UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Reinforcement Learning

Inteligență Artificială

Autor: Man Cristina
Grupa: 30233

Facultatea de Automatică
și Calculatoare

18 Ianuarie 2024

Cuprins

1 Question 1 - Value Iteration **2**

2 Question 2 - Bridge Crossing Analysis **2**

3 Question 3 - Policies **3**

4 Question 4 - Prioritized Sweeping Value Iteration **4**

5 Question 5 - Q-Learning **4**

6 Question 6 - Epsilon Greedy **5**

7 Question 7 - Bridge Crossing Revisited **7**

8 Question 8 - Q-Learning and Pacman **8**

9 Question 9 - Approximate Q-Learning **8**

10 Bibliografie **9**

1 Question 1 - Value Iteration

Cerința acestui punct este implementarea unui agent de tip *value iteration*, mai specific, un *planner offline* pentru care este relevant numărul de iteratii în care se recurge la *value iteration*. Această formă de învățare pornește de la un MDP (Markov Decision Process) și are loc pe baza ecuațiilor Bellman, cu ajutorul cărora se calculează valorile optimale aferente fiecărei stări. Practic, valoarea fiecărei stări este egală cu maximul dintre Q-value-urile care derivă din acea stare.

Un MDP constă într-un set de stări, un set de acțiuni, funcții de tranziție (probabilitatea că acțiunea a va face cadin starea s să se ajunge în starea s'), funcții de recompensă, o stare inițială și o stare terminală. Scopul este de a găsi o politică optimă, anume câte o acțiune pentru fiecare stare. Un alt factor important pentrua acest tip de probleme este discount-ul, acesta determinând importanța unei recompense în funcție de cât de depărtată este de starea finală (în funcție de problema, se poate ca recompensele îndepărtate să nu conteze atât de mult sau, din contra, să fie luate în considerare).

Algoritmul propus pentru această cerință a fost implementat în funcția *runValueIteration()*, însă a fost nevoie și de implementarea funcției *computeQValueFromValues(state, action)*, care are rolul de a returna Q-value-ul perechii (state, action), returnată de *self.values*.

Numărul de repetiții ale algoritmul corespunzător ecuației Bellman din funcția *runValueIteration()* corespunde numărului de iterații introduse în consola. În cadrul fiecărei iterații, se analizează fiecare stare existentă în MDP, iar pentru fiecare stare se procesează fiecare acțiune care poate fi efectuată. Se va reține *Q-value*-ul maxim al perechilor (stare, acțiune) pentru fiecare stare, iar aceste valori vor fi reținute în *self.values*.

Calcularea *Q-value*-ului pentru o stare s se realizează în funcția *computeQValueFromValues(state, action)* prin formulă existentă în cerință. Aceasta depinde de valoarea funcțiilor de tranziție (arereprezintă probabilitatea cadin s să se ajungă în s' prin a) posibile din starea respectivă, de recompense și de valoarea state-ului înmulțită cu factorul de discount. Factorul de discount este acela care determină prioritizarea recompenselor mai apropiate în profida celor îndepărtate. Este firesc ca o recompoensa îndepărtată să nu fie de o mare importanță în starea curentă, fapt pentru care factorul de discount scade exponențial odată cu creșterea mărimii arborelui de stări.

În urmă rulării comenzii de verificare a algoritmului, am putut observa că valorile aferente fiecărei stări nu se schimbă semnificativ de la rularea a 5 iteratii la rularea a 6000.

2 Question 2 - Bridge Crossing Analysis

Cerința acestui punct constă în schimbarea a doar unuia dintre parametrii discount și noise (inițial 0.9 și 0.2) astfel încât agentul, aflat pe un "pod" la capătul căruia se află o stare terminală cu recompensă mică și una cu recompensă mare, să aleagă calea care duce spre starea cu recompensă mare, chiar dacă această se află mai departe.

Deși soluția cea mai la îndemână a fost mărirea discount-ului, întrucât un discount mare ar determina luarea în calcul și a recompenselor aflate la o distanță mai mare decât starea curentă, căpăul podului cu recompensă mare este mult prea îndepărtat, iar capătul cu recompensă mică, mult prea apropiat ca această abordare să funcționeze.

