CS 182 Lecture 16: Actor-Critic & Q-learning

7/15

$\hat{Q}_{i,t}$: estimate of expected reward - "reward to go", can be optimized!

↳ Ideally: $\sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'}) | s_t, a_t]$ (true expected reward, all possible rewards)

$Q^\pi(s_t, a_t)$ → total reward from taking a_t in s_t

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \underbrace{(Q^\pi(s_{i,t}, a_{i,t}) - V^\pi(s_{i,t}))}_{A^\pi(s_{i,t}, a_{i,t})}, \quad V(s_{i,t}) = E_{a_t \sim \pi_\theta(a_t|s_t)}[Q(s_t, a_t)]$$

better estimate → lower variance. "advantage" → $A^\pi(s_{i,t}, a_{i,t})$; baseline "average reward"

→ essence: separately fit $Q^\pi / V^\pi / A^\pi$. (value-function network aka actor-critic).

Bellman equation →

$$Q^\pi(s_t, a_t) = \underbrace{r(s_t, a_t)}_{\text{current reward}} + E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}[V^\pi(s_{t+1})] \quad \left(\sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'}) | s_t, a_t]\right)$$

↖ expectation of the future.

$$\approx r(s_t, a_t) + V^\pi(s_{t+1})$$

$$A^\pi(s_t, a_t) \approx r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t)$$ ← just fit $V^\pi(s)$. → neural network is separately trained ($s \to V$) (policy evaluation)

↳ Policy Evaluation (fit V^π) → "Monte Carlo" method

$$V^\pi(s_t) \approx \sum_{t'=t}^{T} r(s_{t'}, a_{t'})$$ (a trajectory, $y_{i,t}$ "reward to end")

generate samples (run the policy) → fit model → improve policy → $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

training data: $\left\{\left(s_{i,t}, \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})\right)\right\}$

- two network design
- shared network design

Example: TD-gammon (1992)
↳ reward: game outcome
↳ value function ($\hat{V}_\theta^\pi(s_t)$): expected outcome given board state.

use regression + MSE. bootstrap → can improve: $y_{i,t} = r(s_{i,t}, a_{i,t}) + \hat{V}_\theta^\pi(s_{i,t+1})$ (previous fitted value function)

Two networks:
"Actor" - policy π_θ
"Critic" - value V^π

batch actor-critic algorithm:
1. sample $\{s_i, a_i\}$ from $\pi_\theta(a|s)$
2. fit $\hat{V}_\theta^\pi(s)$ to sampled reward sums
3. evaluate $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \hat{V}_\theta^\pi(s_i') - \hat{V}_\theta^\pi(s_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

w/ Monte-carlo or bootstrap

gradient → ascent

discount factor γ: rewards sooner than later. $0 < \gamma < 1$, typically close to 1. Can affect MDP (ex. 188).

$y_{i,t} \approx r(s_{i,t}, a_{i,t}) + \gamma \hat{V}_\theta^\pi(s_{i,t+1})$

SGD online a-c:
1. $a \sim \pi_\theta(a|s)$, (s, a, s', r)
2. $\hat{V}_\theta^\pi \leftarrow r + \gamma \hat{V}_\theta^\pi(s')$
3. $\hat{A}^\pi(s, a) = r(s, a) + \gamma \hat{V}_\theta^\pi(s') - \hat{V}_\theta^\pi(s)$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a|s) \hat{A}^\pi(s, a)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

In practice: "multi-threading" to get multi-batch

Q-learning: RL without policy gradient.

parameters: dataset size N, iterations K, gradient steps S

$\arg\max_{a_t} A^\pi(s_t, a_t)$: best action from s_t. → $\pi'(a_t|s_t) = \begin{cases} 1 \text{ if } a_t = \arg\max_{a_t} A^\pi(s_t, a_t) \\ 0 \text{ otherwise} \end{cases}$ (probably better than $\pi(a_t|s_t)$)

→ fit A^π (or Q^π / V^π) (or estimate V^π).

generate samples ↔ $\pi \leftarrow \pi'$ ← policy iteration algorithm.

$$A^\pi(s, a) = r(s, a) + \gamma E[V^\pi(s')] - V^\pi(s)$$

$$V^\pi(s) \leftarrow r(s, \pi(s)) + \gamma E_{s' \sim p(s'|s, \pi(s))}[V^\pi(s')]$$

← one caveat: curse of dimensionality. → use NN.

learn out $V^\pi(s)$ (unstructured)

larger buffer = more stability.

gradient descent

with DP tricks, value iteration algorithm:
1. set $Q(s, a) \leftarrow r(s, a) + \gamma E[V(s')]$
2. set $V(s) \leftarrow \max_a Q(s, a)$

"epsilon-greedy" exploration: $\pi(a_t|s_t) = \begin{cases} 1-\epsilon \text{ if } a_t = \arg\max \\ \epsilon / (|A| - 1) \text{ otherwise} \end{cases}$

"Boltzmann exploration" $\pi(a_t|s_t) \propto \exp(Q_\phi(s_t, a_t))$ ← proportional.

→ can use replay buffer to sample i.i.d. (similar to batch)

"classic" DQN algorithm (use in HW#4)

1. take some action a_i and observe (s_i, a_i, s_i', r_i), add to $\mathcal{B}$
2. sample mini-batch $\{s_j, a_j, s_j', r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{a_j'} Q_{\phi'}(s_j', a_j')$ using target network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_j, a_j) (Q_\phi(s_j, a_j) - y_j)$; 5. update ϕ': copy ϕ every N steps

$K = 1$

$\approx \max_{a'} Q_\phi(s_i', a_i')$

1. $y_i \leftarrow \max_{a_i} (r(s_i, a_i) + \gamma E[V_\phi(s_i')])$
2. $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|V_\phi(s_i) - y_i\|^2$