CS365: Deep Learning

Deep Reinforcement Learning

Arijit Mondal
Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
arijit@iitp.ac.in

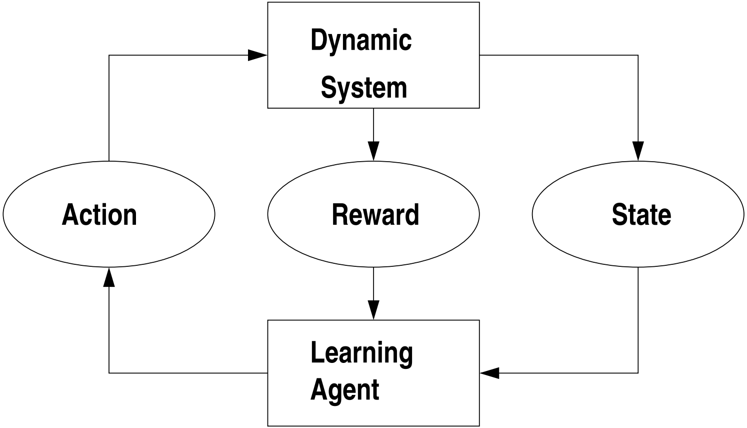
Multi Armed Bandit

- Given k slot machines, an action is to pull an arm of one of the machines
- At each time step t the agent chooses and action a_t among the k actions and receives reward r_t
- Taking action a is pulling arm i which gives reward $r(a)$ with probability p_i
- Goal is to maximize the total expected return
- Expected reward for action a is $Q(a) = \mathbb{E}[r_t | a_t = a]$
- We can estimate the value of $Q_t(a)$ of action a at time t
  - For example, mean reward for each action

Multi Armed Bandit

- Given k slot machines, an action is to pull an arm of one of the machines
- At each time step t the agent chooses and action a_t among the k actions and receives reward r_t
- Taking action a is pulling arm i which gives reward $r(a)$ with probability p_i
- Goal is to maximize the total expected return
- Expected reward for action a is $Q(a) = \mathbb{E}[r_t | a_t = a]$
- We can estimate the value of $Q_t(a)$ of action a at time t
  - For example, mean reward for each action
- A greedy takes the best estimate at time t, exploiting knowledge $a_t = \arg\max_a Q_t(a)$, choosing the action with the largest mean reward

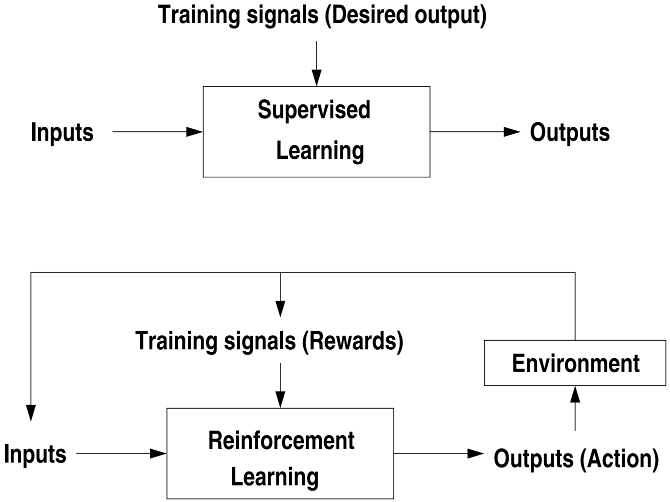
Interaction with environment



Reinforcement learning

- Set of actions that the learner will make in order to maximize its profit
- Action may not only affect the next situation but also subsequent situation
  - Trial and error search
  - Delayed reward
- A learning agent is interacting with environment to achieve a goal
- Agent needs to have idea of state so that it can take right action
- Three key aspects — **observation, action, goal**

Reinforcement vs supervised learning



Reinforcement learning

- It is different from supervised learning
  - Learning from examples provided by a knowledgeable external supervisor
  - Not adequate for learning from interaction
- In interaction problem it is often impractical to obtain examples of desired behavior that are correct and representative of all situations
- Trade-off between *exploration* and *exploitation*
  - To improve reward it must prefer effective action from the past (exploit)
  - To discover such action it has to try unselected actions (explore)
  - Exploit and exploration cannot be pursued exclusively
- Agent interacts with uncertain environment

When to use RL

- Data in the form of trajectories
- Need to make a sequence of decision
- Observe (partial, noisy) feedback to state or choice of action

Examples

- Chess player eg. games
- Robotics
- Adaptive controller
- All involve interaction between active decision making agent and its environment

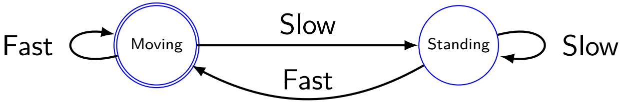
ϵ-Greedy Approach

- A non-greedy action is explored
- We can choose greedily most of the time, sometime non-greedily
- For example, with small probability ϵ we choose greedily and $(1-\epsilon)$ probability non-greedily
- Exploration vs Exploitation

- For each action a do: $Q(a) = 0$, $N(a) = 0$ number of times action is chosen
- For each time step do:
  - $a = \begin{cases} \arg\max_a Q(a) & \text{with probability } (1-\epsilon) \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
  - $N(a) = N(a) + 1$
  - $Q(a) = Q(a) + (r(a) - Q(a))/N(a)$

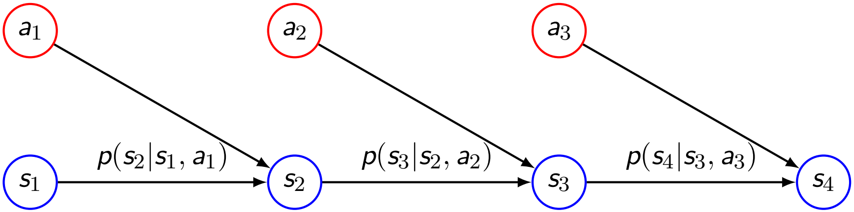
State machines

- S - set of possible state
- X - set of possible inputs
- A transition function $f : S \times X \to S$
- Y - set of possible outputs
- A mapping $g : S \to Y$



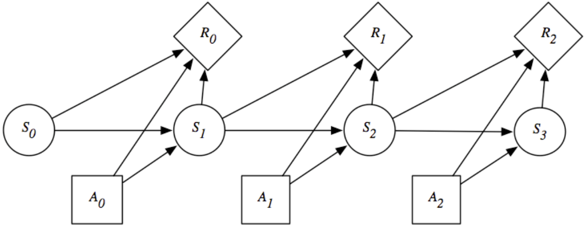
Markov process

- In multi-armed bandit, actions were stateless
- In many scenarios action will depend on past states
- Also the transition from one state to another state can have uncertainty
- Markov decision process assumes that probability of a state s_{t+1} depends only on s_t and a_t, not on any other previous states or actions