Astfel, am trecut la modificarea parametrului noise, care se referă la cât de des un agent dorește să realizeze o acțiune și să ajungă într-o anumită stare, însă ajunge în alta. Deoarece se dorește că agentul să ajungă în starea specifică cu recompensă mare, am imparitit inițial noise-ul

la 10, acesta ajungând la valoarea 0.02. În urmă rulării exemplului, modul în care *BridgeGrid*-ul arată se regăsește în Figura 1. Atunci când se vrea ca agentul să o ia la dreapta, acesta nu o face, însă *GridWord*-ul inițial indică faptul că ne apropiem de rezultat. Dacă valoarea de 0.02 a noise-ului mai este împărțită o dată la 10, ajungându-se la 0.002, se ajunge la soluția optimă, agentul deplasându-se, într-adevăr, spre dreapta.

Figura 1: *BridgeGrid* cu discount=0.9 si noise=0.02

3 Question 3 - Policies

Întrebarea cu numărul 3 se învârte în jurul unui DiscountGrid. Acesta are două stări finale, una cu recompensă mică și apropiată de poziția de start, iar alta cu recompensă mare, însă mai depărtată de poziția de start. Foarte aproape de poziția de start se află o întreagă "prăpastie" de poziții terminale cu recompensă negativă. Scopul acestei întrebări este de a da diferite valori parametrilor discount, noise și living reward pentru a determina diferite scenarii să aibă loc. În aceste scenarii, agentul trebuie să:

- Prefere ieșirea apropiată (+1) și să se riște cu prăpastia (-10)
- Prefere ieșirea apropiată (+1) și să evite prăpastia (-10)
- Prefere ieșirea îndepărtată (+10) și să se riște cu prăpastia (-10)
- Prefere ieșirea îndepărtată (+10) și să evite prăpastia (-10)

Pentru a prefera ieșirea apropiată chiar dacă recompensa acesteia este mică, am ales o valoare mică pentru discount, anume 0.3 (în acest fel, se prioritizează recompensele apropiate), iar pentru ca agentul să riște trecerea pe lângă prăpastie, o valoare foarte mică pentru noise, adică 0.001, fapt ce determina ca probabilitatea ca agentul să ajungă într-o stare nedorită să fie foarte mică.

Evitarea prăpastiei, în timp ce se păstrează preferința pentru recompensele mai apropiate se realizează simplu prin mărirea noise-ului până la valoarea 0.1.

Pentru ca agentul să prefere ieșirea mai depărtată cu recompensă mai mare, dar și să se riște mergând pe lângă prăpastie, e nevoie de un discount mai mare (0.9) pentru a lua în considerare și recompensele mai îndepărtate. Riscarea pe lângă prăpastie se obține, ca în primul caz, printr-o valoare de 0.001 pentru noise.

Păstrarea preferinței pentru recompensa mare și îndepărtată, dar alegera unui drum mai puțin riscant se face prin creșterea noise-ului, acesta având valoarea 0.2. Dacă se doresc valori și mai mari pentru noise, agentul începe să facă mișcări nedorite.

4 Question 4 - Prioritized Sweeping Value Iteration

Am ales să studiez articolul Learning to model other minds de pe site-ul OpenAI. Acesta descrie algoritmul implementat de companie în colaborare cu Univesitatea Oxford, Learning with Opponent-Learning Awareness (LOLA). Algoritmul are ca scop modelarea unor agenți care, folosind reinforcement learning, să învețe de la alți agenți (a căror învățare se bazează pe răspunsurile primite de la mediu) modul în care să se comporte astfel încât să poată colabora cu aceștia ulterior.

Scopul final unui agent LOLA este de a descoperi strategii care, în cadrul unui joc cu oponenți, să se concentreze pe colaborare, întrucât aceasta este în majoritatea cazurilor soluția optimă. Astfel, acesta reușește să găsească strategii foarte bune pentru jocuri care aparțin teoriei jocurilor, cum ar fi dilema prizonierului, un joc infinit cu doi jucători care se repetă fără ca jucătorii să știe când va avea loc ultima rundă.

