Reinforcement Learning

Documentatie proiect

Facultatea de Automatica si Calculatoare

Coltea Alexandra Daria

FACULTATEA DE AUTOMATICA SI CALCULATOARE

UNIVERSITATEA TEHNICA DIN CLUJ-NAPOCA

Ianuarie 2023

Chapter 1

Introducere

Proiectul implementeaza reinforcemend learning, Qlearning si Aproximative Reinforcement learning. In prima faza proiectul dispune de un MDP pe care implementam value iteration, iar apoi implementam Qlearning.Se testeaza mai întâi agenții implementati pe Gridworld, apoi se aplica la Crawler și Pacman.

Chapter 2

Implementare

2.1 Q1-Value Iteration

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma\, V^*(s')\right]$$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma\, V_k(s')\right]$$

La Q1 am implementat un agent value iteration. Acest agent nu este un agent de reinforcement learning, astfel trebuie sa ii spunem cate iteratii sa faca initial in faza de planificare. Constructorul clasei apeleaza functia runValueIteration, care ruleaza pe MDP-ul primit un anumit numar de iteratii.Astfel, functia implementata ruleaza de i ori o actualizare a valorilor. Functia ia pe rand toate starile din MDP, verifica sa nu fie stare terminala(daca e stare terminala nu mai este nevoie de recalculare) si apoi ia pe rand toate actiunile care se pot face din respectiva stare. Pentru fiecare actiune posibila , se recalculeaza Q* si apoi se compara cu valoarea de la iteratiile anterioare. Daca noua valoare este mai buna (mai mare) , se actualizaeaza valoarea din dictionarul values.

De asemenea, la acest pas implementam si functiile computeQValueFromValues si computeActionFromValues. Functia computeQValueFromValues calculeaza Q*

pentru o stare si o actiune data, conform formulei(ia starile in care se poate ajunge din starea actuala si probabilitatile cu care se poate ajunge in fiecare stare si realizeaza suma depinzand probabilitate,reward,valoarea tranzitiei si discount). Functia compute-ActionFromValues returneaza cea mai avantajoasa actiune care se poate realiza dintr-o anumita stare. Cea mai avanatjoasa actiune este cea care are Qvalue cel mai mare.

2.2 Q2-Bridge Crossing Analysis

$$V^*(s) = \max_a Q^*(s,a)$$
$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma\, V^*(s')\right]$$

La Q2 avem doi factori de care depindem, discount si noise (probabilitatea ca sa ajungem intr-o stare nedorita). Astfel, daca noi dorim sa trecem podul si sa ajungem la rewardul mai mare, avem nevoie de un dicount cat mai apropiat de 1 (cat mai mare) pentru a calcula Q(cu rewardul cel mai mare indepartat) si de un noise cat mai mic(preferabil 0) care sa ne garanteze ca de fiecare data cand actiunea cea mai buna este intr-o anume directie, agentul nostru va merge in acea directie. De pe schema de la Q2 se poate observa ca, cu un noise de 0.2, dupa 100 de iteratii agentul nostru a ajuns de atatea ori in starile -100 incat este nu a invatat calea buna(are doar valori negative si va alege reward +1 pentru ca este cel mai mare V). Daca noi garantam ca o sa ajunga mereu in starea aleasa, cu un noise=0.0, el va putea calcula Q si V optime.

2.3 Q3-Policies

$$V^*(s) = \max_a Q^*(s,a)$$
$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma\, V^*(s')\right]$$

1.Prefer the close exit (+1), risking the cliff (-10). In acest caz, daca dorim

sa riscam sa mergem pe langa ”cliff”, trebuie sa ii spunem agentului sa ia in considerare drumul cel mai scurt. Pentru a asigura ca agentul alege calea cea mai buna calculata(cea mai scurta), vom seta noise la 0(probabilitatea ca sa ajungem intr-o stare nedorita este nula),astfel nu vom ajunge in -10 si nu vom urca in sus. Pentru a alege iesirea cu reward mai mic, avem nevoie de un discount cat mai apropiat de 0(care sa faca rewardul 10 sa para prea ”indepartat” pentru a fi tenatant), astfel agentul prefera un reward mai mic dar mai apropiat. LivingReward poate fi setat la 0.5 pentru a indruma agentul sa se miste pe drumul catre iesire(rewardul pe care il are daca doar exista nu e destul de tentant ca sa nu mearga la iesire unde va primi un reward mai mare). Cu cat este mai mic living reward cu atat va incerca sa faca mai multe actiuni ca sa ajunga la o iesire pt a avea un reward mai mare.

