Machine Learning

Reinforcement Learning

Reference: Reinforcement learning An Introduction
By Richard S. Sutton and Andrew G. Barto

Artificial Intelligence & Computer Vision Lab
School of Computer Science and Engineering
Seoul National University

Overview (from reference)

- Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal (value). The agent (learner) is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them.
- The agent and the environment interact continually: the agent selecting actions by sensing the state of the environment (*exploration & expoitation*) and the environment responding to those actions and presenting new situations to the agent. The environment also gives rewards, special numerical values that the agent tries to maximize over time.
- A *task* is a complete specification of an environment, one instance of the reinforcement learning problem: If the agent-environment interaction breaks into subsequences, called episodes, such as plays of a game, trips through maze, or any sort of repeated interactions, then the task with episodes of this kind is called a *episodic task*. If it does not break naturally into identifiable episodes but goes on continually without limit, it is called a *continuing task*.

Overview (cont.)

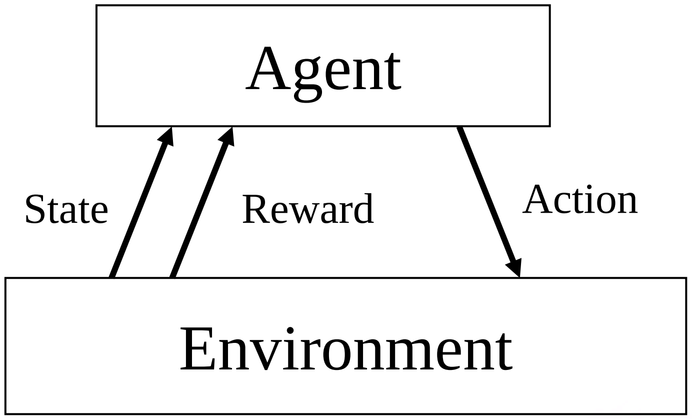
- The agent implements a mapping (policy) from states (situations) to actions so as to maximize the total amount it receives, not the immediate reward but the cumulative reward over time.
- The goal of the reinforcement learning methods is to find the optimal policy maximizing the cumulative reward starting from any given state (situation).
- *Markov property* for reinforcement learning problem: When considered how a general environment might respond at time t +1 to the action at time t, in most general and casual cases this response depends on everything that has happened earlier. If the state and reward response at t +1 depends on only on state and action happened at time t, then the environment is said to have the *Markov property*.
- A reinforcement learning task satisfying the *Markov property* is called a *Markov decision process*. If the state and action spaces are finite, then it is called a *finite Markov decision process*.

Preview (from text)

- Reinforcement learning
  - How an **autonomous agent** that senses and acts in its environment can learn to **choose optimal actions** to achieve its goals?
  - Learn from **indirect, delayed reward** to choose sequences of actions that produce the greatest cumulative reward.
  - ***Q* learning** that can acquire optimal control strategies from delayed rewards, even when the agent has no prior knowledge of the effects of its actions on the environment.
    - Related to **dynamic programming** algorithm

Introduction

- *Agent*
  - Has a set of sensors to observe the *state* of its environment, and a set of *actions* it can perform to alter its state



$$\begin{tikzcd} s_0 \arrow[r, "a_0", "r_0"'] & s_1 \arrow[r, "a_1", "r_1"'] & s_2 \arrow[r, "a_2", "r_2"'] & \cdots\cdots \end{tikzcd}$$

Goal: Learn to choose actions that maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$

where $0 \le \gamma < 1$

Introduction (cont.)

- Agent's learning task
  - At each discrete time step *t*, the agent senses the current state s_t, chooses a current action a_t, and performs it. The environment responds by giving the agent a reward and by producing the succeeding state. The succeeding state functions δ and the reward function *r*, *which are either nondeterministic or deterministic*, are part of the environment and are not necessarily known to the agent.
  - *In an MDP (Markov Decision Process), the functions $\delta(s_t,a_t)$ and $r(s_t,a_t)$, as defined here, depend only on the current state and action, and not earlier states, actions, and rewards.*
  - Agent performs sequences of actions, observes their consequences, and learns a *control policy* $\pi : S \rightarrow A$ to choose actions that *maximize the reward* accumulated over time.