Dilema prizonierului constă în existența a doi suspecți care se află în camere diferite de interogatoriu. Fiecare prizonier poate face două acțiuni: să mărturisească sau să nu vorbească. Dacă un prizonier mărturisește, iar celălalt, nu, aceasta este o acțiune favorabilă pentru cel dintâi și una deloc favorabilă pentru cel din urmă. Dacă ambii aleg să mărturisească, aceștia vor avea de pierdut la fel de mult, dar, dacă niciunul nu mărturisește, urmările nu sunt la fel de grave ca în cazul în care ambii mărturisesc. În cadrul teoriei jocurilor, se cunoaște faptul că cea mai bună alegere pe care o pot face cei doi prizonieri este să colaboreze, adică să nu se trădeze unul pe celălalt, sau să se trădeze unul pe altul. Această strategie se numește tit-for-tat. par Acesta reprezintă un joc la care agenții de reinforcement learning clasici nu ajung să considere colaborarea ca fiind cea mai bună soluție, recurgând la alegeri individualiste. Factorul care diferențiază agenții LOLA este prezența unei forme de teorii a minții (capacitatea de a deduce modul în care se simt alți oameni sau acțiunile pe care sunt predispuși să le facă), care se aplică mai exact în faptul că agentul diferențiază un alt agent de orice altă parte a arborelui aferent problemei (lucru neîntâlnit la agenții de reinforcement learning clasici).

Esența acestui tip de învățare este observarea celorlalți agenți. Dacă agentul A face o alegere, după care agentul B face altă alegere, LOLA reține relația dintre aceste alegeri. După ce agentul A face din nou o alegere, LOLA reține, din nou, relația dintre alegerea agentului B și noua alegere a lui A. Înmulțind valorile acestor două relații, LOLA reușește să determine modul în care alegerile lui B pot influența alegerile lui A. În acest fel, se poate face o predicție a modului în care agenții clasici se vor comporta în viitor bazat pe acțiunile care au fost observate până la un anumit moment.

O serie de dezavantaje pe care Learning with Opponent-Learning Awareness le prezintă sunt faptul că folosește multă memorie și putere de calcul, respectiv faptul că pot prezenta instabilitate în undele scenarii.

5 Question 5 - Q-Learning

Dat fiind faptul că agentul de tip value iteration nu învață din propria experiență, această cerință are în vedere implementarea agentului de Q-learning, care învață de pe urmă acțiunilor sale și a modului în care mediul îi răspunde.

Implementarea propusă se bazează pe formulele prezente în suportul de curs.

Pentru a implementa această funcționalitate, trebuie completată funcția *update()* din cadrul fișierului *qlearningAgents*. Aceasta are rolul de a actualiza valoarea *Q-value*-ului corespunzător stării și acțiunii procesate. Odată calculat și următorul *Q-value* (maximul din cele existente),

acestuia i se adaugă reward-ul pentru a ajunge din starea curentă în starea următoare, iar valoarea obținută este adunată cu valoarea curentă. Cu toate acestea, aici se introduce noțiunea de *learning rate*. Dacă acesta este mare, se va pune accentul pe valoarea nou calculată a *Q-value*-ului, iar, în caz contrar, sedimentarea informațiilor va dura mai mult și se va face mai anevoios.

Funcția computeValueFromQValues(state) a fost implementată cu scopul de a returna Q-value-ul maxim raportat la toate acțiunile legale care se pot realiza dintr-o stare anume.

În cele din urmă, funcția *computeActionFromQValues(state)* are rolul de a selecta acțiunea optimă dintr-o anumită stare. Procedeul prin care se realizează această alegere este următorul: în cazul în care nu există nicio acțiune legală, se va returna *None*, iar, în caz contrar, se procesează fiecare acțiune legală și se alege acțiunea cu cel mai mare *Q-value*. În cazul în care printre aceste acțiuni există vreo acțiune a cărui *Q-value* asociat este zero, aceasta va fi salvată. La sfârșit, dacă toate acțiunile procesate au un *Q-value* negativ, însă există și acțiuni care să aibă un *Q-value* egal cu 0 (adică nevizitate încă), agentul va alege la întâmplare o acțiune nevizitată. Dacă, totuși, s-a găsit o acțiune maximă care să aibă un *Q-value* pozitiv, aceasta va fi cea aleasă.

