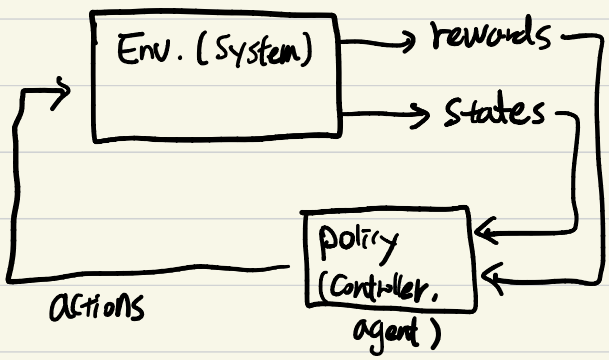
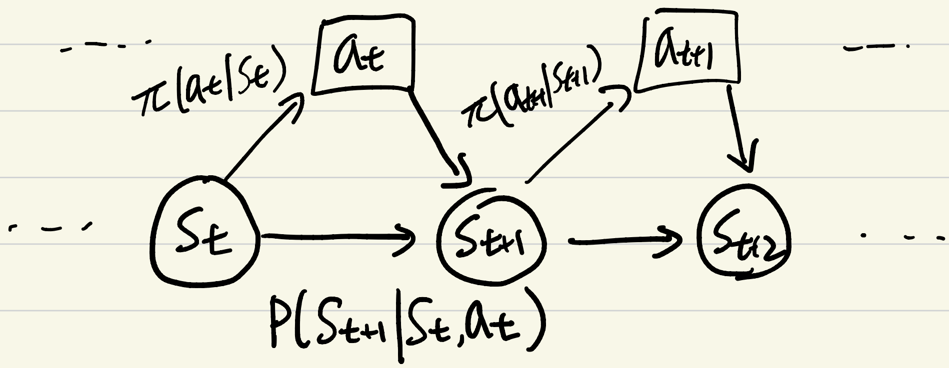
Reinforcement Learning

Today's
- MDP
- Q-Learning
- Deep Q-learning



- states, s_t
- rewards, r_t
- actions, a_t
- Transition prob., p
- discount factor, γ

* Markov Decision Process (MDP)



Goal: train a policy to solve discrete MDP.

policy, π is a function that maps states to actions

optimal policy, π^*

① The return, R_t is the total discounted reward from time t.

$$R_t \doteq \sum_{k=0}^{T-1} \gamma^k r(s_{t+k}, a_{t+k})$$

② The state value function, $V^\pi(s_t)$ is the "expected" total discounted return from state s_t

$$\begin{aligned} V^\pi(s_t) &\doteq \mathbb{E}_\pi [R_t | s_t] \\ &= \mathbb{E}_\pi \sum_{t'=t}^{T-1} [r(s_{t'}, a_{t'}) | s_t] \\ &= \sum_{t'=t}^{T-1} \mathbb{E}_\pi [r(s_{t'}, a_{t'}) | s_t] \end{aligned}$$

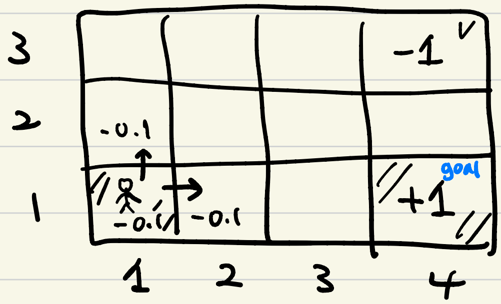
③ The state-action value function, $Q^\pi(s_t, a_t)$ is the expected total discounted return when action a_t is taken in state s_t.