Markovian decision process

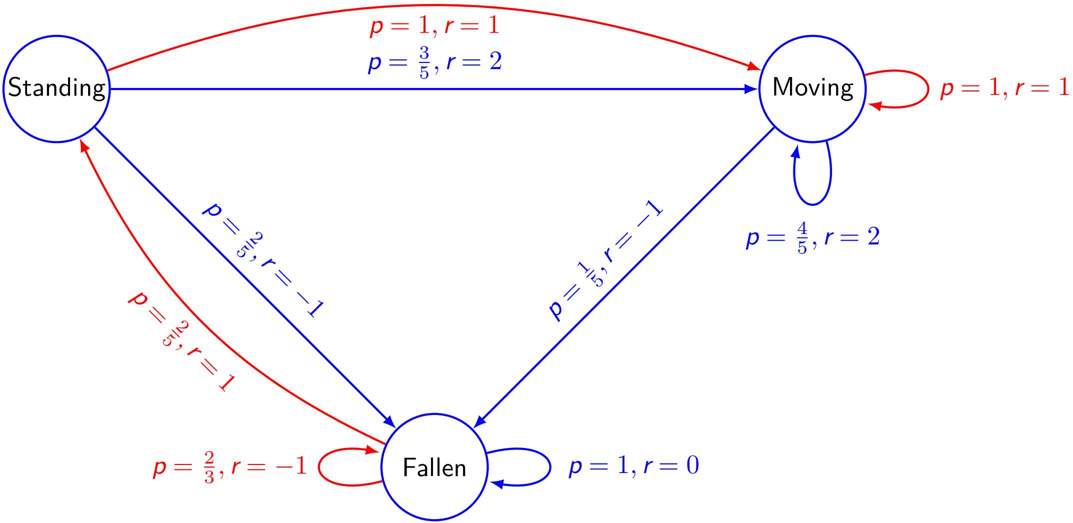
- S — set of states
- A — set of actions
- $Pr(s_t|s_{t-1}, a_{t-1})$ — Probabilistic effects
- r_t — reward function
- μ_t — initial state distribution



- Markov property: The future state depends only on the current state

$$Pr(s_t|s_{t-1}, \ldots, s_0) = Pr(s_t|s_{t-1})$$

Markov process: Example



Policy

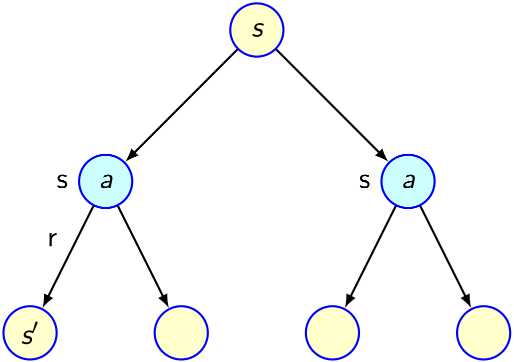
- Policy: $\pi : S \rightarrow A$ - a mapping from state to action
- For every state we need to choose an action
- Example:
  - π_A - always take the slow (red) action
  - π_B - always take the fast (blue) action
  - π_C - if fallen take slow action, fast otherwise
  - π_D - if moving take fast action, slow otherwise
- A policy need not be deterministic - π_E - all states take slow action with probability 0.3 and fast action with probability 0.7
- It may be viewed as rule book

Reward

- Each action is associated with some reward
- Return is the sum of discounted rewards $g_t = r_{t+1} + \gamma r_{t+2} + \ldots + r_T = r_{t+1} + \gamma g_{t+1}$

State action diagram

- The agent starts from a root node s, takes action a
- Action a is chosen by some policy
- State-action diagram represents an episode (s, a, r, s')



Elements of RL

- Agent
- Environment
- Policy — The way agent behaves at a given time
  - Mapping of state-action pair to state
  - Can use look up table or search method
  - Core of reinforcement learning problem
- Reward function — Defines the goal in reinforcement learning problem
  - It maps state-action pair to a single number
  - Objective of RL agent is to maximize total reward
  - Defines bad or good events
  - Must be unalterable by agent, however policy can be changed

Elements of RL (contd.)

- Value function
  - Specifies what is good in long run
  - Value of a state is the total amount of reward an agent can expect to accumulate over future starting from the state
  - Indicates long term desirability of states
  - The action tries to move to a state of highest value (not highest reward)
  - Rewards are mostly given by the environment
  - Value must be estimated or reestimated from the sequence of observation
  - Need efficient method to find values
    - Evolutionary methods (genetic algorithm, simulated annealing) search directly in the space of policies without applying value function

Elements of RL (contd.)

- Model of environment
  - Mimics the behavior of environment
  - Given state and action, model might predict resultant next state and next reward
  - Every RL system uses trial and search methodology to learn

State value function

- State Value Function - what is the value of a policy?
- The agent is allowed to make actions and collects rewards
- $V_\pi^h(s)$ - state value function wrt to π with h horizon. $V_\pi^0 = 0$

State value function

- State Value Function - what is the value of a policy?
- The agent is allowed to make actions and collects rewards
- $V_\pi^h(s)$ - state value function wrt to π with h horizon. $V_\pi^0 = 0$
- This can be computed using induction
  - For $h = 1$, $V_\pi^1(s) = R(s, \pi(s)) + V_\pi^0(s) = R(s, a) + 0$

State value function

- State Value Function - what is the value of a policy?
- The agent is allowed to make actions and collects rewards
- $V_\pi^h(s)$ - state value function wrt to π with h horizon. $V_\pi^0 = 0$
- This can be computed using induction
  - For $h = 1$, $V_\pi^1(s) = R(s, \pi(s)) + V_\pi^0(s) = R(s, a) + 0$
  - For $h = 2$, $V_\pi^2(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s')R(s', \pi(s'))$

State value function

- State Value Function - what is the value of a policy?
- The agent is allowed to make actions and collects rewards
- $V_\pi^h(s)$ - state value function wrt to π with h horizon. $V_\pi^0 = 0$
- This can be computed using induction
  - For $h = 1$, $V_\pi^1(s) = R(s, \pi(s)) + V_\pi^0(s) = R(s, a) + 0$
  - For $h = 2$, $V_\pi^2(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s')R(s', \pi(s'))$
  - For any h, $V_\pi^h(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s')V_\pi^{h-1}(s')$

