Proiect Reinforcement Learning

Poponet Tiberiu Sergiu

Grupa 30231

ValueIteration: implementează algoritmul de iterație a valorilor (Value Iteration) în contextul învățării prin recompensă pentru un proces de decizie Markov (MDP). Într-un ciclu de lungime specificată de self.iterations, se actualizează iterativ valorile stărilor în funcție de acțiunile posibile și utilitățile asociate acestora. Pentru fiecare stare intermediara se calculează estimări ale utilităților pentru acțiunile posibile și se calculează valorile Q asociate, care reprezintă estimările de câștig pe termen lung pentru acele acțiuni în acea stare.  
 Funcția computeQValueFromValues calculează valoarea Q pentru o anumită acțiune într-o stare dată. Se inițializează valoarea Q la 0 și se iterează prin stările următoare și probabilitățile asociate acestora pentru acțiunea specificată. Se adaugă la valoarea Q produsul dintre probabilitate, recompensă și valoarea stării următoare ponderată de factorul de reducere (discount). Rezultatul final reprezintă valoarea Q pentru acțiunea în acea stare.

Funcția computeActionFromValues determină acțiunea optimă într-o stare dată pe baza valorilor stocate în self.values. Se obțin acțiunile legale pentru starea respectivă, iar dacă nu există acțiuni legale (stare terminală), se returnează None. În caz contrar, se iterează prin acțiunile posibile și se calculează valoarea Q pentru fiecare acțiune utilizând funcția computeQValueFromValues. Se alege acțiunea cu cea mai mare valoare Q și se returnează această acțiune ca fiind cea mai bună conform politicii.

Prin ajustarea parametrilor de discount și noise, am explorat diferite scenarii pentru a determina o politică optimală care să încurajeze agentul să încerce să traverseze puntea în mediu. Pentru cerința "BridgeGrid", scopul este să determinăm o configurație a parametrilor astfel încât politica optimă să determine agentul să încerce să traverseze puntea.

În question2(), am ajustat parametrul de discount, reducându-l la 0.3, ceea ce poate face ca agenții să acorde o importanță mai mare recompenselor pe termen scurt. answerNoise este setat la 0 pentru a indica că nu există zgomot sau incertitudine asociată cu acțiunile agentului.

În question3a(), am introdus un nivel mai mic de zgomot (answerNoise = 0.1) împreună cu un discount de 0.9 și o recompensă de trai (answerLivingReward = -3). Aceasta poate face ca agentul să aibă o probabilitate mai mică de a ajunge într-o stare nedorită și, în consecință, să încerce să traverseze puntea pentru a evita recompensele rele.

În question3b(), am ajustat discountul la 0.3 și noise-ul la 0.2, pentru a observa dacă o combinație a acestora ar putea influența agentul să exploreze mai mult zona punții.

Pentru question3c(), discount-ul este setat la 1, indicând o atenție egală acordată tuturor etapelor viitoare, și noise-ul este 0 ,eliminând probabilitatea de a ajunge în stări nedorite.

În question3d(), am crescut nivelul de noise la 0.5 și discountul la 1.0, experimentând cu o probabilitate mai mare ca agentul să ajungă în stări nedorite, astfel încât să exploreze și să considere posibilitatea de a traversa puntea.

În question3e(), am setat discountul la 0 (zero) și noise-ul la 0, indicând că agentul ar acorda o importanță deosebită recompenselor pe termen scurt și nu ar exista nicio probabilitate de a ajunge în stări nedorite.

Aceste ajustări au fost făcute în încercarea de a obține o politică optimă care să determine agentul să încerce să traverseze puntea în mediu, luând în considerare influența parametrilor de discount și noise asupra comportamentului agentului.

getQValue returnează valoarea Q asociată unei perechi (stare, acțiune). Valorile Q sunt stocate într-un dicționar self.q\_values și sunt accesate prin cheia formată din perechea (stare, acțiune). Dacă perechea (stare, acțiune) nu a fost întâlnită anterior, funcția returnează 0.0.

computeValueFromQValues(self, state): Funcția calculează valoarea maximă a funcției Q pentru starea dată. Se obțin acțiunile legale pentru starea curentă, iar dacă nu există acțiuni legale (cazul stării terminale), se returnează 0.0. Altfel, se iterează prin toate acțiunile legale, se obțin valorile Q corespunzătoare și se returnează valoarea maximă.

computeActionFromQValues(self, state): Această funcție determină acțiunea optimă pentru o anumită stare. Se obțin acțiunile, iar dacă nu există nicio acțiune (cazul stării terminale), se returnează None. În caz contrar, se creează un obiect Counter pentru a stoca valorile Q asociate fiecărei acțiuni. Se determină acțiunea cu cea mai mare valoare Q folosind metoda argMax() a obiectului Counter și se returnează această acțiune.

In concluzie, proiectul este o implementare detaliată a algoritmului de învățare automată prin recompensă pentru un MDP în contextul unui joc Pacman. S-au explorat concepte precum iterația valorilor și învățarea optimă a politicii într-un mediu complex. De asemenea, am analizat impactul parametrilor precum discount și noise asupra comportamentului agentului.