r(s, a) + \gamma V^*(\delta(s, a))]$$
 - To calculate the optimal action for any state
- But it may be impossible to predict the outcome of an arbitrary action to an arbitrary state
 - E.g., robot shoveling dirt when the resulting state included positions of dirt particles

Definition of Q -function

- Instead of using

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

define

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

and rewrite $\pi^*(s)$ as

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

which is the optimal action for state s

- This rewrite is important because

- It shows that if the agent learns the Q function instead of the V^* function, it will be able to select optimal actions even when it has no knowledge of functions $r(s, a)$ and $\delta(s, a)$
- It need only consider each action a in its current state s and choose action that maximizes $Q(s, a)$

Example of Q -function and Q -learning

Grid world



Current state (s): $\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$

a

Selected action (a):

Reward (r): 0

Next state (s'): $\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$

max $Q(s')$: 1.0

$Q(s, a)$	S						$\gamma = 0.95$
	$0\ 0\ 0$ $1\ 0\ 0$	$0\ 0\ 0$ $0\ 1\ 0$	$0\ 0\ 0$ $0\ 0\ 1$	$1\ 0\ 0$ $0\ 0\ 0$	$0\ 1\ 0$ $0\ 0\ 0$	$0\ 0\ 1$ $0\ 0\ 0$	
	0.2	0.3	1.0	-0.22	-0.3	0.0	
	-0.5	-0.4	-0.2	-0.04	-0.02	0.0	
	0.21	0.4	0.4	-0.3	0.5	1.0	0.0
	-0.6	-0.1	-0.1	-0.31	-0.01	0.0	

	$0\ 0\ 0$ $1\ 0\ 0$	$0\ 0\ 0$ $0\ 1\ 0$	$0\ 0\ 0$ $0\ 0\ 1$	$1\ 0\ 0$ $0\ 0\ 0$	$0\ 1\ 0$ $0\ 0\ 0$	$0\ 0\ 1$ $0\ 0\ 0$
	0.2	0.3	1.0	-0.22	-0.3	0.0
	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
	0.21	0.4	0.4	-0.3	0.5	1.0
	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

	$0\ 0\ 0$ $1\ 0\ 0$	$0\ 0\ 0$ $0\ 1\ 0$	$0\ 0\ 0$ $0\ 0\ 1$	$1\ 0\ 0$ $0\ 0\ 0$	$0\ 1\ 0$ $0\ 0\ 0$	$0\ 0\ 1$ $0\ 0\ 0$
	0.2	0.3	1.0	-0.22	-0.3	0.0
	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
	0.21	0.4	0.4	-0.3	0.5	1.0
	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

	$0\ 0\ 0$ $1\ 0\ 0$	$0\ 0\ 0$ $0\ 1\ 0$	$0\ 0\ 0$ $0\ 0\ 1$	$1\ 0\ 0$ $0\ 0\ 0$	$0\ 1\ 0$ $0\ 0\ 0$	$0\ 0\ 1$ $0\ 0\ 0$
	0.2	0.3	1.0	-0.22	-0.3	0.0
	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
	0.21	0.4	0.95	-0.3	0.5	1.0
	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

$$\text{New } Q(s, a) = r + \gamma * \max Q(s') = 0 + 0.95 * 1 = 0.95$$

Q - Function Property

- Surprising that one can choose globally optimal action sequences by reacting repeatedly to the local values of Q for the current state
 - Without conducting a lookahead search to explicitly consider what state results from what action
- Value of Q for the current state and action summarizes in a single number all information needed to determine discounted cumulative reward

An algorithm for learning Q

- Learning the Q function corresponds to learning the optimal policy
- How can Q be learned?
- Key problem: finding a reliable way to estimate training values for Q given only a sequence of immediate rewards r spread out over time
- This can be accomplished through iterative approximation

Recursive definition of Q

- Notice the close relationship between Q and V^*

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

- Which allows rewriting
$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$
 as

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

- Bellman's equation
- This recursive definition of Q provides the basis of algorithms that iteratively approximate Q

Table of state-action pairs

- Symbol \hat{Q} refers to the learner's estimate of the actual Q
- Learner represents its hypothesis \hat{Q} by a large table with a separate entry for each state-action pair
- Table entry for pair (s, a) stores value for $\hat{Q}(s, a)$
 - The learner's current hypothesis about the actual but unknown value $Q(s, a)$
 - Table is initially filled with random values
 - Easier to understand algorithm with initial values of zero

Update rule for table entries

- Agent repeatedly observes its current state $t s$, chooses some action a
- Executes this action, then observes the reward $r = r(s, a)$ and the new state $s' = \delta(s, a)$
- It then updates the table entry for following each such transition, according to the rule

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

Example of updating Q-Learning table

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0

	327	0	0	0	0	0	0

	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0

	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017

	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Q-Learning table of states by actions that is initialized to zero, then each cell is updated through training.