State value function

- State Value Function - what is the value of a policy?
- The agent is allowed to make actions and collects rewards
- $V_\pi^h(s)$ - state value function wrt to π with h horizon. $V_\pi^0 = 0$
- This can be computed using induction
  - For $h = 1$, $V_\pi^1(s) = R(s, \pi(s)) + V_\pi^0(s) = R(s, a) + 0$
  - For $h = 2$, $V_\pi^2(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s') R(s', \pi(s'))$
  - For any h, $V_\pi^h(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s') V_\pi^{h-1}(s')$
- Goal is to compute $V_\pi(s) = \mathbb{E}_\pi[g_t | s_t = s] = \mathbb{E}_\pi[\sum_k \gamma^k r_{t+k+1} | s_t = s]$
- γ - discount factor. $\gamma = 0$ is myopic, 1 means farsighted

State value function

- State Value Function - what is the value of a policy?
- The agent is allowed to make actions and collects rewards
- $V_\pi^h(s)$ - state value function wrt to π with h horizon. $V_\pi^0 = 0$
- This can be computed using induction
  - For $h = 1$, $V_\pi^1(s) = R(s, \pi(s)) + V_\pi^0(s) = R(s, a) + 0$
  - For $h = 2$, $V_\pi^2(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s')R(s', \pi(s'))$
  - For any h, $V_\pi^h(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s')V_\pi^{h-1}(s')$
- Goal is to compute $V_\pi(s) = \mathbb{E}_\pi[g_t|s_t = s] = \mathbb{E}_\pi[\sum_k \gamma^k r_{t+k+1}|s_t = s]$
- γ - discount factor. $\gamma = 0$ is myopic, 1 means farsighted
- $V_\pi(s) = \mathbb{E}[R_0 + \gamma R_1 + \ldots|\pi, s_0 = s] = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s')V_\pi(s')$

Action value function

- Action Value Function - similar to state value function, it maps state-action to value
- $Q_\pi^h(s,a)$ - state-action value function wrt to π with h horizon at state s with action a. $Q_\pi^0(s,a) = 0$

Action value function

- Action Value Function - similar to state value function, it maps state-action to value
- $Q_\pi^h(s, a)$ - state-action value function wrt to π with h horizon at state s with action a. $Q_\pi^0(s, a) = 0$
- This can be computed using induction
  - For $h = 1$, $Q_\pi^1(s, a) = R(s, a) + Q_\pi^0(s, 0) = R(s, a) + 0$

Action value function

- Action Value Function - similar to state value function, it maps state-action to value
- $Q_\pi^h(s, a)$ - state-action value function wrt to π with h horizon at state s with action a. $Q_\pi^0(s, a) = 0$
- This can be computed using induction
  - For $h = 1,\ Q_\pi^1(s, a) = R(s, a) + Q_\pi^0(s, 0) = R(s, a) + 0$
  - For $h = 2,\ Q_\pi^2(s, a) = R(s, a) + \sum_{s'} T(s, \pi(s), s') \max_{a'} R(s', a')$

Action value function

- Action Value Function - similar to state value function, it maps state-action to value
- $Q_\pi^h(s,a)$ - state-action value function wrt to π with h horizon at state s with action a. $Q_\pi^0(s,a) = 0$
- This can be computed using induction
  - For $h = 1$, $Q_\pi^1(s,a) = R(s,a) + Q_\pi^0(s,0) = R(s,a) + 0$
  - For $h = 2$, $Q_\pi^2(s,a) = R(s,a) + \sum_{s'} T(s,\pi(s),s') \max_{a'} R(s',a')$
  - For any h, $Q_\pi^h(s,a) = R(s,a) + \sum_{s'} T(s,\pi(s),s') \max_{a'} Q_\pi^{h-1}(s',a')$
  - For n states, m actions, h horizon, computation time is $O(nmh)$
- Goal is to compute $Q_\pi(s,a) = \mathbb{E}_\pi[g_t | s_t = s, a_t = a] = \mathbb{E}_\pi[\sum_k \gamma^k r_{t+k+1} | s_t = s, a_t = a]$

Relationship between V and Q

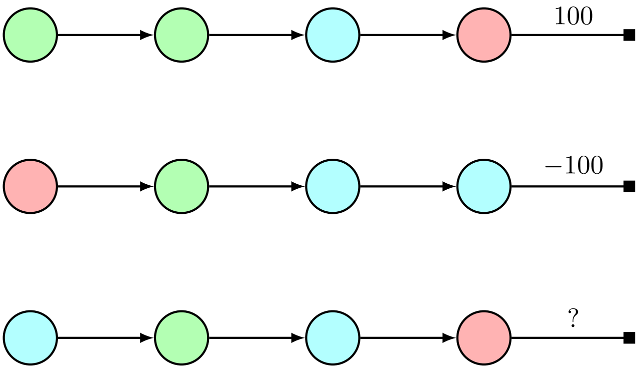
- One can be determined if the other is known
- $V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s, a)$

Model based vs Model free

- Model based - it tries to learn transition function, reward function
  - Start with a policy, interact with environment, learn world model
  - Use the world model to train the agent
  - More suitable for complex environment
  - Needs more computational resources
- Model free - finds policy or directly estimates value function or both
  - Q-learning
  - Actor-critic learning
  - More suitable for real time applications
  - Less likely to succeed in complex environment

Agent Representation of State

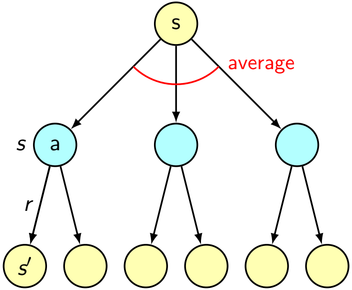
- State representation - green, blue, red sequence
- State representation - number of reds, greens, blues



Bellman expectation for state value function

- Expected return starting from s following policy π satisfies recursive relation

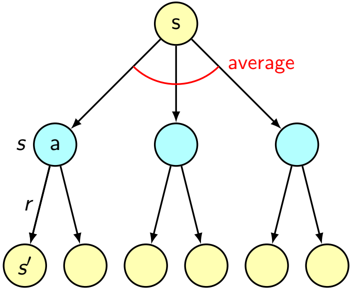
$$V_\pi(s) \quad = \quad \mathbb{E}_\pi[g_t | s_t = s]$$



Bellman expectation for state value function

- Expected return starting from s following policy π satisfies recursive relation

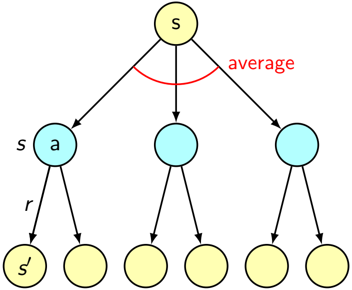
$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[g_t | s_t = s] \\ &= \mathbb{E}_\pi\left[\sum_k \gamma^k r_{t+k+1} | s_t = s\right] \end{aligned}$$



Bellman expectation for state value function

- Expected return starting from s following policy π satisfies recursive relation