Introduction (cont.)

- Several characteristics of reinforcement learning
  - ***Delayed reward***
    - Training example has the form of *a sequence of immediate reward values* as the agent executes its sequence of actions rather than the form of $\langle s, \pi(s) \rangle$: *Temporal credit assignment problem* determining which of the actions in its sequence are to be credited with producing the eventual rewards.
  - ***Exploration & Exploitation***
    - The agent influences the distribution of training examples by the action sequence it chooses.
    - *Exploration* of unknown states and actions (to gather new information)
    - *Exploitation* of states and actions that it has already learned will yield high reward (to maximize its cumulative reward)

Introduction (cont.)

– ***Partially observed states***

- In many practical situations agent's sensors cannot perceive the entire state of the environment at each time step but provide only partial information. When agent chooses actions, thus, it needs to consider its previous observations together with its current sensors.

– ***Life-long learning***

- Robot learning often requires that robot learns several related tasks within same environment using same sensors: A mobile robot need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers. This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

Learning Task

- Problem based on Markov Decision Process (MDP)
  - Assume a finite set S of states, a finite set A of actions, and the deterministic functions of δ and *r*.
- Task of the agent
  - Learn a policy, $\pi : S \rightarrow A$, $\pi(s_t) = a_t$, that maximize the cumulative reward.

  - Cumulative reward value following an arbitrary policy π from an arbitrary initial state s_t :

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

    - $V^{\pi}(s)$: discounted cumulative reward
    - $0 \leq \gamma < 1$: discounted factor for future rewards.

Learning Task (cont.)

- Other definition

finite horizon reward

$$\sum_{i=0}^{h} r_{t+i}$$

average reward

$$\lim_{h \to \infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$$

- Optimal policy
  - Policy π that maximizes $V^{\pi}(s)$ for all states

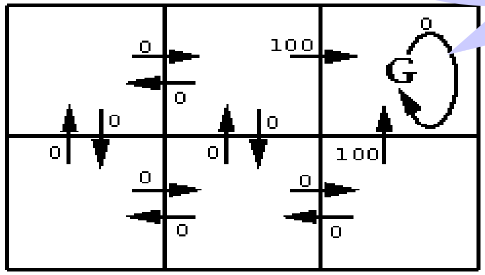
$$\pi^{*} \equiv \underset{\pi}{\operatorname{argmax}} V^{\pi}(s), (\forall s)$$

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

Learning Task (cont.)

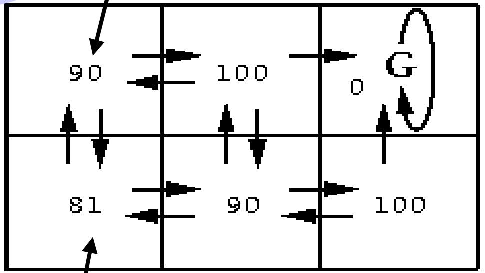
– Simple grid-world environment $\gamma = 0.9$

