Învățare Automată - Laboratorul 5

Algoritmul Q-Learning

Tudor Berariu

Facultatea de Automatică și Calculatoare

Scopul laboratorului

- Scopul laboratorului îl reprezintă înțelegerea și implementarea algoritmului Q-Learning.

Amintiri din laboratorul 4

- În cadrul laboratorului 4 ați implementat doi algoritmi de programare dinamică pentru găsirea unei politici optime pentru un MDP.
- Algoritmii **Policy Iteration** și **Value Iteration** presupuneau o cunoaștere completă a dinamicii mediului.
- În majoritatea problemelor reale dinamica mediului nu este cunoscută, iar agentul trebuie să estimeze câștigurile și să își îmbunătățească politica pe baza interacțiunii directe cu mediul.

Valoarea acțiunii într-o stare

- Câștigul mediu adus de o acțiune a dintr-o stare s indus de o politică π:

$$\begin{aligned} q^{\pi}(s,a) &= \mathbb{E}_{\pi}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \middle| S_t = s, A_t = a\right] \\ &= \mathbb{E}_{\pi}\left[\sum_{\tau=t+1} \gamma^{\tau-t-1} R_{\tau} \middle| S_t = s, A_t = a\right] = \mathbb{E}_{\pi}\left[G_t | S_t = s, A_t = a\right] \end{aligned}$$

- Relațiile dintre valorile q induse de o politică deterministă (ecuațiile Bellman):

$$\begin{aligned} q^{\pi}(s,a) &= \mathbb{E}_{\pi}\left[R_{t+1} + \gamma q^{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right] \\ &= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a)\left[r + \gamma q^{\pi}(s', \pi(s'))\right] \end{aligned}$$

Algoritmul Q-learning

- Agentul învață prin interacțiune cu mediul observând consecințele acțiunilor pe care le ia.

procedure ϵ-GREEDY(s, q, ϵ)
  cu probabilitate ϵ: **return** *random*($\mathcal{A}$)
  cu probabilitate $1 - \epsilon$: **return** $\operatorname{argmax}_{a \in \mathcal{A}} q(s, a)$
end procedure

procedure Q-LEARNING($\langle \mathcal{S}, \mathcal{A}, \gamma \rangle, \epsilon$)
  for all $s \in \mathcal{S}, a \in \mathcal{A}$ **do**
    $q(s, a) \leftarrow 0$ ▷ Valorile inițiale sunt zero.
  end for
  for all episodes **do**
    $s \leftarrow$ stare inițială
    while s nu este stare finală **do**
      alege acțiunea a conform ϵ-Greedy(s, q, ϵ)
      execută a și observă recompensa r și noua stare s'
      $q(s, a) \leftarrow q(s, a) + \alpha \left(r + \gamma \max_{a'} q(s', a') - q(s, a) \right)$
      $s \leftarrow s'$
    end while
  end for
  for all $s \in \mathcal{S}$ **do** ▷ Construim politica lacomă în raport cu q.
    $\pi(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} q(s, a)$
  end for
  return π
end procedure

Algoritmul Q-learning

- Agentul învață prin interacțiune cu mediul observând consecințele acțiunilor pe care le ia.
- Valorile q sunt ajustate prin diferențe temporale.

procedure ϵ-GREEDY(s, q, ϵ)
  cu probabilitate ϵ: **return** *random*($\mathcal{A}$)
  cu probabilitate $1 - \epsilon$: **return** $\underset{a \in \mathcal{A}}{\text{argmax}}\, q(s, a)$
end procedure

procedure Q-LEARNING($\langle \mathcal{S}, \mathcal{A}, \gamma \rangle, \epsilon$)
  for all $s \in \mathcal{S}, a \in \mathcal{A}$ **do**
    $q(s, a) \leftarrow 0$ ▷ Valorile inițiale sunt zero.
  end for
  for all episodes **do**
    $s \leftarrow$ stare inițială
    while s nu este stare finală **do**
      alege acțiunea a conform ϵ-Greedy(s, q, ϵ)
      execută a și observă recompensa r și noua stare s'
      $q(s, a) \leftarrow q(s, a) + \alpha \left(r + \gamma \max_{a'} q(s', a') - q(s, a)\right)$
      $s \leftarrow s'$
    end while
  end for
  for all $s \in \mathcal{S}$ **do** ▷ Construim politica lacomă în raport cu q.
    $\pi(s) \leftarrow \underset{a \in \mathcal{A}}{\text{argmax}}\, q(s, a)$
  end for
  return π
end procedure