$$\begin{aligned} Q^\pi(s_t, a_t) &\doteq \mathbb{E}_\pi [R_t | s_t, a_t] \\ &= \mathbb{E}_\pi \sum_{t'=t}^{T-1} [r(s_{t'}, a_{t'}) | s_t, a_t] \\ &= \sum_{t'=t}^{T-1} \mathbb{E}_\pi [r(s_{t'}, a_{t'}) | s_t, a_t] \end{aligned}$$

example,,

$$\begin{aligned} Q(s_1, a_1) &= r(s_1, a_1) + \mathbb{E}_{s_2 \sim p(s_2|s_1,a_1)} \left[\mathbb{E}_{a_2 \sim \pi(a_2|s_2)} [r(s_2, a_2) | s_2] \right] \\ &= r(s_1, a_1) + \mathbb{E}_{s_2 \sim p(s_2|s_1,a_1)} V^\pi(s_2) \\ &\approx r(s_1, a_1) + V^\pi(s_2) \end{aligned}$$

Example : Grid World



- States : grid Coordinates (1,1), (1,2) ⋯ (4,4)
- actions : { left, right, up, down }
- rewards : value in the box
- Trans. prob :
  - 60% : you can go to next states what you intend.
  - 20% : go right
  - 20% : go left.
- objective : maximize R_t
- initial states : (1,1)

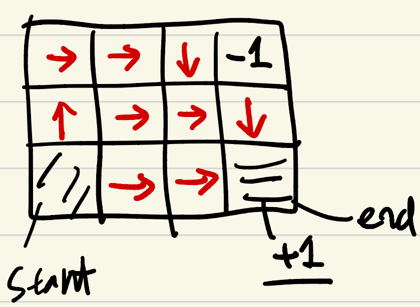
How do we find an optimal policy for Grid World.

Q-function tells you the value of starting in state s_t, applying action, a_t

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T-1} r(s_t, a_t) \,\middle|\, s_t, a_t\right]$$

$$= r(s_t, a_t) + \underset{1}{\gamma} \sum_{a_t \in A} \pi(a_t|s_t) \underbrace{\sum_{s_{t+1} \in S} p(s_{t+1}|s_t, a_t)}_{V^{\pi}(s_{t+1})}$$

The optimal Q-function for MDP, is deterministic



$$Q^{*}(s_t, a_t) = r(s_t, a_t) + \underset{1}{\gamma} \underbrace{\sum_{s_{t+1} \in S} p(s_{t+1}|s_t, a_t)}_{V^{\pi}(s_{t+1})}$$

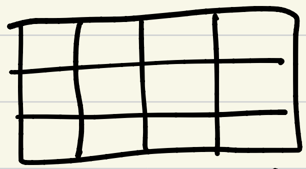
Useful relationship: $V^*(s) = \max_a Q^*(s,a)$

$$* \quad Q^*(s,a) = r(s,a) + \gamma \sum_{s_{t+1} \in S} P(s_{t+1} | s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

What's the optimal action? a^*?

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

* Iterated Q-Learning



Gridworld (2D)

① Initialize V_0, Q_0

② For $i = 1$: num_steps :

For $\forall s \in S$:

$$Q_i(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q_i(s',a')$$

once it's converged, $\pi(a|s) = \arg\max_a Q(s,a)$

If we have a large number of states, then tabular representation is inefficient and intractable.

Quick Review

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} p(s_{t+1} | s_t, a_t) V(s_{t+1})$$

Standard Q-learning

$$Q(s,a) \leftarrow r(s,a) + \mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s_t,a_t)} \underline{[V(s_{t+1})]}$$

$$\Longleftrightarrow$$

$$\leftarrow r(s,a) + \sum_{s_{t+1} \in S} p(s_{t+1}|s_t,a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

How do we compute $\mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s_t,a_t)} [V(s_{t+1})]$?

$\Rightarrow$ The key idea in Q-learning is to use samples of the next state s_{t+1} in place of expectation.

$$\Longleftrightarrow \quad \mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s_t,a_t)} [V(s_{t+1})] \approx \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

This will converge for the tabular case. Provable convergence of Q-learning requires visiting all state-action pairs infinitely many times.

* Q-learning w/ function Approximator.