G : goal state,
an absorbing state



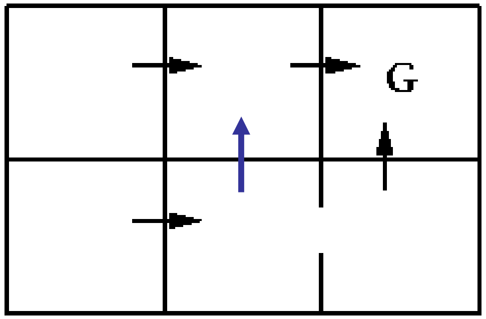
$r(s,a)$ (immediate reward) values

$0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \cdots = 90$



$V^*(s)$ values

$0 + \gamma 0 + \gamma^2 100 + \gamma^3 0 + \cdots = 81$



One optimal policy

Q Learning

- How can an agent learn an optimal policy in arbitrary environment?
  - difficult to learn the function $\pi^* : S \rightarrow A$ directly
    - The training examples of the form *(s,a)* is not given but the sequence of immediate rewards $r(s_i, a_i)$ for $i = 0,1,2\ldots$ is available.
    - Learn a *numerical evaluation function* defined over *states and actions* and implement the optimal policy in terms of this evaluation function.
- What evaluation function should the agent attempt to learn?
  - The optimal action in state s,

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

  - Agent can acquire the optimal policy by learning *V**, *provided it has prefect knowledge of the immediate the reward function* r *and the state transition* δ, which is impossible in many practical robot control problems.

$\Longrightarrow$ **Evaluation function *Q***

Q Function

- ***Q function***
  - Maximum discounted cumulative reward that can be achieved starting from *s* and applying *a* as the first action

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

  - Optimal action *a* in *s*, $\pi^*(s) = \arg\max_a Q(s,a)$
  - If the agent learn the *Q* function,
    - it can choose optimal actions *even when it has no knowledge of the functions* r *and* δ
    - it can choose optimal action without ever conducting a look ahead search
      - get optimal policy by selecting actions with maximal *Q* values

Algorithm for Learning *Q*

- Iterative approximation
  - Relationship between *Q* and *V**: $V^*(s) = \max_{a'} Q(s, a')$
  - Recursive definition of *Q*:

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

  - $\hat{Q}$: learner's estimate (hypothesis) to *Q*.
    - Represent hypothesis $\hat{Q}$ by large lookup table (*Q* table) for each state-action pair
    - Table entry with the pair <*s*, *a*> stores $\hat{Q}(s,a)$

- Training rule

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

  - Learn by observing the resulting new state s' and reward *r*

Algorithm for Learning *Q* (cont.)

- ***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

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

    - $s \leftarrow s'$

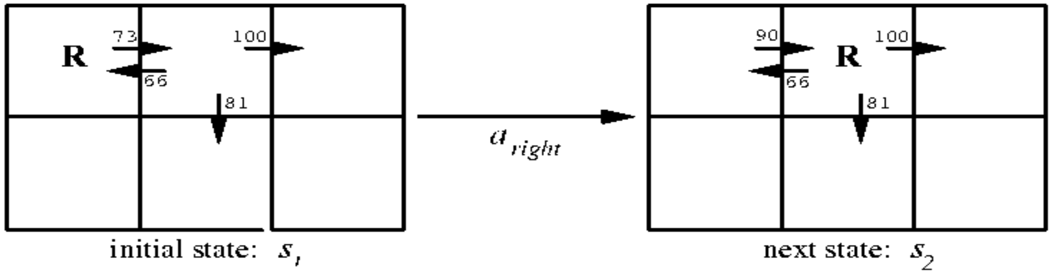
deterministic Markov decision process

r is bounded

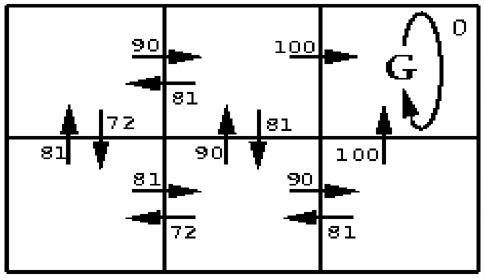
every state-action pair is visited infinitely often

Illustrative Example

- Assume that training consists of a series of episodes, where episode is a sequence of interactions between agent and environment:



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



$Q(s, a)$ values

Illustrative Example (cont.)

- During each episode, the agent begins at some randomly chosen initial state and execute actions until it reaches the goal state
- When it reaches the goal state, the episode ends and the agent is transported to a new, randomly chosen, initial state for the next episode.

