UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Marian Iosif Andrei
Grupa: 30232

Proiect reinforcement learning

Q1: Value Iteration:

Implementarea acestor funcții este bazată pe algoritmul de iterare a valorilor (Value Iteration) în cadrul unui Markov Decision Process (MDP). Algoritmul este utilizat pentru a găsi valorile optime ale stărilor și politicile asociate acestora într-un MDP.

1. **Funcția runValueIteration:**
    - Acesta este algoritmul principal care iterează de mai multe ori pentru a estima valorile optime ale stărilor (V_k).
    - Pentru fiecare iterație, se parcurg toate stările posibile din MDP.
    - Pentru fiecare stare, se calculează Q-value maximă pentru toate acțiunile posibile.
    - Q-values calculate sunt utilizate pentru a actualiza valorile stărilor următoare (V_k+1).
    - Aceasta se repetă pentru un număr specific de iterații (self.iterations = k).
    - Este important să se utilizeze varianta "batch" a algoritmului, unde valorile succesive V_k sunt calculate pe baza valorilor anterioare V_k-1.
2. **Funcția computeQValueFromValues:**
    - Calculează Q-value pentru o pereche (stare, acțiune) folosind valorile stărilor și funcția de tranziție a MDP-ului.
    - Se obțin stările succesoare și probabilitățile asociate tranzițiilor folosind “getTransitionStatesAndProbs”.
    - Se calculează suma ponderată a recompensei immediate și a valorii stării succesoare, luate în considerare și factorul de discount.
    - Aceasta reflectă valoarea așteptată a următoarei stări date starea curentă și acțiunea.
3. **computeActionFromValues Function:**
    - Calculează acțiunea optimă pentru o stare dată, folosind valorile stărilor și Q-valori calculate.
    - Dacă starea este o stare terminală, se setează valoarea stării la “0” și se returnează “None”.
    - Altfel, se determină acțiunea care maximizează Q-valoarea pentru starea respectivă.

Marian Iosif Andrei
Grupa: 30232

Q2: Bridge Crossing Analysis:

- Pentru a face ca politica optimă să determine agentul să încerce să traverseze podul în harta BridgeGrid, avem două opțiuni: putem ajusta factorul de discount sau putem ajusta nivelul de zgomot (noise). Eu am ales să se ajustez nivelul de zgomot (answerNoise).
- Valoarea answerNoise este setată la un număr foarte mic, apropiat de zero, în acest caz, 0.00000001. Aceasta sugerează că agentul are o probabilitate foarte mică de a ajunge într-o stare succesoare nedorită atunci când execută o acțiune. Cu alte cuvinte, agentul are o probabilitate foarte ridicată de a ajunge în starea dorită atunci când efectuează o acțiune, ceea ce îl face să urmeze politica optimă care îl conduce să traverseze podul.
- Practic, ajustarea nivelului de zgomot reduce incertitudinea asociată cu acțiunile agentului, făcând mai probabil ca acesta să urmeze traseul optim care trece peste pod. Acest lucru face ca traversarea podului să devină o opțiune mai atrăgătoare, chiar dacă recompensele pe pod pot fi mai mici decât cele din alte părți ale hărții.

Q3: Policies:

1. question3a - Prefer the close exit (+1), risking the cliff (-10)
   - **answerDiscount = 0.9**: Valoarea relativ mare a discount-ului indică că agentul acordă o atenție semnificativă recompensei viitoare, ceea ce îl motivează să optimizeze pentru câștigul imediat.
   - **answerNoise = 0.2**: Un nivel moderat de zgomot indică că agentul poate să se abată cu o probabilitate de 20%, dar preferă să se apropie de ieșirea apropiată.
   - **answerLivingReward = -4**: O recompensă pentru viață negativă motivează agentul să ajungă cât mai repede posibil la o ieșire, chiar dacă aceasta implică un risc de a cădea în prăpastie.
2. question3b - Prefer the close exit (+1), but avoiding the cliff (-10)
   - **answerDiscount = 0.5**: Discount-ul mai mic indică că agentul este mai orientat către recompensele imediate.
   - **answerNoise = 0.2**: Se menține un nivel moderat de zgomot, însă discount-ul mai mic îl face pe agent să fie mai precaut și să evite prăpastia.
   - **answerLivingReward = -1**: O recompensă pentru viață mai puțin negativă îl face pe agent să fie mai dispus să își prelungească traseul pentru a evita prăpastia.
3. question3c - Prefer the distant exit (+10), risking the cliff (-10)
   - **answerDiscount = 0.95**: Un discount mai mare indică că agentul este mai orientat către obținerea recompensei mari din ieșirea îndepărtată.
   - **answerNoise = 0.05**: Un nivel scăzut de zgomot indică că agentul este dispus să riste apropiindu-se de prăpastie pentru a ajunge mai rapid la ieșirea de departe.

