Reinforcement Learning with Continuous Actions

recall:

state space S reward space R

action space A

1) discrete state space S

e.g. gridworlds, etc.

a) $|S|$ is manageable numerically

(i.e. all states can be visited during training)

→ use tabular RL methods:

e.g. Q-table, MC-methods

(first part of RL lecture)

b) $|S|$ is too large

→ use function approximation, e.g. NN

(Chapter 9 in Sutton & Barto)

- goal is to learn the value function of unseen states

2) continuous state space

e.g. qubit example $S = \{(\theta, \varphi) \mid \theta \in [0, \pi), \varphi \in [0, 2\pi)\}$

- cannot explore all states (uncountably many)
  => function approx. methods required

similarly, A can be either discrete or continuous:

1) discrete action space $A = \{a_1, \ldots, a_{|A|}\}$

e.g. moves along Gridworld $\{S, N, E, W\}$

application quantum gates $\{U_0, U_x, U_y, U_z\}$

note: the number of trajectories τ in an episodic task of T steps is $|A|^T$, i.e. exponential

problem: depending on the sparsity of rewards, exploring a large trajectory space can be infeasible

a) $|\mathcal{A}| \lesssim 30$

use PG methods, AC methods, DQN, etc.

b) $|\mathcal{A}| \sim 100$

advantageous to use guided exploration,

e.g. Monte Carlo Tree Search (MCTS)

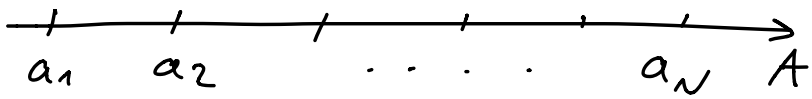
[Chapter 8, Sutton & Barto → planning]

→ AlphaZero algorithm

2) <u>continuous action space</u> $\mathcal{A}$:

e.g. external electric/magnetic field, force, torque

a) discretize action space $\mathcal{A}$

e.g. 

works well if $\dim \mathcal{A} \leq 2$

e.g. if $\mathcal{A} = [0,1]^3$, i.e. the unit cube

use linear grid of 10 points per dimension

⇒ a total of 10^3 actions for $\mathcal{A} = [0,1]^3$

b) goal: find RL algorithms to handle continuous A

- Policy Gradient Methods (incl. actor-critic)

$\pi_\theta(a|s) : A \times S \longrightarrow [0,1]$

should parametrize a continuous distribution over action space A

I) unbounded action spaces:

e.g. $A = \mathbb{R}$ for $d=1$

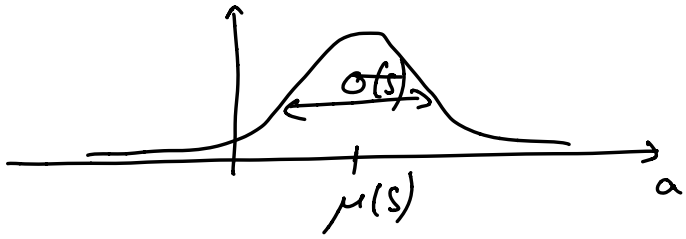
simplest possible cont. distr.: Gaussian

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi\sigma_\theta^2(s)}} \exp\left[-\frac{(a-\mu_\theta(s))^2}{2\sigma_\theta^2(s)}\right]$$

uniquely fixed by its first & second moments:

$\mu_\theta : S \longrightarrow \mathbb{R}$ mean

$\sigma_\theta : S \longrightarrow [0,\infty)$ variance



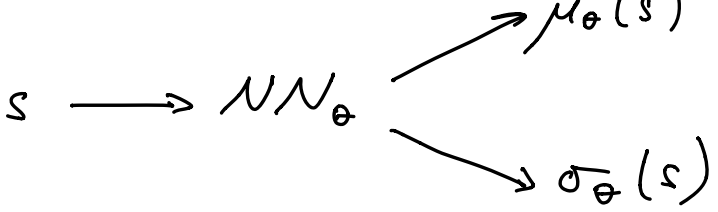
Note: $\mu(s)$, $\sigma(s)$ are state-dependent

- every state s has its own policy distr. to learn

idea: find variational approximations to

$$\mu(s), \sigma(s) \longrightarrow \mu_\theta(s), \sigma_\theta(s)$$

e.g. NN:



- having $\mu_\theta(s)$ & $\sigma_\theta(s)$, sample an action a from Gaussian policy

$$\pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta^2(s))$$

problem: Gaussian distr. has unbounded support

$\Rightarrow$ action can take any value in $(-\infty, \infty)$

what if we need bounded continuous actions?