- Two general properties of *Q* learning algorithm
  - $\hat{Q}$ values never decrease during training.

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

  - Every $\hat{Q}$ value will remain in the interval [0, true *Q*].

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

- Deterministic MDP.
- Initialize all $\hat{Q}$ values to zero.
- Non-negative reward.

Convergence

- The $\hat{Q}$ converge to the true Q!
  - Assumption
    - System is a deterministic MDP.
    - The immediate reward values are bounded. $(\forall s, a) \quad |r(s,a)| \le c$
    - The agent selects actions in such a fashion that it visits every possible state-action pair infinitely often.
  - The key idea
    - The table entry $\hat{Q}(s,a)$ with the largest error must have its error reduced by a factor γ whenever it is updated.

$\hat{Q}_n(s,a)$ converges to $Q(s,a)$ as $n \rightarrow \infty$, for all *s*, *a*

Convergence (cont.)

- Proof:
  - Let $\hat{Q}_n$ be the table after n updates, and Δ_n be the maximum error in $\hat{Q}_n$; that is

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

  - 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

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

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

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

$$| \hat{Q}_{n+1}(s,a) - Q(s,a) | \leq \gamma^n \Delta_0, \quad \Delta_n \rightarrow 0 \text{ as } n \rightarrow \infty$$

Experimental Strategies

- How actions are chosen by the agent?
  - Selection the action a that maximizes $\hat{Q}(s,a)$ (exploitation)
    - The risk to overcommit to actions that are found during early training to have high $\hat{Q}$ values
    - failing to explore other actions that have even higher values.
  - Use a probabilistic approach

$$P(a_i \mid s) = \frac{k^{\hat{Q}(s,a_i)}}{\sum_j k^{\hat{Q}(s,a_j)}} \qquad k > 0$$

    - Actions with higher *Q* values are assigned higher probabilities
    - Large *k* → exploit, Small *k* → explore (*k*<*1*, *k*=*1*, *1*<*k*)
    - *k* is varied with the number of iterations (exploration → exploitation)

Updating Sequence

- Strategy for improving the rate of convergence (training efficiency)
  - Train the identical episode in reverse chronological order
    - Converge in fewer iterations, but requires more memory

  - Store past state-action transitions along with the immediate reward
    - If $\hat{Q}(s,a)$ is determined by $\hat{Q}(s',a)$ of the successor state $s' = \delta(s,a)$ and subsequent training changes $\hat{Q}(s',a)$, retaining on the transition <*s*,*a*> may result in the altered value for $\hat{Q}(s,a)$

- If $r(s,a)$ and $\delta(s,a)$ are known, many more efficient methods possible (simulation)

Nondeterministic Reward and Actions

- Nondeterministic case
  - Reward function $r(s,a)$ and transition function $\delta(s,a)$: probabilistic outcomes

- Nondeterministic Markov decision process
  - The probabilistic distributions of $r(s,a)$ and $\delta(s,a)$ depend solely on the current state and action
  - *Expected value* of the discounted cumulative reward

$$V^{\pi}(s_t) \equiv E\left[\sum_{i=0}^{\infty} \gamma^{i} r_{i+1}\right]$$

  - Covered the deterministic case

Nondeterministic Reward and Actions (cont.)

- Redefine of *Q*-value: taking expected value of *Q*

$$\begin{aligned} Q(s,a) &\equiv E\left[r(s,a) + \gamma V^*(\delta(s,a))\right] \\ &= E[r(s,a)] + \gamma E\left[V^*(\delta(s,a))\right] \\ &= E[r(s,a)] + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \end{aligned}$$

$$Q(s,a) = E[r(s,a)] + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} (s', a')$$

- Training rule

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

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

  - $visits_n(s,a)$: the number of times this state-action pair has been visited up to and including the *n*th iteration

Nondeterministic Reward and Actions (cont.)

