RL using Function Approximation_ For continuous space

1. Q-function Approximation

Background & Basic Idea

In real-life control, X, U continuous

- approximate Q-function $\hat{Q}$ must be used

Method 1: use linearly parametrized approximation

Q-function Approximation

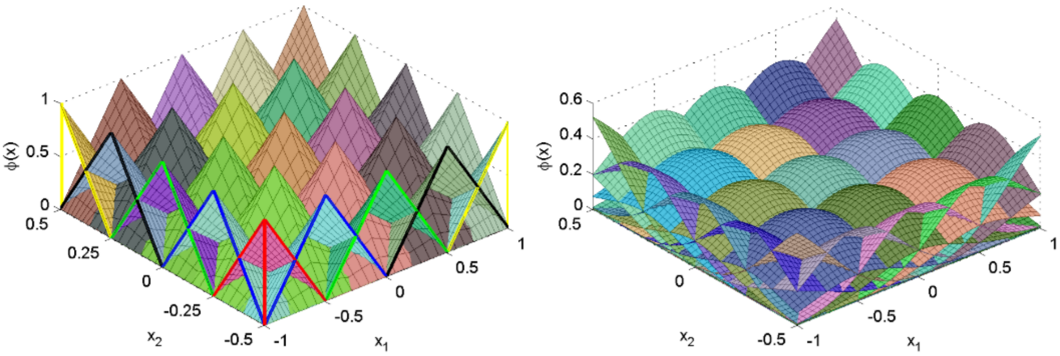
- use **Basis Function** to approximate Q-function in a continuous space

$$\widehat{Q} = \sum_{i=1}^{N} \theta_i \phi_i(x,u)$$

$$\phi_i(x,u) : X \times U \mapsto \mathbb{R}$$

usually normalized: $\sum_i \phi_i(x) = 1$

- E.g., fuzzy approximation, RBF network approximation



- Policy is greedy in $\hat{Q}$, computed on demand for given x:

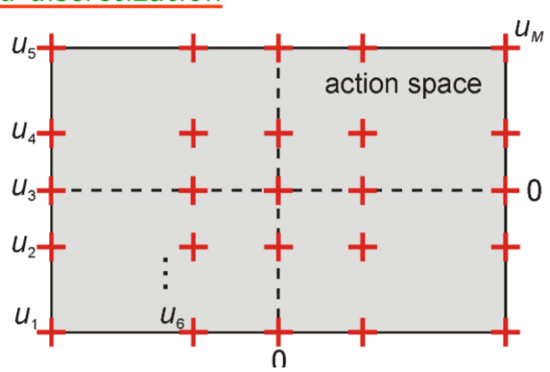
  $\pi(x) = \arg\max_u \widehat{Q}(x,u)$

  - Approximator must ensure efficient arg max solution

Action Space Discretization

- Choose M discrete actions $u_1, \cdots, u_M \in U$
- Solve "arg max" by explicit enumeration
- In a control problem, we always make the grid more detail arround the attractor, for example, 0 in the graph

Example: grid discretization



Fuzzy Q-iteration

Model

Given:

- N basis functions $\phi_1, \cdots, \phi_N$
- M discrete actions $u_1, \cdots, u_M$

Store

- N ×M matrix of parameters θ (one for each pair basis function–discrete action)
    - same row: same ϕ
    - same column: same action

$$\widehat{Q}^{\theta}(x, u_j) = \sum_{i=1}^{N} \phi_i(x)\theta_{i,j} = [\phi_1(x) \ldots \phi_N(x)] \begin{bmatrix} \theta_{1,j} \\ \vdots \\ \theta_{N,j} \end{bmatrix}$$

Policy

$$\widehat{\pi}^*(x) = \underset{u_j, j=1,\ldots,M}{\arg\max}\, \widehat{Q}^{\theta^*}(x, u_j)$$

$$(\theta^* = \text{ converged parameter matrix })$$

Fuzzy Q-iteration

Fuzzy Q-iteration

repeat at each iteration ℓ
  for all cores x_i, discrete actions u_j **do**
    $\boxed{\theta_{\ell+1,i,j} = \rho(x_i, u_j) + \gamma \max_{j'} \widehat{Q}^{\theta_\ell}(f(x_i, u_j), u_{j'})}$
  end for
until convergence