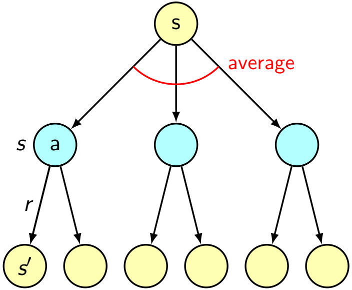
$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[g_t | s_t = s] \\
&= \mathbb{E}_\pi\left[\sum_k \gamma^k r_{t+k+1} | s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma \sum_k \gamma^k r_{t+k+2} | s_t = s\right]
\end{aligned}
$$



Bellman expectation for state value function

- Expected return starting from s following policy π satisfies recursive relation

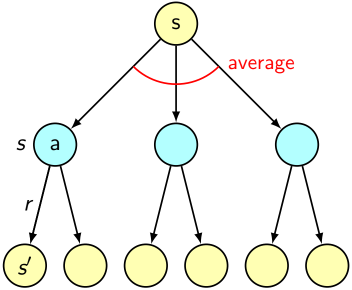
$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[g_t | s_t = s] \\ &= \mathbb{E}_\pi\left[\sum_k \gamma^k r_{t+k+1} | s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma \sum_k \gamma^k r_{t+k+2} | s_t = s\right] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma g_{t+1} | s_t = s] \end{aligned}$$



Bellman expectation for state value function

- Expected return starting from s following policy π satisfies recursive relation

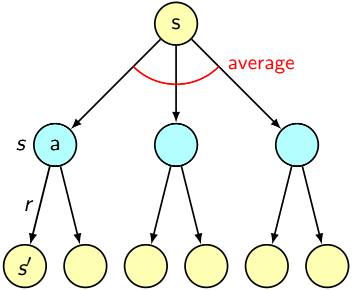
$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[g_t|s_t = s] \\
&= \mathbb{E}_\pi\left[\sum_k \gamma^k r_{t+k+1}|s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma \sum_k \gamma^k r_{t+k+2}|s_t = s\right] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma g_{t+1}|s_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)(r + \gamma \mathbb{E}_\pi[g_{t+1}|s_{t+1} = s'])
\end{aligned}
$$



Bellman expectation for state value function

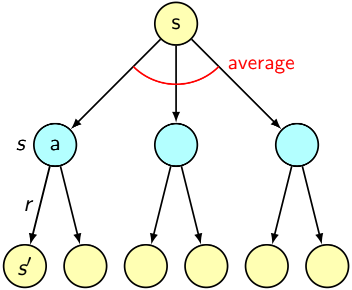
- Expected return starting from s following policy π satisfies recursive relation

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[g_t | s_t = s] \\
&= \mathbb{E}_\pi\left[\sum_k \gamma^k r_{t+k+1} | s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma \sum_k \gamma^k r_{t+k+2} | s_t = s\right] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma g_{t+1} | s_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)(r + \gamma \mathbb{E}_\pi[g_{t+1} | s_{t+1} = s']) \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)(r + \gamma V_\pi(s'))
\end{aligned}
$$



Bellman expectation for state value function

- Expected return starting from s following policy π satisfies recursive relation



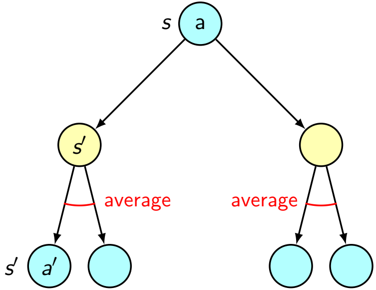
$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[g_t | s_t = s] \\
&= \mathbb{E}_\pi\left[\sum_k \gamma^k r_{t+k+1} | s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma \sum_k \gamma^k r_{t+k+2} | s_t = s\right] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma g_{t+1} | s_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)(r + \gamma \mathbb{E}_\pi[g_{t+1} | s_{t+1} = s']) \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)(r + \gamma V_\pi(s'))
\end{aligned}
$$

- In matrix form $\mathrm{V}_\pi^{h+1} = r + \mathrm{T}\mathrm{V}_\pi^h$, T - transition matrix

Bellman expectation for state-action value function

- Expected return starting from s following policy π satisfies recursive relation

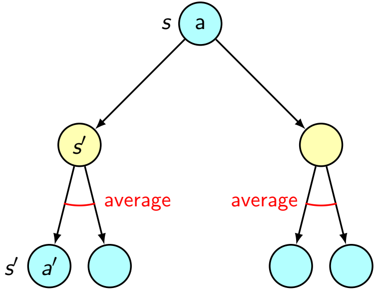
$$Q_\pi(s, a) \quad = \quad \mathbb{E}_\pi[g_t | s_t = s, a_t = a]$$



Bellman expectation for state-action value function

- Expected return starting from s following policy π satisfies recursive relation

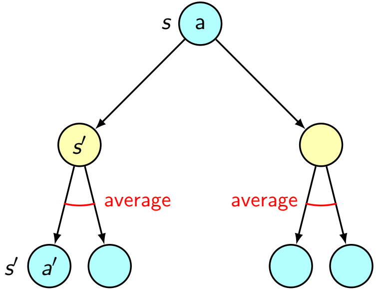
$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi[g_t | s_t = s, a_t = a] \\
&= \mathbb{E}_\pi\left[\sum_k \gamma^k r_{t+k+1} | s, a\right]
\end{aligned}
$$



Bellman expectation for state-action value function

- Expected return starting from s following policy π satisfies recursive relation

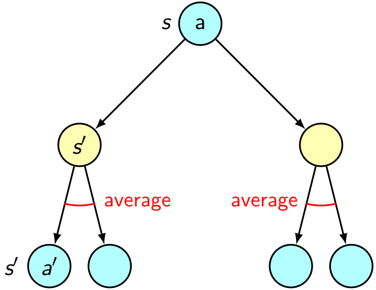
$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi[g_t | s_t = s, a_t = a] \\
&= \mathbb{E}_\pi\left[\sum_k \gamma^k r_{t+k+1} | s, a\right] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma Q(s', a') | s, a]
\end{aligned}
$$



Bellman expectation for state-action value function

- Expected return starting from s following policy π satisfies recursive relation

$$\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi[g_t | s_t = s, a_t = a] \\
&= \mathbb{E}_\pi\left[\sum_k \gamma^k r_{t+k+1} | s, a\right] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma Q(s', a') | s, a] \\
&= \sum_{s',r} p(s', r | s, a)\left(r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s', a')\right)
\end{aligned}$$



Optimal policy

- A policy π is better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all all states: $\pi \geq \pi'$ iff $V_{\pi}(s) \geq V_{\pi'}(s) \quad \forall s$