- Revisions to $\hat{Q}$ are made more gradually than deterministic case $\hat{Q}$

- By reducing α at some rate during training, convergence to correct Q function is achieved
  - Convergence of Q learning for nondeterministic MDP
    - bounded reward, initialize to arbitrary finite value,

If $0 \le \alpha_n < 1$ and $\sum_{i=1}^{\infty} \alpha_{n(i,\,s,\,a)} = \infty, \quad \sum_{i=1}^{\infty} \left[\alpha_{n(i,s,a)}\right]^2 < \infty$ then $\hat{Q}(s,a)$ converges to $Q(s,a)$ as $n \rightarrow \infty$, for all s, a with probability 1.

Temporal Difference Learning

- *Q*-learning
  - Learns by iteratively reducing the discrepancy between *Q* value estimates for adjacent states
  - A special case of general class of *temporal difference algorithms*
    - learn by reducing discrepancies between estimates made by agent at different times
  - *Q* learning training rule :

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

$$\vdots$$

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

- TD(λ) by Sutton(1988)

$$Q^{\lambda}(s_t, a_t) = r_t + \gamma[(1-\lambda) \max_a \hat{Q}(s_t, a_t) + \lambda Q^{\lambda}(s_{t+1}, a_{t+1})] \quad 0 \le \lambda \le 1$$

Temporal Difference Learning (cont.)

– If $\lambda = 0$,
  - Considers only one-step discrepancies in the $\hat{Q}$ estimate

As λ increases, the algorithm emphasizes on discrepancies based on more distance lookaheads

– If $\lambda = 1$,
  - Only the observed r_{i+1} values are considered, with no contribution from the current $\hat{Q}$ estimate

Motivation of TD(λ)

- In some settings, training will be more efficient if more distant lookaheads are considered

Generalizing from Examples

- Most constraining assumption of *Q* learning
  - » The target function is represented as an ***explicit lookup table***, with a distinct table entry for every distinct input value (state-action pair)

  - – A kind of rote learning and make **no attempt** to estimate the *Q* value for **unseen state-action pairs** by generalizing from those that have been seen

Generalizing from Examples (cont.)

- Practical systems
  - Incorporate *function approximation methods* (BP) into the Q learning rule, by *substituting a neural network for the lookup table* and using each $\hat{Q}(s,a)$ update as a training examples :
    - Using the encoded state and action as input the network is trained to output the target values of $\hat{Q}$ given by training rules (13. 7) and (13,10)

Relationship to Dynamic Programming

- Reinforcement learning (*Q* learning) are *closely related* with dynamic programming to solve MDP.
  - DP: Perfect knowledge of $r(s,a)$ and $\delta(s,a)$
    - Find method using less computational effect

- Bellman's equation
  - The foundation for many dynamic programming

$$(\forall s \in S)\, V^*(s) = E\left[r(s,\pi(s)) + \gamma V^*(\delta(s,\pi(s)))\right]$$

    - Note that Bellman showed that the optimal policy π^* satisfies the above equation, and any policy π satisfying this equation is an optimal policy.

- *Dynamic Programming*: It refers to a collection of algorithms that can be used to compute optimal policies, given a perfect model of the environment as a *Markov decision process*. Classical *DP* algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense, but still important theoretically.
- *Monte Carlo Methods*: Not assuming complete knowledge of the environment, they require only experience – sample sequences of states, actions, and rewards from online or simulated interaction with an environment. *Monte Carlo* methods are the ways of solving the reinforcement learning problem based on averaging sample returns. They are defined only for episodic tasks.
- *Temporal-Difference Learning Methods* (Combination of *Monte Carlo* and *Dynamic programming*): Like *Monte Carlo* methods, *TD* methods can learn directly from raw experience without a model of the environment's dynamics. Like *DP*, *TD* update estimates based on in part on other learned estimates without waiting for a final outcome as in *MC* methods.