- Q-learning

$$Q(s_t, a_t) \leftarrow r(s_t, a_t) + \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

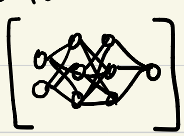
- we parameterize Q function by func. appox, $\underline{\underline{Q_\theta}}$.
- θ are the weights of the approximator,
  - $y_i \leftarrow r(s_{(i,t)}, a_{(i,t)}) + \max_{a_{(i,t+1)}} Q_\theta(s_{(i,t+1)}, a_{(i,t+1)})$
  - $\theta \leftarrow \underset{\theta}{\operatorname{argmin}} \sum_i \| y_i - Q_\theta(s_{(i,t)}, a_{(i,t)}) \|^2$
- Use gradient descent to find θ.

※ with this approach, the convergence is no longer guaranteed.

---

* Deep Q-learning (DQN)

D = Deep, Q = Q function, N = Network

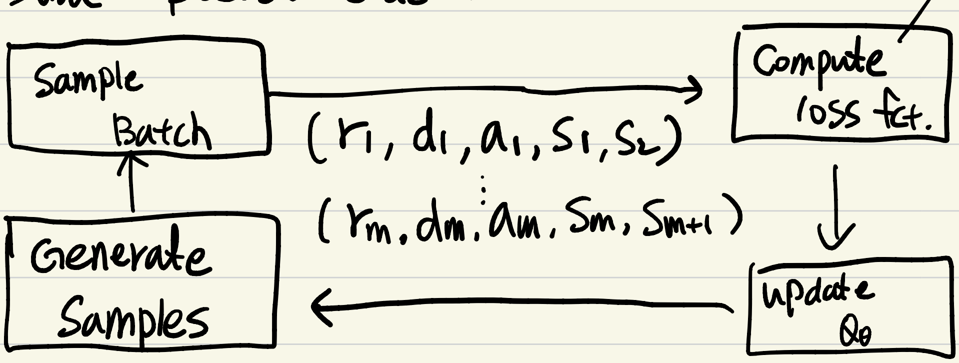


NN

1. Replay Buffers
2. Target Q-function

(s, a, s', r)

- Visual pseudo-code :



$$\sum_i^m \| Q_\theta(s_i, a_i) - \left(r(s_i, a_i) + (1 - d_i) \max_a Q_t(s_{t+1}, a) \right) \|^2$$

target Q-function.

(update $Q_t = Q_\theta$)

① Replay Buffer.

↳ Save samples, $(s_i, a_i, s'_i, r_i, d_i)$

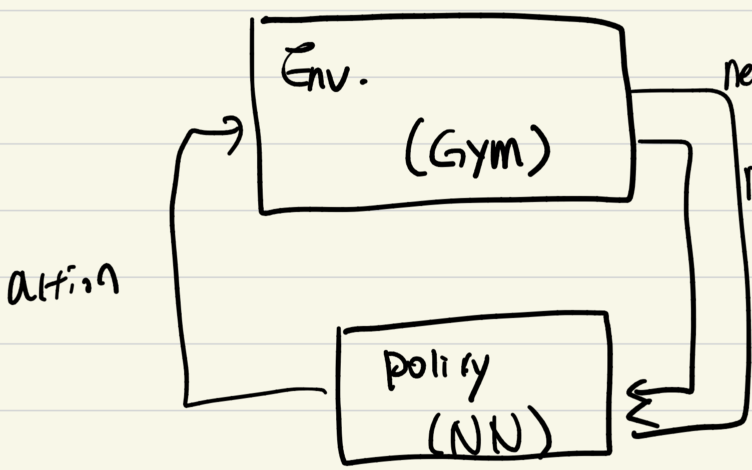
② Target Q-function

Idea: slow down the moving target value.

$$y_i \Leftarrow r(s_i, a_i) + \max_{a'} Q_t(s'_i, a')$$

please check the class notes about DQN algorithm

Youtube link for DQN on Atari-games.



next_state = Env.step(action)

nextact = Qfct(next_state)