2.Prefer the close exit (+1), but avoiding the cliff (-10). In acest caz, pentru a indruma agentul spre drumul mai lung, avem nevoie de noise(o mica probabilitate ca agentul sa ajunga intr-o stare diferita). Astfel, putem seta noise la 0.1, in timp ce livingReward poate ramane tot la 0.5. Pentru a ne asigura ca agentul prefera cea mia apropiata iesire, trebuie sa setam un discount apropiat de 0, care scade importanta rewardului de la iesirea indepartata, facand astfel rewardul mai mic dar mai apropiat sa para mai bun. 3.Prefer the distant exit (+10), risking the cliff (-10). In acest caz avem nevoie de noise cat mai mic(in cazul nostru aleg 0) pentru a ne asigura ca o va lua in directia aleasa. Living rewardul poate fi crescut la 0.7(orice valoare atata vreme cat e mai mic decat 1[pt discount 0.9]-daca ar fi 1 nu ar mai avea motiv sa mearga pana la iesirea 10 deoarece cu discount 0.9 si living reward 1 am avea mai buna starea de a sta in labirint). Discountul trebuie sa fie cat mai apropiat de 1 pentru ca agentul sa aleaga rewardul mai mare chiar daca este mai indepartat(daca avem discount mai mare, iesirea indepartata este optima deoarece nu conteaza atata de mult un reward imediat, ci un reward mai mare). 4.Prefer the distant exit (+10), avoiding the cliff (-10). Cazul acesta se aseamana mult cu cazul 3, diferenta fiind ca pentru a alege drumul mai lung avem nevoie de noise(probabilitate sa o luam in alta directie decat cea oprima). Astfel, setam noise la 0.2. 5.Avoid both exits and the cliff (so an episode should never terminate). Daca setam parametrul de living reward la valoarea 10(valoarea maxima a rewardului pe care il putem primi) si setam discount-ul diferit de 0(ex 0.1) agentul nu va ajunge niciodata la starea de iesire deoarece va calcula ca este mai rentabil sa nu termine

episodul(rewardul de iesire nu va fi suficient de insemnat pentru ca sa conteze). Ca sa evitam ”the cliff” setam noise la 0.

2.4 Q4-Prioritized Sweeping Value Iteration

Prima data realizam un set cu toti predecesorii fiecarei stari in care exista probabilitatea sa ajungem. Un predecesor al unei stari b este o stare a , daca din starea a in starea b pot ajunge realizand o actiune. Probabilitatea de a ajunge din a in b trebuie sa fie nenula pozitiva pentru a fi un predecesor valid. Urmatorul pas este de a realiza o priority queue, care are starile ca si chei si -abs(diff) ca si valoare. Diff reprezinta diferenta intre valoarea curenta a starii si Q-ul maxim calculat din starea curenta Ultimul pas in calcularea valorii este rularea a i iteratii pentru faza de planificare(pentru a ajuta agentul sa invete mediul). La fiecare iteratie scoatem elemente din coada(cata vreme exista) si actualizam valoarea in self.values. Pentru fiecare predecesor al starii(pe care l-am calculat anterior) calculam din nou diferenta diff in acelasi mod si o adaugam in coada.

2.5 Q5- Q-learning

computeValueFromQValues: Aceasta metoda calculeaza cea mai buna valoare(cea mai mare) la care agentul poate ajunge dintr-o stare data in urma unei actiuni. computeActionFromQValues: Metoda returneaza actiunea optima care se poate realiza dintr-o stare data. Optimalitatea se considera in functie de valoarea maxima a Qvalue(cea mai buna actiune este cea care are Qvalue maxim). Daca exista mai multe actiuni cu valori egale alegem una la intamplare.

- Consider your new sample estimate:

$$sample = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

no longer policy evaluation!

- Incorporate the new estimate into a running average:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha)\,[sample]$$

update:Metoda update se ocupa de invatarea treptata propriu-zisa. In cadrul acestei

metode implementam ecuatiile de mai sus. Intai calculam noua valoare estimata a starii, si apoi pentru a avea valoarea propriu-zisa a acestui pas, calculam o medie intre Qvalue avut si noul Q calculat. Astfel, avem o medie intre ce stiam deja si ce am aflat la acest pas(agentul devine mai informat).

2.6 Q6-Epsilon Greedy

getAction: Aceasta metoda alege o actiune posibila pe care agentul o poate face dintr-o anumita stare. In functie de valoarea epsilon, actiunea returnata va fi ori una random, ori una (cu siguranta) optima calculata cu ajutorul metodei implementate computeActionFromQValues. Avand aceasta probabilitate de a alege o actiune random, agentul nu face mereu ce ne-am astepta sa faca(nu ia mereu drumul care a invatat deja ca e cel mai bun), ci uneori exploreaza o alta cale despre care inca nu cunoastem informatii.

2.7 Q7-Bridge Crossing Revisited

Indiferent de valorile la care le setam pe epsilon si learning rate, agentul nostru nu va gasi varianta optimala dupa doar 50 de iteratii. Are nevoie de un numar mai mare de iteratii, pentru a acumula un numar mai mare de cunostinte pe baza carora sa ia decizia optimala. Cand epsilon este pozitiv, agentul este imprevizibil alege stari random pt a le explora,lucru care mareste bagajul de cunsotinte, dar nu suficient in 50 de iteratii.

2.8 Q8- Q-Learning and Pacman

Ceea ce am implementat pana acum se aplica pt pacman. Pacman va avea 2 faze.In faza de planificare si invatare,pacman va rula de cateva ori pentru a invata informatiile de baza despre lumea in care se afla(va calcula values si Q). In faza de testare, pacman va utiliza informatiile invatate pentru a vedea daca sunt optime(aduc rezultate). Pe labirintul pe care este testat, pacman se va descurac f bine cu informatiile invatate in starea de invatare.

2.9 Q9-Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

getQValue: calculeaza si returneaza Qvalue, dar de aceasta data Q value este dependenta de weights si features. se aplica formula de mai sus update:actualizeaza

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$$
$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$$
$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$

valorile din weights pe baza feature-rurilor active in acel moment. Se da starea actuala , starea urmatoare si o actiune si se calculeaza cea mai buna actiune pentru care se poate opta din starea urmatoare.Se calculeaza apoi diferenta dintre valoarea actiunii noi calculate si valoarea actiunii din starea curenta. Noua valoare a weights este o aproximare Q, calculata din vechea valoare (informatie deja cunoscuta) la care se adauga produsul intre diferernta calculata, factorul alpha si feature-ul corespunzator.