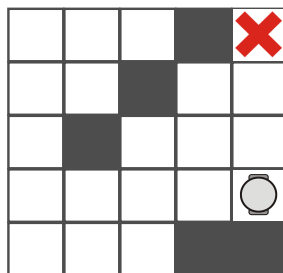


Control prin învățare – Laborator 2: Învățarea Q

Regulament – același ca și la laboratorul precedent, cu excepția termenului limită pentru predarea soluției, care este **20 aprilie**.

Introducere

Codul care formează baza laboratorului este același ca și la primul laborator. Codul poate fi descărcat de pe site-ul cursului, secțiunea “Practical assignments”. Dezarhivați codul într-un director de pe calculatorul dvs, navigați din MATLAB în acest director și rulați scriptul `startup`. În plus, un șablon pentru algoritmul de învățare Q, care poate opțional servi ca punct de plecare pentru implementarea din Partea 1, este disponibil în fișierul `qlearning.m`, împreună cu un script pentru calcularea funcției Q optimale, `gridnav_nearoptsol`, necesar în Partea 2.



Vom considera din nou problema de navigație 2D, în care un robot trebuie să găsească cea mai scurtă cale înspre o țintă (X roșu), evitând obstacolele (pătrate gri). Starea robotului este $x = [x_1, x_2]^T \in \{1, 2, 3, 4, 5\} \times \{1, 2, 3, 4, 5\}$; $x = [1, 1]^T$ reprezintă colțul stânga-jos în figură. Robotul se poate deplasa într-una dintre cele patru direcții cardinale, iar aceste patru acțiuni u sunt reprezentate prin numerele 1 (stânga), 2 (dreapta), 3 (jos), 4 (sus). Orice deplasare care rezultă în lovirea unui perete sau obstacol eșuează, și în acest caz robotul rămâne în poziția precedentă. Robotul primește o recompensă -0.1 la fiecare pas în care nu a atins ținta (inclusiv la coliziuni), și 10 când o atinge, scopul fiind atingerea țintei în cel mai scurt timp. Ținta este o stare terminală, așadar când ea este atinsă episodul se termină și robotul este resetat într-o poziție inițială. Reamintim funcțiile MATLAB care creează modelul problemei de navigație, (`gridnav_problem`), simulează tranzițiile și calculează recompensele (`gridnav_mdp`), și vizualizează grafic elementele problemei (`gridnav_visualize`).

Partea 1

Implementați învățarea Q cu explorare greedy. Algoritmul trebuie să ruleze pentru un număr dat de traiectorii (en. *trials*), T . Fiecare traiectorie se termină la atingerea stării țintă terminale, sau după un număr maxim de pași K . Această a doua limitare trebuie impusă pentru cazul în care starea terminală nu este atinsă într-un interval rezonabil.

Algoritmul trebuie implementat într-o funcție care acceptă la intrare parametrii experimentului și ai algoritmului de învățare Q: rata de învățare, probabilitatea de explorare, factorul de discount, numărul de traiectorii, numărul maxim de pași pe traiectorie etc.¹. Algoritmul trebuie să returneze (a) o **secvență**

¹O sugestie opțională: Pentru mai multă flexibilitate și simplitate, puteți folosiți o structură Matlab (`struct`) pentru a furniza toți acești parametri, în loc de variabile separate.

de funcții Q , conținând funcțiile Q găsite la sfârșitul fiecărei traiectorii, și (b) *returnurile cu discount* $R(x_0)$ acumulate de-a lungul fiecărei traiectorii. Fiecare return este calculat relativ la starea inițială x_0 a traiectoriei, iar valorile în traiectorii multiple sunt comparabile doar dacă x_0 este același pentru fiecare traiectorie. Suma de recompense, neputând fi infinită în practică, se oprește la ultimul pas al traiectoriei indiferent dacă starea terminală este atinsă sau nu. **[4p]**

Demonstrați că funcția creată funcționează corect aplicând-o pentru un discount γ între 0.9 și 0.99 la problema de navigație 2D. Valori sugerate pentru parametri: $T = 100$, $K = 1000$, $\alpha = 0.1$ (constant), $\varepsilon = 0.3$ (constant). Starea poate fi inițializată la o valoare care necesită traiectorii dificile și informative, de exemplu în căsuța cea mai depărtată de țintă. Vizualizați online poziția robotului în timp ce acesta învață. Opțional, vizualizați de asemenea funcțiile Q de la sfârșitul fiecărei traiectorii, precum și legile de control greedy corespunzătoare. Folosiți funcția `gridnav.visualize`. De reamintit că trebuie să refolosiți obiectul `view` creat. **[1p]**

Partea 2

Pentru a economisi timp, în experimentele din partea 2 dezactivați complet vizualizarea (experimentele vor rula mult mai rapid dacă vizualizarea nu trebuie actualizată).

Implementați o secvență de explorare care descrește exponențial. Mai exact, probabilitatea de explorare ε este inițializată la o valoare mare, de exemplu 1, ținută constantă de-a lungul fiecărei traiectorii, iar la sfârșitul unei traiectorii scăzută cu formula $\varepsilon \leftarrow \varepsilon \cdot d$ unde d este rata de scădere.

Încercați câteva rate de scădere în intervalul 0.9 – 0.99. Folosiți o rată de învățare $\alpha = 0.1$ constantă, $T = 100$ traiectorii, și $\gamma = 0.98$. Inițializați robotul tot timpul în aceeași stare. Pentru fiecare valoare a lui d , folosiți secvențele de funcții Q și de returnuri produse de algoritm pentru a crea două grafice:

1. Primul grafic arată distanța între funcțiile Q găsite la sfârșitul fiecărei traiectorii și funcția Q optimă. Graficul are numărul traiectoriei pe orizontală și distanța între funcțiile Q pe verticală. **[1.5p]** Pentru a găsi funcția Q optimă, folosiți scriptul `gridnav.nearoptsol`. Pentru a calcula distanța între două funcții Q , folosiți norma 2 transformând întâi tabelul Q într-un vector, vezi operatorul “:” și funcția `norm` în Matlab.
2. Al doilea grafic arată returnul obținut de robot de-a lungul fiecărei traiectorii. Acest grafic are numărul traiectoriei pe orizontală și returnul pe verticală. **[1.5p]**

Discutați care rată de scădere ar fi mai bună: pentru precizia funcției Q , pentru return, și pentru un compromis rezonabil între ambele. Explicați relația între cele două grafice și motivul pentru care intervine acest compromis. **[2p]**

În raport, descrieți pe scurt problemele pe care le-ați rezolvat, alegerile semnificative pe care le-ați făcut în implementarea algoritmilor (nu descrieți codul linie cu linie – codul complet trebuie inclus oricum în raport), și mai în detaliu studiile cerute. Nu uitați mai ales să includeți studiul și discuțiile din partea 2, cu graficele cerute pentru valorile alese ale ratei de scădere.