Aprendizado Por Reforço

Introdução

Framework

Historico e Estados

Outros componentes

Markov Reward Process

Função de Valor

Markov Decision Process

Política

Função de valor

Diagrama de Backup

Função de valor ótimo

Estratégias de Aprendizado

Programação Dinâmica

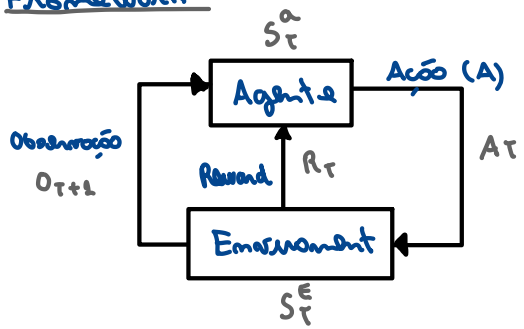
Q-learning

Introdução

Aprendizado de Máquina
- supervisionado (target): KNN, regressão, MLP, árvores
- não supervisionado (dados): SOM, K-means, K-modes
- aprendizado por reforço (recompensa)

Exemplos:
* Self driving car
* chatbot
* gomoku

Framework



goal: Maximizar a recompensa acumulada

Exemplos:
* Atari
  - + reward se aumentar o score
  - − reward se diminuir o score
* Finanças
  - + reward se ganhar dinheiro
  - − reward se perder dinheiro
* Power station
  - + reward se produzir energia de forma segura
  - − reward se exceder a produção segura do reator

Histórico e Estado

$H_t = [(O_1, A_1, R_1), \dots, (O_{t-1}, A_t, R_t)]$

$S_t = f(H_t)$

devemos usar todo o histórico?

* Maior parte dos problemas em RL são modelados como cadeias de Markov

Definição: S_t é Markov se

$$P[S_{t+1} | S_t, S_{t-1}, \dots, S_1] = P[S_{t+1} | S_t]$$

Não precisamos armazenar o histórico

Outros Componentes

Agente
- Política (mapeia $s \to a$)
  - determinística: $a = \pi(s)$
  - estocástica: $\pi(a|s) = P[A_t = a | S_t = s]$
- função de valor: avalia os estados $V_\pi(s)$
- Modelo: Prediz o que o environment vai fazer após o agente tomar a decisão
  - Model Free
    - Value Based
    - Policy Based
    - Actor critic: policy + value
  - Model Based: Model + Policy and/or value function

Markov Reward Process

É uma tupla $\langle S, P, R, \gamma \rangle$
* S: States
* P: Matriz de transição
* R: função de recompensa, $R_s = E[R_{t+1} | S_t = s]$
* γ: fator de desconto, $\gamma \in [0, 1]$

Retorno:

total descontado da reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Função de Valor

$V(s)$ é o valor esperado de retorno iniciando no estado s

$$V(s) = E[G_t | S_t = s]$$

Função de valor (cont.)

Seja $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ e $v(s) = E[G_t | S_t = s]$,

a função de valor pode ser decomposta

em duas partes:

* Recompensa Imediata
* Valores com desconto dos sucessivos estados

Veja:

$$v(s) = E[G_t | S_t = s]$$
$$E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$
$$E[R_{t+1} + \gamma (\underbrace{R_{t+2} + \gamma R_{t+3} + \dots}_{G_{t+1}}) | S_t = s]$$
$$E[\underbrace{R_{t+1}}_{\text{Imediate reward}} + \gamma \underbrace{v(s')}_{\text{discounted value of } s' \text{ state}} | S_t = s]$$

Chamamos isso de Bellman Equation:

$$v(s) = R_s + \gamma P_{ss'} v(s')$$

Para resolver basta:

$$v = (I - \gamma P)^{-1} R$$

* Complexidade $O(n^3)$
* Métodos Interativos → Monte Carlo
  → Programação dinâmica
  → Temporal-Difference learning

Markov Decision Process

* MDP é uma MRP com decisão (actions)
* MDP é uma tupla $\langle S, A, P, R, \gamma \rangle$
  * S: conjunto de estados
  A: conjunto de ações
  P: matriz de transição
  R: função de recompensa $R_s^a = E[R_{t+1} | S_t = s, A_t = a]$
  γ: fator de desconto, $\gamma \in [0, 1]$