II) bounded continuous action spaces:

e.g. $A = [0, 1]$, or $A \in [a, b]$

idea: parametrize π_θ by a distribution with compact support

caveat: we have to be able to sample actions from it easily:

eg. $B(a; \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} a^{\alpha-1}(1-a)^{\beta-1}$

Beta distribution with params α, β

where $\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx$ Gamma function

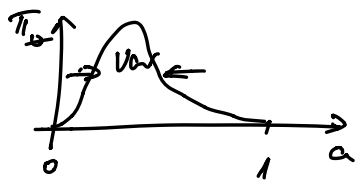
(continuous generalization of the factorial $n!$)

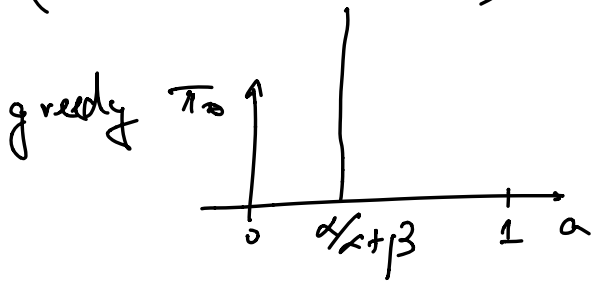
bounded support:

i) $[0,1]$ for $\alpha, \beta > 1$

ii) $(0,1)$ for $0 < \alpha, \beta < 1$

$\pi_\theta(a|s) = B(a; \alpha_\theta(s), \beta_\theta(s))$





• properties:

mean: $\mathbb{E}_{a\sim B(a;\alpha,\beta)}[a] = \frac{\alpha}{\alpha+\beta}$ → use for greedy evaluation

variance: $\mathrm{Var}_{a\sim B(a;\alpha,\beta)}[a] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$

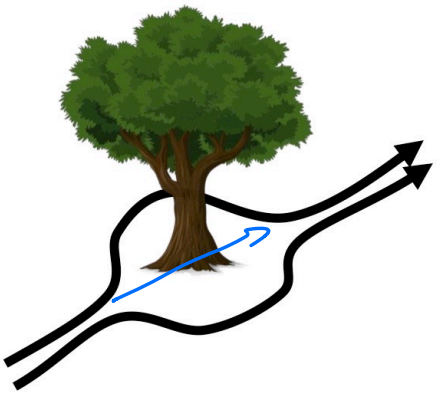
• in PG methods:

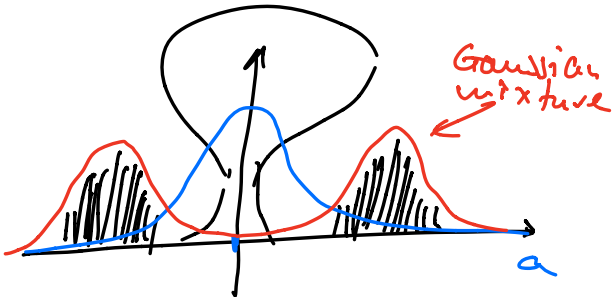
$\alpha \rightarrow \alpha_\theta(s)$
$\beta \rightarrow \beta_\theta(s)$
} can be variationally approximated by a NN

reference: by Chou et al.

"Improving stochastic policy gradients in continuous control with deep reinforcement learning using Beta distribution".

Problem: unimodality (for Gaussian policy)





way out: 1) Gaussian mixture models
→ lecture 1, notebook 1

2) normalizing flow models, etc.
Tang & Agrawal arXiv: 1809.10326

- Q-learning with Continuous Actions:

recall: DQN involves the operations:

$$\pi(a|s) = \begin{cases} 1, & \text{if } a = \underset{a}{\operatorname{argmax}}\, Q_\varphi(s,a) \\ 0, & \text{otherwise} \end{cases}$$

target values: $y_j = r_j + \gamma \max_{a'} Q_\varphi(s',a')$

→ Q-learning contains a maximization procedure over actions

problem: max, argmax require separate optimization procedure in the case of continuous actions!