Optimal policy

- A policy π is better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all all states: $\pi \geq \pi'$ iff $V_\pi(s) \geq V_{\pi'}(s) \quad \forall s$
- Optimal value function: $V_* = \max_\pi V_\pi(s) = \max_\pi \mathbb{E}_\pi[g_t | s_t = s]$
- Optimal state-action value: $Q_*(s, a) = \max_\pi Q_\pi(s, a)$

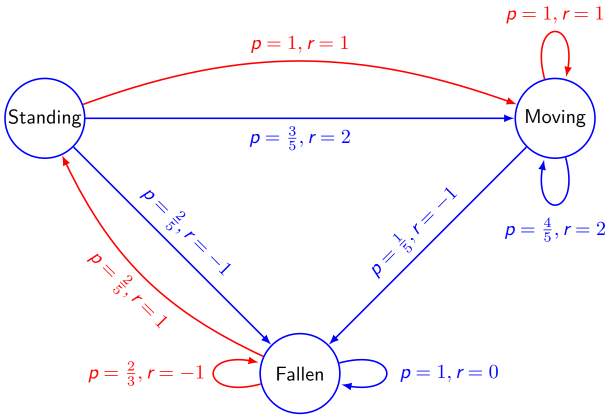
Optimal policy

- A policy π is better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all all states: $\pi \geq \pi'$ iff $V_\pi(s) \geq V_{\pi'}(s) \quad \forall s$
- Optimal value function: $V_* = \max_\pi V_\pi(s) = \max_\pi \mathbb{E}_\pi[g_t | s_t = s]$
- Optimal state-action value: $Q_*(s, a) = \max_\pi Q_\pi(s, a)$
- Bellman optimal equation for state value: $V_*(s) = \max_a \sum_{s', r} p(s', r | s, a)(r + \gamma V_*(s'))$
- Bellman optimal equation for state-action value:

  $$Q_*(s, a) = \sum_{s', r} p(s', r | s, a)(r + \gamma \max_{a'} Q_*(s', a'))$$

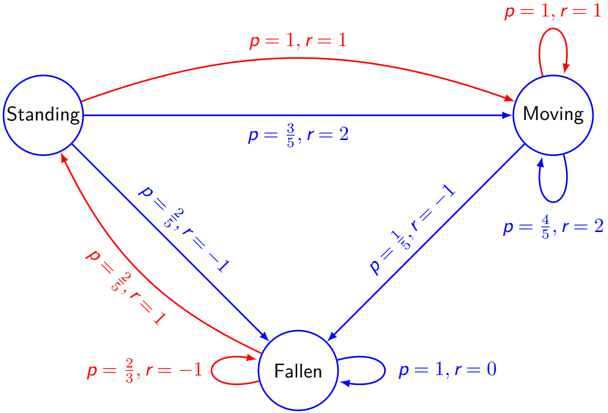
Example

- $V_*^1(f) = 0$, $V_*^1(s) = 1$, $V_*^1(m) = \frac{7}{5}$



Example

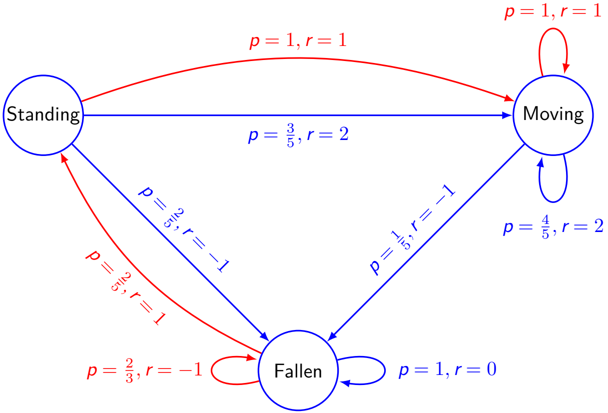
- $V_*^1(f) = 0$, $V_*^1(s) = 1$, $V_*^1(m) = \frac{7}{5}$
- $V_*^2(f) = \max\{-\frac{1}{5} + \frac{2}{5} \times 1, 0 + 0\} = \frac{1}{5}$,



Example

- $V_*^1(f) = 0$, $V_*^1(s) = 1$, $V_*^1(m) = \frac{7}{5}$
- $V_*^2(f) = \max\{-\frac{1}{5} + \frac{2}{5} \times 1, 0 + 0\} = \frac{1}{5}$,

  $V_*^2(s) = \max\{1 + \frac{7}{5}, \frac{4}{5} + \frac{3}{5} \times \frac{7}{5}\} = \frac{12}{5}$,

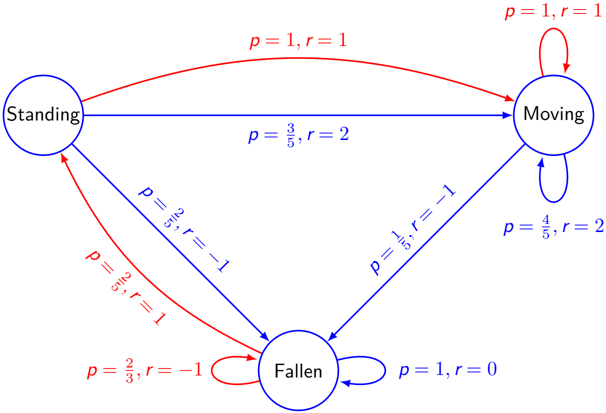


Example

- $V_*^1(f) = 0,\ V_*^1(s) = 1,\ V_*^1(m) = \frac{7}{5}$
- $V_*^2(f) = \max\{-\frac{1}{5} + \frac{2}{5} \times 1, 0 + 0\} = \frac{1}{5},$

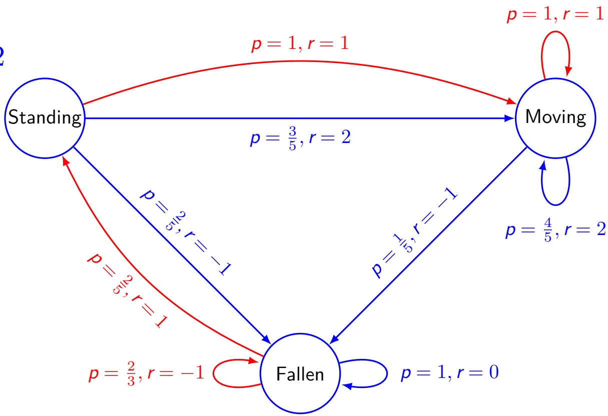
  $V_*^2(s) = \max\{1 + \frac{7}{5}, \frac{4}{5} + \frac{3}{5} \times \frac{7}{5}\} = \frac{12}{5},$

  $V_*^2(m) = \max\{1 + \frac{7}{5}, \frac{7}{5} + \frac{4}{5} \times \frac{7}{5}\} = 2.52$