Política

$$\pi(a|s) = P[A_t = a | S_t = s]$$

Função de valor:

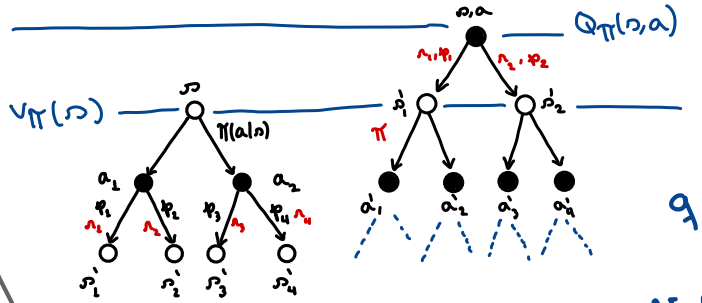
State value function: seguindo uma política π, s

$$v_\pi(s) = E[G_t | S_t = s]$$

Action value function: seguindo uma política π, s, a

$$q_\pi(s,a) = E[G_t | S_t = s, A_t = a]$$

Diagrama de backup: Ajuda p/ montar as equações



$$q_\pi(s,a) = \sum_{s'} p(s'|s,a)[r + \gamma v_\pi(s')]$$

$$v_\pi(s) = \sum_a \pi(a|s)\, q_\pi(s,a)$$

Função de Valor Ótima

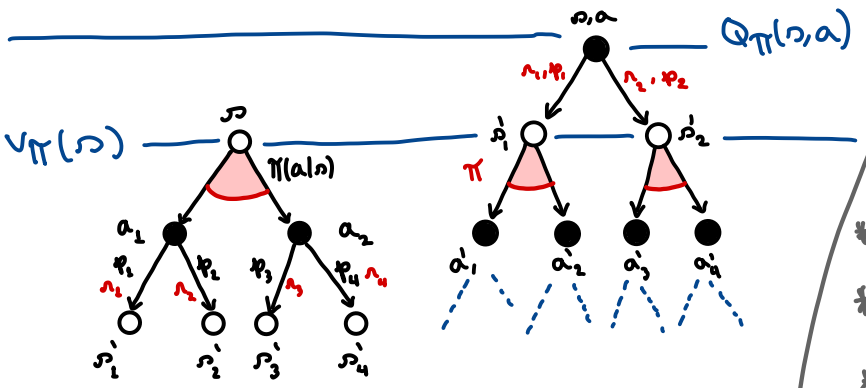
Existe uma política que maximiza a função de valor:

$$v_*(s) = \max_\pi v_\pi(s)$$

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

A melhor política pode ser encontrada a partir de $q(s,a)$

$$\pi_* = \begin{cases} 1 & \text{se } a = \text{argmax}(q_*(s,a)) \; \forall a \in A \\ 0 & \text{caso contrário} \end{cases}$$



$$v_*(s) = \max\{q_*(s,a)\}$$

Como encontrar a melhor Política

- métodos
  - Programação dinâmica
    - Policy Iteraction
    - Value Iteraction
  - RL Algo.
    - TD-Learning
    - Q-Learning

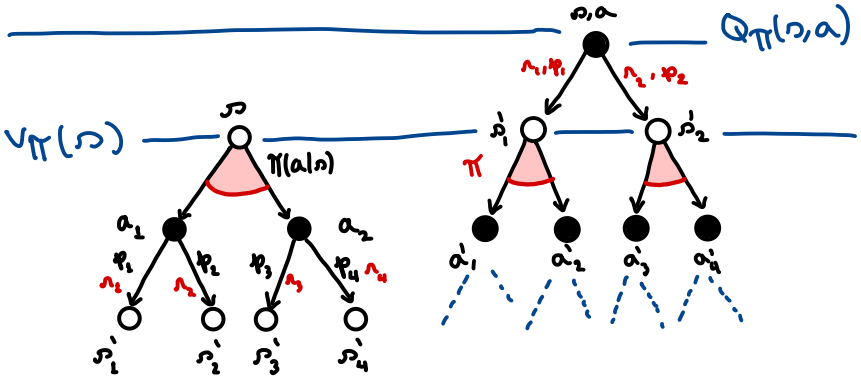
Programação Linear

* Divide em problema em sub problemas
* É recursivo
* Preciso conhecer o Environment (Model based)
* Usa Bellman eq. properties