6 Question 6 - Epsilon Greedy

Această cerință se focuseaza pe implementarea unui scurt algoritm care să aleagă aleatoriu una din acțiunile legale pentru *epsilon* la sută din timp, iar, în rest, să aleagă cea mai bună acțiune în funcție de *Q-value*-ul implementat la cerință anterioară.

Pentru aceasta, a fost folosită funcția *util.flipCoin(p)*, care returnează True cu o probabilitate p, iar Fals, cu o probabilitate 1-p. Dacă funcția returnează True, se alege o acțiune random din acțiunile legale cu ajutorul funcției *random.choice*, iar, dacă returnează False, se calculează acțiunea cu *Q-value*-ul maxim.

Astfel, în urmă rulării codului cu un epsilon mic (0.1), se poate observa că agentul nu explorează prea mult acțiunile și stările existente, anume, odată ce a găsit un drum bun, rămâne pe acesta. Figura 2 ilustrează rezultatul acestui comportament după 100 de iteratii.

Dacă, totuși, se alege un epsilon mare (0.9), se poate observa cum agentul ajunge să exploreze întreg *GridWord*-ul, din cauza faptului că de foarte multe ori alege să facă acțiuni aleatorii. Totodată, timpul de efectuarea a celor 100 de iteratii durează mult mai mult decât în cazul unui epsilon mic, însă harta ajunge să fie mult mai bine cunoscută decât în cazul anterior. Rezultatul acestui comportament după 100 de iteratii se regăsește în Figura 3.