Example

- $V_*^1(f) = 0$, $V_*^1(s) = 1$, $V_*^1(m) = \frac{7}{5}$
- $V_*^2(f) = \max\{-\frac{1}{5} + \frac{2}{5} \times 1, 0 + 0\} = \frac{1}{5}$,
  $V_*^2(s) = \max\{1 + \frac{7}{5}, \frac{4}{5} + \frac{3}{5} \times \frac{7}{5}\} = \frac{12}{5}$,
  $V_*^2(m) = \max\{1 + \frac{7}{5}, \frac{7}{5} + \frac{4}{5} \times \frac{7}{5}\} = 2.52$
- $V_*^3(f) = 0.88$, $V_*^3(s) = 3.52$, $V_*^3(m) = 3.52$



Iterative policy evaluation

- Steps to determine $V(s)$ given a policy:

  initialize $V(s) = 0 \quad \forall s$

  do:

  $\Delta = 0$

  for $s \in S$ do:

  $v = V(s)$

  $V(s) = \sum_a \pi(a|s) \sum_{s',a} p(s', r|s, a)(r + \gamma V(s'))$

  $\Delta = \max\{\Delta, \|v - V(s)\|\}$

  until $\Delta < \epsilon$

Policy iteration

- Steps for determining policy:

  initialize $V(s), \pi(s)$

  do:

  Run iterative policy evaluation (compute V)

  convergence = True

  for $s \in S$ do:

  $a = \pi(s)$

  $\pi(s) = \arg\max_a \sum_{s',a} p(s', r | s, a)(r + \gamma V(s'))$

  if $a \neq \pi(s)$ then: convergence = False

  until convergence

Value iteration

- Steps are as follows

initialize $Q(s,a) = 0 \quad \forall s, a$

do:

for $(s,a) \in (S,A)$ do:

$q = Q(s,a)$

$Q(s,a) = r(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$

$\Delta = \max\{\Delta, \|v - V(s)\|\}$

until $\Delta < \epsilon$

Model based RL

- To model the transition relation and rewards based on states, actions and rewards experienced, (s, a, r, s')
- Let the agent make action and observes the states and rewards
- Transition can be estimated as: $T(s, a, s') = \dfrac{N(s, a, s') + 1}{N(s, a) + |S|}$, where

  $N(s, a, s')$ - number of times the agent was in s, moves to s' on action s,

  $N(s, a) = \sum_{s'} N(s, a, s')$
- Reward function can be estimated as: $R(s, a) = \dfrac{\sum_{(s,a)} r(s, a)}{N(s, a)}$

Monte Carlo sampling

- It considers experiences is divided into episodes that terminates
- Reward value can be computed only when after termination
- Example: incremental mean computation
  - Mean at time step t is updated based on current value x_t and mean at time $(t-1)$
  - $\mu_t = \frac{1}{t}\sum_{i=1}^{t} x_i = \mu_{t-1} + \frac{1}{t}(x_t - \mu_{t-1})$
- For MDP, $V(s_t) = V(s_t) + \alpha(g_t - V(s_t))$, g_t actual return

Monte Carlo prediction

- Steps are as follows

  initialize $V(s) = 0$, return$(s) = \emptyset$

  do:

  Generate episode of π

  for $s \in S$ do:

  $g =$ return following the first occurence of s

  return(s)= return$(s) \cup g$

  $V(s) = \mu($return$(s))$

  until convergence

Temporal difference

- It also considers experience to solve the prediction problem
- Reward value is computed only until the next time step (Monte Carlo considers termination)
- For MDP, $V(s_t) = V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$

Temporal difference prediction

- Steps for determining $V(s)$ are as follows

  initialize $V(s) = 0$

  do:

      Generate episode of π

      for $s \in S$ do:

          $a =$ action given by π at s

          Take action a, observe r, s'

          $V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$

          $s = s'$

  until s is terminal

Q-Learning

- It is a model free learning method, directly estimates the a value function
- In model based we estimate T and R using value iteration. Given T and R we can compute $Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$
- In Q-learning, we estimate $Q(s,a) = Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$

Q-Learning

- It is a model free learning method, directly estimates the a value function
- In model based we estimate T and R using value iteration. Given T and R we can compute $Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$
- In Q-learning, we estimate $Q(s,a) = Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$
- Steps:

  initialize $Q(s,a) = 0$; Select start state s_0

  do:

  $a =$ select an action based on ϵ-greedy strategy

  $q = Q(s,a)$

  Take action a to get reward r and next state s'

  $Q(s,a) = Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$

  $\Delta = \max\{\Delta, \|q - Q(s,a)\|\}; \quad s = s'$

  until Δ is less than a given threshold

SARSA

- In Q-learning, we estimate $Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
- In SARSA, we estimate $Q(s,a) = Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right)$
  - It needs (s,a,r,s',a') tuple for learning
- On policy vs Off policy

State value function approximation

- A neural network may be used to estimate $V_\theta(s) \approx V_\pi(s)$ or $Q_\theta(s,a) \approx Q_\pi(s,a)$ or a policy $p_\theta(a|s)$

State value function approximation

- A neural network may be used to estimate $V_\theta(s) \approx V_\pi(s)$ or $Q_\theta(s,a) \approx Q_\pi(s,a)$ or a policy $p_\theta(a|s)$
- Our goal is to find θ such that MSE between $V_\pi(s)$ and $V_\theta(s)$ is minimized

State value function approximation

- A neural network may be used to estimate $V_\theta(s) \approx V_\pi(s)$ or $Q_\theta(s, a) \approx Q_\pi(s, a)$ or a policy $p_\theta(a|s)$
- Our goal is to find θ such that MSE between $V_\pi(s)$ and $V_\theta(s)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_s[(V_\pi(s) - V_\theta(s))^2]$

State value function approximation

- A neural network may be used to estimate $V_\theta(s) \approx V_\pi(s)$ or $Q_\theta(s, a) \approx Q_\pi(s, a)$ or a policy $p_\theta(a|s)$
- Our goal is to find θ such that MSE between $V_\pi(s)$ and $V_\theta(s)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_s[(V_\pi(s) - V_\theta(s))^2] = \frac{1}{2S}\sum_{s \in S}[(V_\pi(s) - V_\theta(s))^2]$

State value function approximation

- A neural network may be used to estimate $V_\theta(s) \approx V_\pi(s)$ or $Q_\theta(s, a) \approx Q_\pi(s, a)$ or a policy $p_\theta(a|s)$
- Our goal is to find θ such that MSE between $V_\pi(s)$ and $V_\theta(s)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_s[(V_\pi(s) - V_\theta(s))^2] = \frac{1}{2S}\sum_{s \in S}[(V_\pi(s) - V_\theta(s))^2] \approx \sum_{s \in S}\mu(s)[(V_\pi(s) - V_\theta(s))^2]$