Q -learning algorithm for deterministic MDP

- For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero
- Observe the 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

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

– $s \leftarrow s'$

Q-Learning for Atari Breakout

<https://www.youtube.com/watch?v=V1eYniJ0Rnk&vl=en>

Example of Q -Learning

$r(s, a)$ immediate reward values

- Agent R in top left cell (state s_1) performs a_{right} and receives:

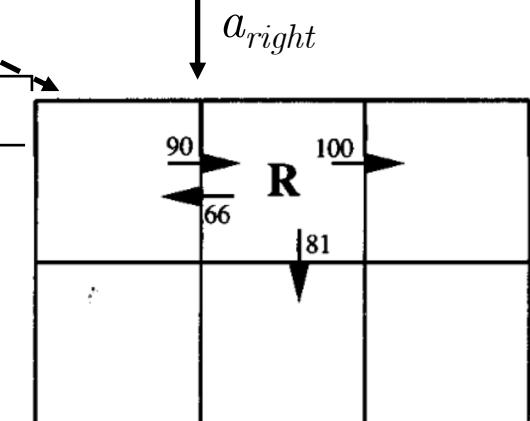
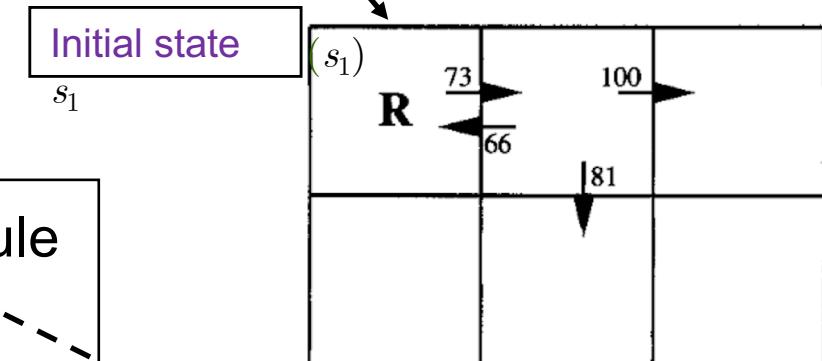
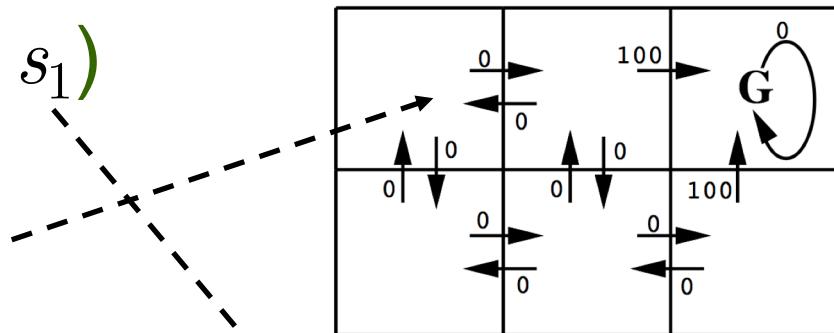
1. Immediate reward, $r(s, a) = 0$
2. Total reward for a_{right} :

$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max\{66, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

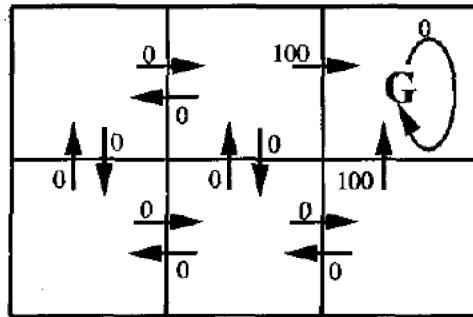
Obtained by applying training rule
 $\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$
 to refine its estimate of \hat{Q}

Algorithm

Each time the agent moves from an old state to a new one, Q -learning propagates \hat{Q} estimates backward from the new state to the old.
 At the same time the immediate reward received for the transition is used to augment the propagated values of \hat{Q}



Episodes and \hat{Q} evolution



Immediate reward function definition:
Zero everywhere except when entering the goal state
Episodes:
Since goal state is an absorbing state,
training consists of a series of episodes

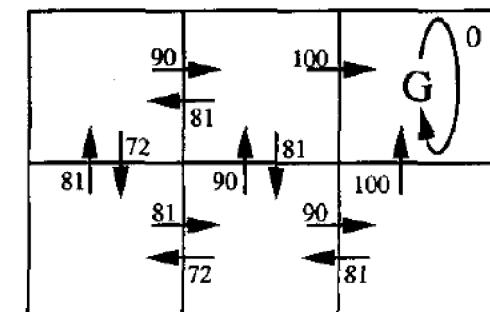
During each episode, agent starts at a random state and executes actions until it reaches the absorbing state

Evolution of \hat{Q} values

Since all \hat{Q} values are initialized to 0, agent makes no changes until it reaches G and it receives a nonzero reward.