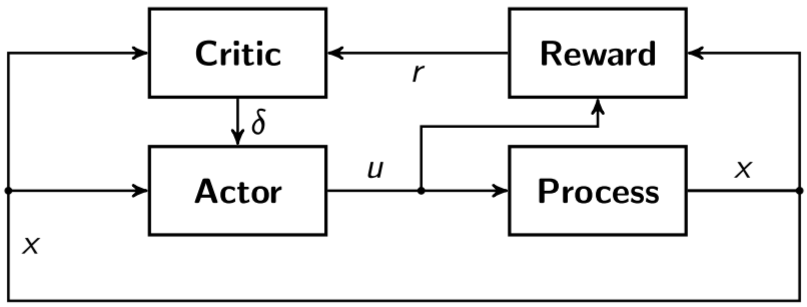
weighted sum

2. Actor-critic methods

Structure

Explicitly separated value function and policy

- Actor = control policy $\pi(x)$
- Critic = state value function $V(x)$



Actor-critic

for every trial **do**
  initialize x_0, choose initial action $u_0 = \tilde{u}_0$
  repeat for each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    choose **next** action $u_{k+1} = \hat{\pi}(x_{k+1}, \varphi_k) + \tilde{u}_{k+1}$
    $\Delta_k = r_{k+1} + \hat{V}(x_{k+1}, \theta_k) - \hat{V}(x_k, \theta_k)$
    $\theta_{k+1} = \theta_k + \alpha_c \Delta_k \left. \frac{\partial \hat{V}(x,\theta)}{\partial \theta} \right|_{\substack{x=x_k \\ \theta=\theta_k}}$
    $\varphi_{k+1} = \varphi_k + \alpha_a \Delta_k \tilde{u}_k \left. \frac{\partial \hat{\pi}(x,\varphi)}{\partial \varphi} \right|_{\substack{x=x_k \\ \varphi=\varphi_k}}$
  until terminal state
end for

When facing continuity:

- Actor parameterized in $\phi : \hat{\pi}(x; \phi)$
- Critic parameterized in $\theta : \hat{V}(x; \theta)$

Parameters ϕ and θ, have finite size, but approximate functions on continuous (infinitely large) spaces

Update Critic: Value Estimation

The task of the critic is to **predict the expected future reinforcement r the process will receive being in the current state and following the current control policy.**

- For doing that, we need to train critic, the prediction error is always a train input:

  Use sample $(x_k; u_k; x_{k+1}; r_{k+1})$ at each step k and parameterized V:

  Note that both $\hat{V}(s_k)$ and $\hat{V}(s_{k+1})$ are known at time k, since $\hat{V}(s_{k+1})$ is a prediction obtained for the current process state: For example, assuming a NN, we can always get a prediction of next state value based on current state, we get the Δ from **real system feedback** r_{k+1} and **critic originial prediction**

$$\Delta_k = V\left(s_k\right) - \hat{V}\left(s_k\right) = r_{k+1} + \gamma \hat{V}\left(s_{k+1}\right) - \hat{V}\left(s_k\right)$$

- For example, Let the critic be represented by a neural network or a fuzzy system, in my opinion, **the index of θ represents the iteration version, it accidentally equal to state/step number, because each step we make a update. It does not mean the corresponding θ of state k or k+1**

  $\hat{V}\left(s_{k+1}\right) = \hat{V}\left(s_{k+1}, \theta_k\right)$

  To update θ_k, a gradient-descent learning rule is used:

$$\theta_{k+1} = \theta_k + \alpha_c \Delta_k \frac{\partial \hat{V}\left(s_k, \theta_k\right)}{\partial \theta_k}$$
$$\alpha_c > 0, \text{learning rate of critic}$$

$$\Delta_k > 0, \text{ i.e., } r_{k+1} + \gamma \hat{V}^\pi\left(x_{k+1}, \theta_k\right) > \hat{V}^\pi\left(x_k, \theta_k\right)$$
$$\Rightarrow \text{old estimate too low, increase } \hat{V}$$

$$\Delta_k < 0, \text{ i.e., } r_{k+1} + \gamma \hat{V}^\pi\left(x_{k+1}, \theta_k\right) < \hat{V}^\pi\left(x_k, \theta_k\right)$$
$$\Rightarrow \text{old estimate too high, decrease } \hat{V}$$

Update Actor: Policy Update

The actor (i.e., the policy) can be adapted in order to **establish an optimal mapping between the system states and the control actions.**

$$u_k = \hat{\pi}\left(x_k, \varphi_k\right) + \tilde{u}_k, \hat{\pi} = \text{ actor, } \tilde{u}_k = \text{ exploration}$$

$$\varphi_{k+1} = \varphi_k + \alpha_a \Delta_k \tilde{u}_k \left.\frac{\partial \hat{\pi}(x, \varphi)}{\partial \varphi}\right|_{x=x_k}$$

$$\Delta_k > 0, \text{ i.e., } r_{k+1} + \gamma \hat{V}^\pi\left(x_{k+1}, \theta_k\right) > \hat{V}^\pi\left(x_k, \theta_k\right)$$
$$\Rightarrow \widetilde{u}_k \text{ had positive effect, move in that direction}$$

$$\Delta_k < 0, \text{ i.e.}, r_{k+1} + \gamma \hat{V}^\pi(x_{k+1}, \theta_k) < \hat{V}^\pi(x_k, \theta_k)$$
$$\Rightarrow \widetilde{u}_k \text{ had negative effect, move away from that direction}$$

3. One Really Classical Example

Lecture Notes, Application 1