State value function approximation

- A neural network may be used to estimate $V_\theta(s) \approx V_\pi(s)$ or $Q_\theta(s,a) \approx Q_\pi(s,a)$ or a policy $p_\theta(a|s)$
- Our goal is to find θ such that MSE between $V_\pi(s)$ and $V_\theta(s)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_s[(V_\pi(s) - V_\theta(s))^2] = \frac{1}{2S}\sum_{s\in S}[(V_\pi(s) - V_\theta(s))^2] \approx \sum_{s\in S}\mu(s)[(V_\pi(s) - V_\theta(s))^2]$
- Gradient: $\nabla_\theta J(\theta) = -\mathbb{E}_s[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$
- For gradient update: $\Delta\theta = -\alpha\nabla_\theta J(\theta) = \alpha\mathbb{E}_s[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$

State value function approximation

- A neural network may be used to estimate $V_\theta(s) \approx V_\pi(s)$ or $Q_\theta(s,a) \approx Q_\pi(s,a)$ or a policy $p_\theta(a|s)$
- Our goal is to find θ such that MSE between $V_\pi(s)$ and $V_\theta(s)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_s[(V_\pi(s) - V_\theta(s))^2] = \frac{1}{2S}\sum_{s\in S}[(V_\pi(s) - V_\theta(s))^2] \approx \sum_{s\in S}\mu(s)[(V_\pi(s) - V_\theta(s))^2]$
- Gradient: $\nabla_\theta J(\theta) = -\mathbb{E}_s[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$
- For gradient update: $\Delta\theta = -\alpha\nabla_\theta J(\theta) = \alpha\mathbb{E}_s[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$
- If we have single sample (like SGD), $\Delta\theta = \alpha\nabla_\theta J(\theta) = \alpha[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$

State value function approximation

- A neural network may be used to estimate $V_\theta(s) \approx V_\pi(s)$ or $Q_\theta(s,a) \approx Q_\pi(s,a)$ or a policy $p_\theta(a|s)$
- Our goal is to find θ such that MSE between $V_\pi(s)$ and $V_\theta(s)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_s[(V_\pi(s) - V_\theta(s))^2] = \frac{1}{2S}\sum_{s \in S}[(V_\pi(s) - V_\theta(s))^2] \approx \sum_{s \in S}\mu(s)[(V_\pi(s) - V_\theta(s))^2]$
- Gradient: $\nabla_\theta J(\theta) = -\mathbb{E}_s[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$
- For gradient update: $\Delta\theta = -\alpha\nabla_\theta J(\theta) = \alpha\mathbb{E}_s[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$
- If we have single sample (like SGD), $\Delta\theta = \alpha\nabla_\theta J(\theta) = \alpha[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$
    - For Monte Carlo: $\Delta\theta = \alpha[(g - V_\theta(s))\nabla_\theta V_\theta(s)]$

State value function approximation

- A neural network may be used to estimate $V_\theta(s) \approx V_\pi(s)$ or $Q_\theta(s,a) \approx Q_\pi(s,a)$ or a policy $p_\theta(a|s)$
- Our goal is to find θ such that MSE between $V_\pi(s)$ and $V_\theta(s)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_s[(V_\pi(s) - V_\theta(s))^2] = \frac{1}{2S}\sum_{s\in S}[(V_\pi(s) - V_\theta(s))^2] \approx \sum_{s\in S}\mu(s)[(V_\pi(s) - V_\theta(s))^2]$
- Gradient: $\nabla_\theta J(\theta) = -\mathbb{E}_s[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$
- For gradient update: $\Delta\theta = -\alpha\nabla_\theta J(\theta) = \alpha\mathbb{E}_s[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$
- If we have single sample (like SGD), $\Delta\theta = \alpha\nabla_\theta J(\theta) = \alpha[(V_\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)]$
  - For Monte Carlo: $\Delta\theta = \alpha[(g - V_\theta(s))\nabla_\theta V_\theta(s)]$
  - For Temporal Difference: $\Delta\theta = \alpha[(r + \gamma V_\theta(s') - V_\theta(s))\nabla_\theta V_\theta(s)]$

Action value function approximation

- Need to estimate $Q_\theta(s, a)$
- Our goal is to find θ such that MSE between $Q_\pi(s, a)$ and $Q_\theta(s, a)$ is minimized

Action value function approximation

- Need to estimate $Q_\theta(s,a)$
- Our goal is to find θ such that MSE between $Q_\pi(s,a)$ and $Q_\theta(s,a)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_{(s,a)\sim\pi}[(Q_\pi(s,a) - Q_\theta(s,a))^2]$

Action value function approximation

- Need to estimate $Q_\theta(s, a)$
- Our goal is to find θ such that MSE between $Q_\pi(s, a)$ and $Q_\theta(s, a)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_{(s,a)\sim\pi}[(Q_\pi(s, a) - Q_\theta(s, a))^2]$
- Gradient: $\nabla_\theta J(\theta) = -\mathbb{E}_{(s,a)\sim\pi}[(Q_\pi(s, a) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$

Action value function approximation

- Need to estimate $Q_\theta(s, a)$
- Our goal is to find θ such that MSE between $Q_\pi(s, a)$ and $Q_\theta(s, a)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_{(s,a)\sim\pi}[(Q_\pi(s, a) - Q_\theta(s, a))^2]$
- Gradient: $\nabla_\theta J(\theta) = -\mathbb{E}_{(s,a)\sim\pi}[(Q_\pi(s, a) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$
- For gradient update: $\Delta\theta = -\alpha\nabla_\theta J(\theta) = \alpha\mathbb{E}_s[(Q_\pi(s, a) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$

Action value function approximation

- Need to estimate $Q_\theta(s, a)$
- Our goal is to find θ such that MSE between $Q_\pi(s, a)$ and $Q_\theta(s, a)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_{(s,a)\sim\pi}[(Q_\pi(s, a) - Q_\theta(s, a))^2]$
- Gradient: $\nabla_\theta J(\theta) = -\mathbb{E}_{(s,a)\sim\pi}[(Q_\pi(s, a) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$
- For gradient update: $\Delta\theta = -\alpha\nabla_\theta J(\theta) = \alpha\mathbb{E}_s[(Q_\pi(s, a) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$
- If we have single sample, $\Delta\theta = \alpha\nabla_\theta J(\theta) = \alpha[(Q_\pi(s, a) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$