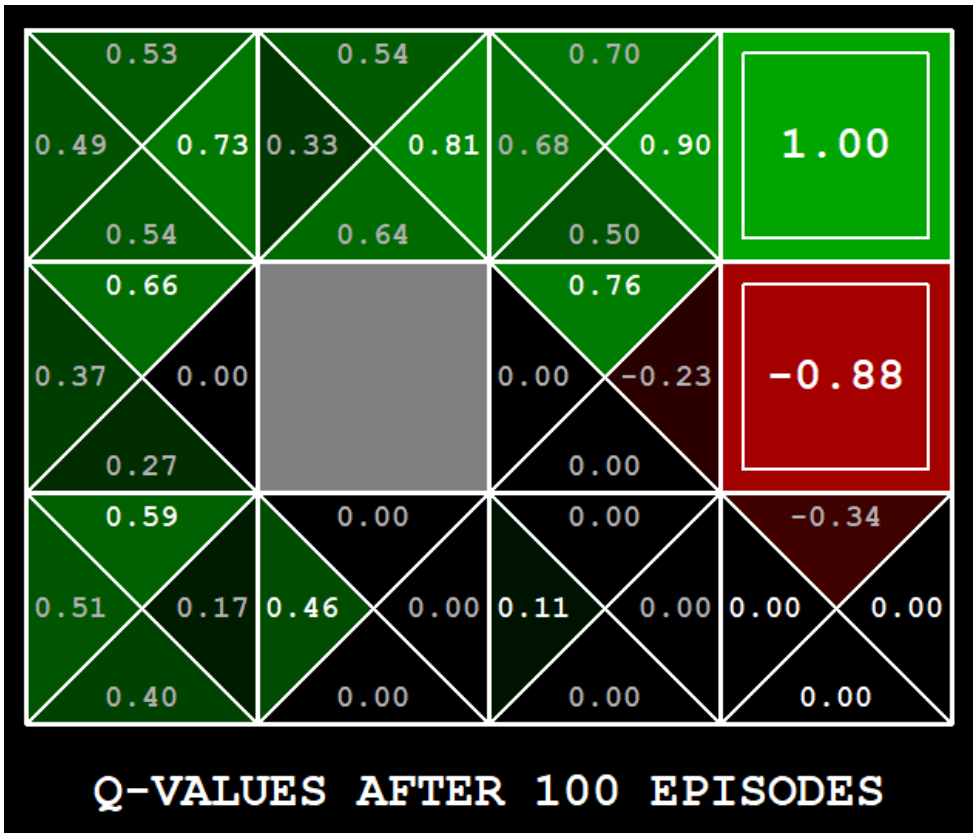

Figura 2: Epsilon Greedy, epsilon=0.1

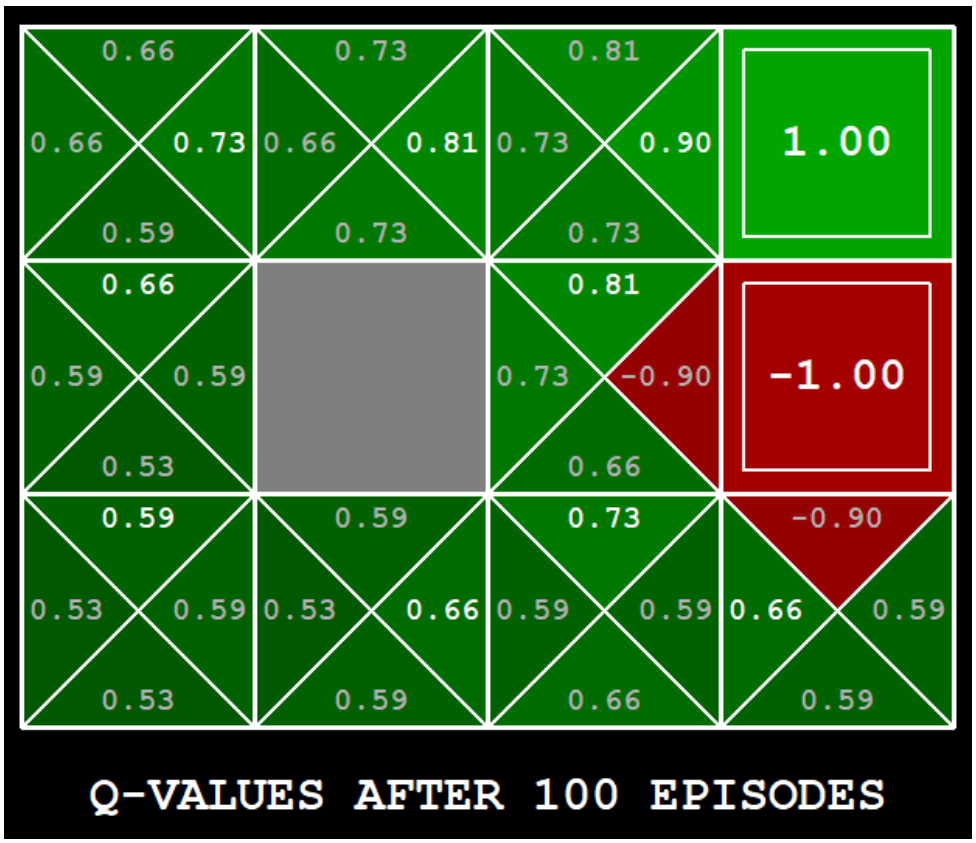

Figura 3: Epsilon Greedy, epsilon=0.9

7 Question 7 - Bridge Crossing Revisited

Cerința actuală începe cu nevoia de a antrena un Q-learner într-un *BridgeGrid* cu un epsilon de valoare 1 și fără că noise-ul să fie prezent pentru 50 de episoade. Acest pas are rolul de a permite observarea agentului care nu ajunge să găsească politica optimă în acest fel (adică partea îndepărtată, dar cu recompensă mare). Q-value-urile calculate se pot observa în Figura 4.

Figura 4: BridgeWorld, epsilon=1

În schimb, dacă se rulează același exemplu cu un epsilon de valoare 0, odată găsită calea spre recompensa dorită, aceasta se menține, iar alegerea mai multor acțiuni random nu are loc. Acest fapt poate fi observat în Figura 5.

Figura 5: BridgeWorld, epsilon=0

Astfel, cerința constă în găsirea unei valori pentru epsilon și pentru learning rate astfel încât să se găsească politica optimă după cele 50 de iterații.