This results in refining the \hat{Q} value for the single transition leading to G.

If on the next episode agent passes through a state next to this state, its nonzero value refines a state two steps from G. Eventually results in \hat{Q} table containing the \hat{Q} values shown:



Properties of Q -Learning Algorithm

Algorithm

- Initialize all table entries $\hat{Q}(s, a)$ to zero
- Observe the 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

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

For a deterministic MDP with non-negative rewards
Assuming all values are initialized to 0

After the n^{th} iteration of the training procedure

$(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$ i.e., \hat{Q} values never decrease during training

$0 \leq \hat{Q}_n(s, a) < Q(s, a)$ i.e., during training every \hat{Q} value remains between 0 and true value of Q

Convergence of Algorithm

- Q -Learning algorithm converges towards a \hat{Q} equal to Q under the following conditions:
 1. Deterministic MDP
 2. Immediate reward values are bounded $r(s, a) < c$
 3. Agent visits every possible state-action pair infinitely
- Q-Learning convergence theorem is formally stated next

Q -Learning Convergence Theorem

- *Theorem:* Convergence of Q learning for deterministic MDP
- Key idea in proof:
 - The table entry $\hat{Q}(s, a)$ with the largest error must have its error reduced by a factor γ whenever it is updated
 - The reason that its new value depends only in part on error-prone \hat{Q} estimates, with the remainder depending on the error-free observed immediate reward r

Experimentation Strategies

- Q -learning algorithm doesn't specify how actions are chosen by the agent
- One obvious strategy:
 - Select a that maximizes $\hat{Q}(s, a)$ thereby exploiting the current approximation \hat{Q}
 - However risks overcommitting to actions found during early training to have high \hat{Q} values while failing to explore actions with even higher values
- Q-learning convergence requires that each state-action transition occurs infinitely often
 - Thus Q-learning uses a probabilistic approach

Exploitation and Exploration

- Probability of action a_i is

$$P(a_i|s) = \frac{k^{\hat{Q}(s, a_i)}}{\sum_j k^{\hat{Q}(s, a_j)}}$$

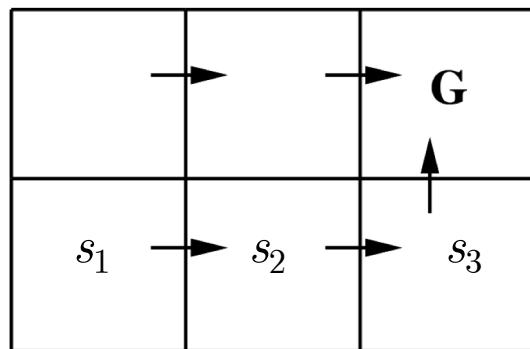
- Larger values of k will assign higher probabilities to actions with above average \hat{Q}
 - causing the agent to **exploit** what it has learned and seek actions it believes will maximize its reward
- Small values of k will allow higher probabilities for other actions, leading the agent
 - to **explore** actions that do not currently have high \hat{Q} values.

From exploitation to exploration

- k is varied with the number of iterations
 - So that the agent favors exploration during early stages of learning
 - Then gradually shifts toward a strategy of exploitation

Action sequence for training

- Implication of convergence theorem is that:
 - Need not train on optimal action sequences $(s_1, a_1), (s_2, a_2) \dots$ for converging to optimal policy



- Randomly chosen actions suffice
 - Possible to learn the Q -function (and hence the optimal policy) while training from actions chosen completely at random at each step
 - As long as resulting training sequence visits every (s, a) transition infinitely often

Action Sequences for Efficiency

- Since any action sequence suffices
 - We can change sequences of training example transitions
 - to improve training efficiency without endangering final convergence
- How to determine such sequences?

Ex: Reversed action sequence

- Begin with all \hat{Q} initialized to zero
 - After 1st episode only one entry of \hat{Q} will change
 - corresponding to final transition into goal state
 - 2nd episode: same sequence from same start
 - then a second table entry would be made nonzero
 - Repeated identical episodes:
 - frontier of nonzero \hat{Q} values will creep backward
 - From goal state: one state-action transition per episode
- Now perform same updates in reverse order
 - After 1st episode, update rule will have updated \hat{Q} for every transition it took to the goal ⇒
 - Converges in fewer iterations (but requires memory to store entire episode)

Second strategy: replay old transitions

- Store past state-action transitions, along with the immediate reward received, and retrain on them periodically
 - Might seem a waste to retrain on same transition
 - But recall that the updated $\hat{Q}(s, a)$ value is determined by the values $\hat{Q}(s', a)$ of successor state $s' = \delta(s, a)$
 - If subsequent training changes one of the $\hat{Q}(s', a)$ values then retraining on the transition $\langle s, a \rangle$ may result in an altered value for $\hat{Q}(s, a)$
 - Extent of replay depends on tradeoff of cost in replay versus collecting new info from world