→ 3 options to compute max for A continuous

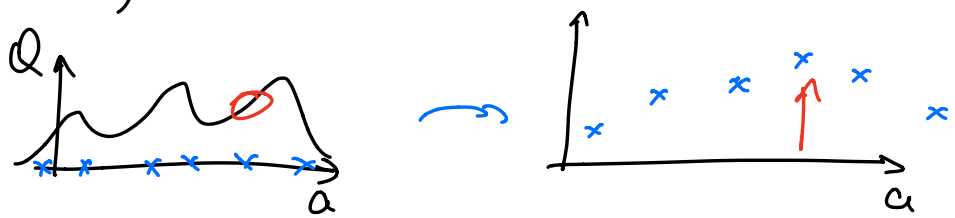
Option 1:

1) gradient-based opt., e.g. SGD
   → may be slow (max, argmax operations are in innermost loop of DQN)

2) stochastic optimization: $|A| \sim 30$
   → works well for low-dimensional A spaces

$$\max_a Q_\theta(s,a) \approx \max \{ Q_\theta(s,a_1), Q(s,a_2), \dots, Q(s,a_N) \}$$

   i) select random actions $a_1, \dots, a_N$ (e.g. uniformly) from A
   ii) evaluate $Q(s,\cdot)$ at $a_1, \dots, a_N$
   iii) take discrete max



- fast, might work
- not very accurate

→ extension: cross-entropy method
(→ Berkeley lectures)

Option 2:

→ use function classes for Q which allow to analytically compute the max

• normalized advantage functions (NAFs)

$$Q_\varphi(s,a) = -\frac{1}{2}(a-\mu_\varphi(s))^t P_\varphi(s)(a-\mu_\varphi(s)) + V_\varphi(s)$$

$$s \to DNN_\varphi \begin{cases} \to \mu_\varphi \in \mathbb{R}^d \\ \to P_\varphi \in \mathbb{R}^{d\times d} \\ \to V_\varphi \in \mathbb{R} \end{cases}, \quad d = \dim(A)$$

note:

$\operatorname{argmax}_a Q_\varphi(s,a) = \mu_\varphi(s)$

$\max_a Q_\varphi(s,a) = V_\varphi(s)$

} max & argmax known analytically → no need to compute them!

• downside: Q-function is quadratic in a → lose representational power (exact Q-function may not be quadratic in a)

Option 3

→ use a function approximator to learn doing the maximization

- algorithm: Deep Deterministic Policy Gradient (DDPG)
  → mishmosher

recall: notation for deterministic policies

$$\pi: S \to A, \quad a = \pi(s)$$

idea: train a deterministic policy network $\mu_\theta(s)$, s.t.

$$\mu_\theta(s) \approx \arg\max_{a \in A} Q_\phi(s,a)$$

- how do we find the optimal params θ?

solve: $\theta \leftarrow \arg\max_\theta Q_\phi(s, \mu_\theta(s))$

↑ held fixed

$$\nabla_\theta Q_\phi(s, \mu_\theta(s)) = \nabla_\theta \mu_\theta \frac{dQ_\phi}{da}$$

done automatically in any auto-diff package, e.g. JAX

- new target for DQN algorithm

recall: $\max_a Q(s,a) = Q\left(s, \underbrace{\operatorname{argmax}_a Q(s,a)}_{\approx \mu_\theta(s)}\right)$

$$y_j \overset{\text{def}}{=} r_j + \gamma Q_\varphi\left(s'_j, \operatorname{argmax}_{a'} Q_\varphi(s'_j, a'_j)\right)$$

$$\approx r_j + \gamma Q_\varphi\left(s'_j, \mu_\theta(s'_j)\right)$$

- pseudo code (DDPG)

1. take action a_j using some policy, observe (s_j, a_j, r_j, s'_j); add it to buffer B.
2. sample minibatch from B uniformly
3. compute $y_j = r_j + \gamma Q_{\varphi'}(s'_j, \mu_{\theta'}(s'_j))$ using target nets: $Q_{\varphi'}, \mu_{\theta'}$
4. $\varphi \leftarrow \varphi - \alpha \sum_j \nabla_\varphi Q_\varphi(s_j, a_j)\left(Q_\varphi(s_j, a_j) - y_j\right)$
5. update deterministic policy params:
$\theta \leftarrow \theta + \beta \sum_j \nabla_\theta Q_\varphi(s_j, \mu_\theta(s_j))$
6. updat target nets φ', θ' (e.g. Polyak averaging)