Action value function approximation

- Need to estimate $Q_\theta(s, a)$
- Our goal is to find θ such that MSE between $Q_\pi(s, a)$ and $Q_\theta(s, a)$ is minimized
- $J(\theta) = \frac{1}{2}\mathbb{E}_{(s,a)\sim\pi}[(Q_\pi(s, a) - Q_\theta(s, a))^2]$
- Gradient: $\nabla_\theta J(\theta) = -\mathbb{E}_{(s,a)\sim\pi}[(Q_\pi(s, a) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$
- For gradient update: $\Delta\theta = -\alpha\nabla_\theta J(\theta) = \alpha\mathbb{E}_s[(Q_\pi(s, a) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$
- If we have single sample, $\Delta\theta = \alpha\nabla_\theta J(\theta) = \alpha[(Q_\pi(s, a) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$
  - For Monte Carlo: $\Delta\theta = \alpha[(g - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$
  - For Temporal Difference: $\Delta\theta = \alpha[(r + \gamma Q_\theta(s', a') - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)]$
- Can diverge using neural network due to
  - Correlation between samples
  - Non-stationary target

Experience replay

- Neural network needs to learn from states, action, reward information ie. $e_i = (s_i, a_i, r_i, s_i')$
- Successive samples are usually correlated
- Need to use replay buffer that stores e_i
- Sample from the buffer when updating Q values

Neural fitted Q-iteration (NFQ)

- Our goal is to find θ such that MSE between $Q_*(s,a)$ and $Q_\theta(s,a)$ is minimized
- Loss: $J(\theta) = \frac{1}{2}\mathbb{E}_{(s,a)\sim\pi}[(Q_*(s,a) - Q_\theta(s,a))^2]$
- For gradient update: $\Delta\theta = \alpha[(Q_*(s,a) - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a)]$
- Since we do not know $Q_*(s,a)$, optimal action value can be approximated as $Q_*(s,a) \approx r + \gamma \max_{a'} Q_\theta(s',a')$
- Hence network parameters updated by $\Delta\theta = \alpha\left[r + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)\right]\nabla_\theta Q_\theta(s,a)$

Deep Q-Network (DQN)

- It uses a second neural network
- In NFQ, we set the target as $y_{NFQ} = r + \gamma \max_{a'} Q_\theta(s', a')$
- In case of DQN, we use $y_{DQN} = r + \gamma \max_{a'} Q_{\theta-}(s', a')$
- DQN minimizes MSE loss

$$L(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim D_i}\left[(y_i - Q_{\theta_i}(s,a))^2\right] = \mathbb{E}_{(s,a,r,s')\sim D_i}\left[r + \gamma \max_{a'} Q_{\theta\_}(s', a') - Q_{\theta_i}(s,a))^2\right]$$

- Parameters $\theta\_$ of the target network $Q_{\theta\_}(s', a')$ are frozen for multiple steps, θ_i are updated using SGD
- $\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim D_i}\left[(r + \gamma \max_{a'} Q_{\theta\_}(s', a') - Q_{\theta_i}(s,a))\nabla_{\theta_i} Q_{\theta_i}(s,a)\right]$

DQN Algorithm

- Steps are as follows:

  initialize (1) $D = \emptyset$ - empty reply buffer, (2) online Q_θ network parameters with θ with random values, (3) set for target network $Q_{\theta\_}$ parameters $\theta\_ = \theta$, (4) start state $s = s_0$

  repeat:

      for each episode do:

          run ϵ-greedy policy based Q_θ network

          collect transitions (s, a, r, s') in D

      Select a sample (s, a, r, s') from D

      $q = Q_\theta(s, a)$;

      $Q_\theta(s, a) = Q_\theta(s, a) + \alpha(r + \gamma \max_{a'} Q_{\theta\_}(s', a') - Q_\theta(s, a))$

      $\Delta = \max\{\Delta, \|q - Q_\theta(s, a)\|\}$

      $s = s'$

      update $\theta\_ = \theta$ every k number of episodes

  until stopping criteria

References

- *Reinforcement Learning: An Introduction* by Andrew Barto and Richard S. Sutton
- *Human-level control through deep reinforcement learning* by Deep Mind, Google

Example: Recycling Robot

- A robot does one of the following at each time step
  - Actively search for a can
  - Remain stationary and wait for someone to bring a can
  - Go back to home base to recharge battery

Recycling Robot: Transition relation

s	s'	a	$p(s' \mid s, a)$	$r(s, a, s')$
high	high	search	α	r_{search}
high	low	search	$1 - \alpha$	r_{search}
low	high	search	$1 - \beta$	-3
low	low	search	β	r_{search}
high	high	wait	1	r_{wait}
high	low	wait	0	r_{wait}
low	high	wait	0	r_{wait}
low	low	wait	1	r_{wait}
low	high	recharge	1	0
low	low	recharge	0	0

Example

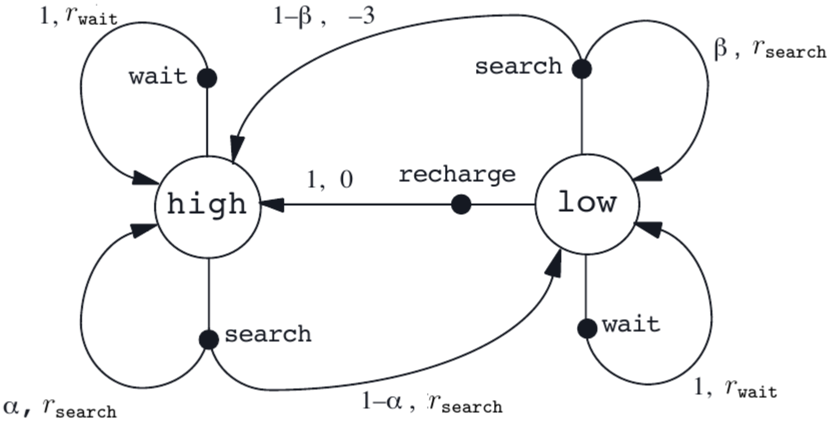


Image source: Reinforcement Learning by Andrew Barto and Richard S. Sutton

Optimal value computation

- For recycling robot - h - high, l - low, s - search, w - wait, r - recharge

$$
\begin{aligned}
V^*(h) &= \max\left\{ \begin{array}{l} p(h|h,s)[r(h,s,h)+\gamma V^*(h)] + p(l|h,s)[r(h,s,l)+\gamma V^*(l)], \\ p(h|h,w)[r(h,w,h)+\gamma V^*(h)] + p(l|h,w)[r(h,w,l)+\gamma V^*(l)] \end{array} \right\} \\
V^*(h) &= \max\{r_s + \gamma[\alpha V^*(h) + (1-\alpha)V^*(l)],\ r_w + \gamma V^*(h)\}
\end{aligned}
$$

$$
V^*(l) = \max\left\{ \begin{array}{l} \beta r_s - 3(1-\beta) + \gamma[(1-\beta)V^*(h) + \beta V^*(l)] \\ r_w + \gamma V^*(l), \\ \gamma V^*(h) \end{array} \right\}
$$