Q-Learning

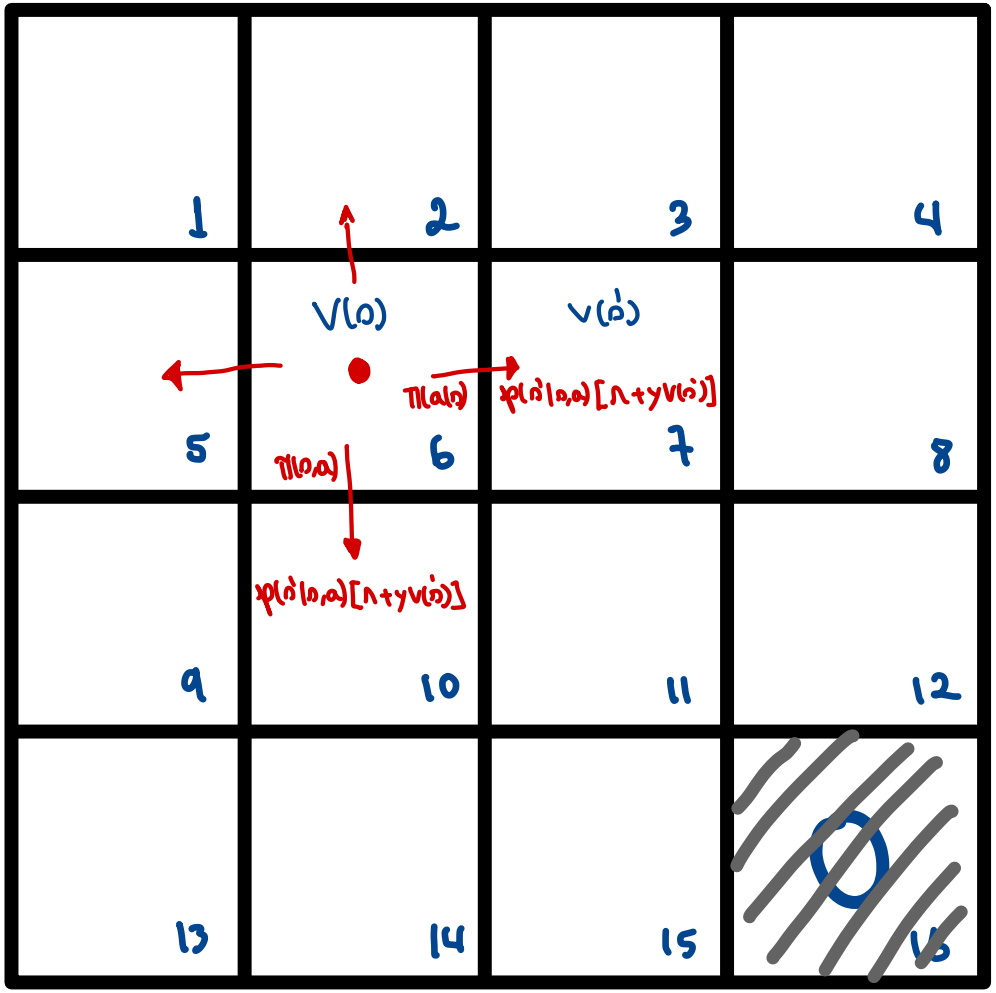
* Exploration e exploitation
* Uso o $Q(s,a)$ para encontrar a melhor opção
* Model Free, não preciso conhecer o environment
* mais famoso em RL.

Programação Dinâmica



while $\Delta > \epsilon$:
  calcular v
  $\Delta = |v(s) - v|$

Grid World



Policy Iteration

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

Value Iteration

$$V(s) = \max_a \left\{ \sum_{s'} p(s'|s,a)[r + \gamma V(s')] \right\}$$

Policy	×	value
faster		slow
close		converge to optimal π
—		more interactions