Algoritmul Q-learning

- Agentul învață prin interacțiune cu mediul observând consecințele acțiunilor pe care le ia.
- Valorile q sunt ajustate prin diferențe temporale.
- Învățarea este off-policy
  - politica învățată este lacomă
  - politica de joc permite explorare

procedure ϵ-GREEDY(s, q, ϵ)
  cu probabilitate ϵ: **return** *random*($\mathcal{A}$)
  cu probabilitate $1-\epsilon$: **return** $\operatorname{argmax}_{a \in \mathcal{A}} q(s, a)$
end procedure

procedure Q-LEARNING($\langle \mathcal{S}, \mathcal{A}, \gamma \rangle, \epsilon$)
  for all $s \in \mathcal{S}, a \in \mathcal{A}$ **do**
    $q(s, a) \leftarrow 0$ ▷ Valorile inițiale sunt zero.
  end for
  for all episodes **do**
    $s \leftarrow$ stare inițială
    while s nu este stare finală **do**
      alege acțiunea a conform ϵ-Greedy(s, q, ϵ)
      execută a și observă recompensa r și noua stare s'
      $q(s, a) \leftarrow q(s, a) + \alpha \left(r + \gamma \max_{a'} q(s', a') - q(s, a)\right)$
      $s \leftarrow s'$
    end while
  end for
  for all $s \in \mathcal{S}$ **do** ▷ Construim politica lacomă în raport cu q.
    $\pi(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} q(s, a)$
  end for
  return π
end procedure

Algoritmul Q-learning

- Agentul învață prin interacțiune cu mediul observând consecințele acțiunilor pe care le ia.
- Valorile q sunt ajustate prin diferențe temporale.
- Învățarea este off-policy
  - politica învățată este lacomă
  - politica de joc permite explorare

procedure ϵ-GREEDY(s, q, ϵ)
  cu probabilitate ϵ: **return** *random*($\mathcal{A}$)
  cu probabilitate $1 - \epsilon$: **return** $\underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q(s, a)$
end procedure

procedure Q-LEARNING($\langle \mathcal{S}, \mathcal{A}, \gamma \rangle, \epsilon$)
  for all $s \in \mathcal{S}, a \in \mathcal{A}$ **do**
    $q(s, a) \leftarrow 0$ ▷ Valorile inițiale sunt zero.
  end for
  for all episodes **do**
    $s \leftarrow$ stare inițială
    while s nu este stare finală **do**
      alege acțiunea a conform ϵ-Greedy(s, q, ϵ)
      execută a și observă recompensa r și noua stare s'
      $q(s, a) \leftarrow q(s, a) + \alpha \left(r + \gamma \max_{a'} q(s', a') - q(s, a)\right)$
      $s \leftarrow s'$
    end while
  end for
  for all $s \in \mathcal{S}$ **do** ▷ Construim politica lacomă în raport cu q.
    $\pi(s) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q(s, a)$
  end for
  return π
end procedure

Greuceanu și Balaurul

- Pe o hartă bidimensională se înfruntă Greuceanu și-un balaur. Greuceanu trebuie să găsească mărul fermecat înainte de a fi prins de balaur și înainte ca balaurul să calce pe măr.
- Balaurul simte direcția în care se află Greuceanu și se îndreaptă către el.
- Greuceanu câștigă jocul și primește 10 puncte dacă ajunge primul la mărul fermecat. Greuceanu pierde jocul dacă este prins de balaur sau dacă balaurul calcă pe măr. De asemenea, la fiecare moment de timp Greuceanu pierde câte 0.1 puncte. Dacă ajunge la -20 de puncte Greuceanu pierde jocul.

Abstractizarea jocului

- `get_initial_state(input_file)` primește calea unui fișier ce conține harta jocului și întoarce starea inițială.
- `get_valid_actions(state)` primește o stare și întoarce o listă de acțiuni ce pot fi executate în acea stare.
- `apply_action(state, action)` primește o stare și o acțiune și întoarce starea în care se ajunge, recompensa și un mesaj despre joc
- `is_final_state(state, score)` primește starea și scorul curente și verifică dacă episodul s-a încheiat.
- `display_state(state)` afișează starea curentă.

Cerințe

[6p] Implementați algoritmul Q-learning.

[2p] Implementați strategia ϵ-Greedy de selecție a unei acțiuni. Dacă există acțiuni ce nu au fost explorate, atunci se va alege aleator una dintre acelea. Altfel, cu o probabilitate ϵ se va alege aleator o acțiune, iar cu probabilitate $1 - \epsilon$ se va alege cea mai bună acțiune din starea respectivă.

[2p] Implementați rutine de evaluare a politicii lacome învățate.

[2p] Găsiți parametrii potriviți pentru o învățare cât mai rapidă pe toate cele trei hărți. Încercați și modificarea lui ϵ pe parcursul învățării.