Intuiția spune că, pentru a ajunge la cealaltă parte a *BridgeGrid*-ului, epsilon-ul ar trebui să fie maxim, iar learning rate-ul, minim. Aceste valori ar determina agentul să realizeze și acțiuni aleatorii, ceea ce i-ar permite să exploreze harta mai bine, iar learning rate-ul mic ar consolida deciziile de a alege căile "nebătătorite" pe parcursul explorării. Cu toate acestea, politica optimă nu este găsită în 50 de iterații. Pentru a mă asigura că nu am mers în direcția

greșită cu presupunerea făcută despre valori, am încercat și creșterea learning-rate-ului, pentru ca agentul să învețe mai repede, spre exemplu, că stările terminale negative nu sunt o alegere bună și să se focuseze pe restul, continuând explorarea, însă soluția optimă tot nu a fost găsită. Aceste observații au fost făcute atât în urmă a 50 de iterații, cât și în urmă a 500 de iterații.

Așadar, răspunsul pentru această întrebare este că nu e posibil că agentul să găsească politica optimă în cadrul *BridgeWorld*-ului în doar 50 de iterații și cu modificarea doar a epsilon-ului și a learning rate-ului.

8 Question 8 - Q-Learning and Pacman

Acest punct cere doar antrenarea lui Pacman, urmată de testarea acetuia. Fiindcă *Q-learning* a fost deja implementat, această cerință nu are ca scop implementarea de cod, ci doar observarea modului în care Pacman se comportă cu ajutorul codului implementat la cerința anterioară care vizează implementarea *Q-learning*-ului într-un *GridWorld*.

Astfel, în urma antrenării (în cadrul căreia următorii parametrii sunt definiți astfel: epsilon=0.05, alfa=0.2, gama=0.8), urmează cele 10 runde de joc, în care parametrii epsilon și alfa sunt setați la 0 pentru a-i permite agentului să se bazeze doar pe informațiile acumulate în cadrul training-ului. Se poate observa cum fiecare rundă se termină cu Pacman câștigând.

9 Question 9 - Approximate Q-Learning

Ultima cerință are ca scop implementarea funcționalității de *aproximate Q-learning*.

Dat fiind faptul că, într-un grid complex, nu este rentabil să se rețină toate *Q-value*-urile, deoarece acestea sunt în număr foarte mare, iar exlporarea tuturor stărilor în faza de antrenare ar dura prea mult, este nevoie să generalizăm *Q-learning*-ul. Această generalizare constă în explorarea unui număr mai mic de stări și generalizarea anumior experiențe care se pot întâlni și în alte părți. Astfel, o pereche (stare, acțiune) poate fi generalizată prin atașarea unor proprietăți specifice acelei stări (cum ar fi distanțăa până la cea mai apropiată fantomă, distanță până la cea mai apropiată bucată de mâncare etc.), dar și a unor ponderi atașate fiecărei proprietăți, cele din urmă determaind importantă proprietăților.

Fiindcă funcția care returnează lista de proprietăți aferente unei perechi (stare, acțiune) este deja funcțională, mai trebuie implementată funcția *update(state, action, nextState, reward)*. Conform formulelor din materialul de curs, implementarea acesteia constă în calcularea *Q-value*-ului actual și a *Q-value*-ului stării următoare, urmată de calcularea diferenței dintre vloarea următorului *Q-value* (înmulțită cu discount-ul și adunată cu recompensă) și a actualului *Q-value*. Se poate observă cum, și în acest caz, se păstrează aplicarea proprietății discount-ului fiecărui *Q-value* al unei stări următoare. La final, fiecărei ponderi aferente unei proprietăți i se adaugă valoarea proprietății înmulțită cu *learning rate*-ul și cu diferența calculată anteorior.

Totodată, pentru calcularea *Q-value*-ului curent, e nevoie de funcția *getQValue(state, action)*. Spre deosebire de *Q-learning*-ul simplu, în acest caz, *Q-value*-ul este reprezentat de suma tuturor proprietăților înmulțite cu ponderile aferente.

Q-value-ul următoarei stări se calculează folosind funcția *computeValueFromQValues(state)*, implementată în cadrul *Q-learning*-ului simplu.

Rularea exemplului atât pe labirintul de mărime mică, cât și pe cel de mărime medie are cs rezultat câștigarea aproape a fiecărei runde de către Pacman(9/10).

10 Bibliografie

Learning to model other minds,
https://openai.com/research/learning-to-model-other-minds
Tit for tat,
https://en.wikipedia.org/wiki/Tit_for_tat
Prisoners' Dilemma,
https://www.econlib.org/library/Enc/PrisonersDilemma.html