- **answerLivingReward = -0.1**: O recompensă pentru viață puțin negativă îl motivează pe agent să aleagă traseul cel mai scurt, chiar dacă implică un risc de a cădea în prăpastie.

4. question3d - Prefer the distant exit (+10), avoiding the cliff (-10)

   - **answerDiscount = 0.8**: Un discount mai mic indică că agentul acordă o atenție mai mare recompenselor imediate și nu se concentrează exclusiv pe recompensa mai mare a ieșirii îndepărtate.
   - **answerNoise = 0.4**: Un nivel mai ridicat de zgomot îl face pe agent să fie mai precaut și să evite prăpastia pentru a ajunge la ieșirea de departe.
   - **answerLivingReward = -1**: O recompensă pentru viață mai puțin negativă îl face pe agent să fie dispus să își prelungească traseul pentru a evita prăpastia.

5. question3e - Avoid both exits and the cliff

   - **answerDiscount = 0.8**: Un discount mai mic indică că agentul acordă atenție mai mare recompenselor imediate și nu se orientează spre obținerea recompenselor mari din ieșirile distante.
   - **answerNoise = 0.4**: Un nivel mai ridicat de zgomot îl face pe agent să evite atât ieșirile, cât și prăpastia.
   - **answerLivingReward = 2**: O recompensă pentru viață pozitivă îl face pe agent să evite complet situațiile negative, inclusiv ieșirile cu recompense pozitive.

Q5: Q-Learning:

1. getQValue(self, state, action)

   - Această funcție returnează valoarea Q asociată perechii (state, action) din dicționarul "q_vals". Dacă această pereche nu există în dicționar, funcția returnează 0.0, indicând că nu s-a văzut încă această stare-acțiune.

2. computeValueFromQValues(self, state)

   - Calculează valoarea maximă a Q pentru starea dată, luând în considerare toate acțiunile legale disponibile. Dacă nu există acțiuni legale (cazul stării terminale), funcția returnează 0.0.

3. computeActionFromQValues(self, state)

   - Calculează acțiunea cu cea mai mare valoare Q pentru starea dată. Dacă nu există acțiuni legale (cazul stării terminale), funcția returnează "None". Pentru a gestiona legarea în cazul de egalitate, se utilizează "max" cu o funcție lambda și "default=0" pentru a evita un comportament neașteptat.

4. update(self, state, action, nextState, reward)
    - Această funcție este apelată pentru a observa o tranziție de la starea “state” cu acțiunea “action” la starea “nextState” cu recompensa “reward”.
    - Obține valoarea Q actuală pentru perechea (state, action).
    - Calculează noua valoare Q folosind formula de actualizare Q-learning.
    - Actualizează dicționarul “q_vals” cu noua valoare Q pentru perechea (state, action).
    - Se utilizează parametrii “alpha” și “discount” pentru a controla rata de învățare și factorul de discount.

Q6: Epsilon Greedy:

- “legal_actions” stochează acțiunile legale disponibile în starea curentă.
- Dacă nu există acțiuni legale sau cu o probabilitate epsilon, se alege o acțiune aleatoare, altfel se alege acțiunea cu cea mai mare Q-value conform politicii actuale.
- Agentul alege aleatoriu cu probabilitate epsilon pentru a descoperi noi acțiuni.
- În rest, alege acțiunea cu cea mai mare Q-value cunoscută pentru a maximiza recompensele. Această abordare promovează balanța între explorarea mediului și alegerea acțiunilor cunoscute cu potențial mare de recompense.
- Strategia ajută la învățarea eficientă în medii complexe și nesigure. Implementarea face parte din agentul Q-learning, care învață prin încercare și eroare din interacțiunea cu mediul.

Q7: Bridge Crossing Revisited:

- Alegerea pentru “question8” de a returna “NOT POSSIBLE” este bazată pe faptul că un epsilon de 0 și o rată de învățare (learning rate) nu permit explorarea mediului. Într-un scenariu fără explorare, un agent Q-learning nu are șansa de a descoperi acțiuni alternative sau de a ajusta valorile Q în funcție de experiențele anterioare.
    - Când epsilon este 0, agentul alege întotdeauna acțiunea cu cea mai mare valoare Q cunoscută (exploatare), nefiind dispus să încerce acțiuni noi.
    - Dacă rata de învățare este și ea 0, atunci valorile Q rămân neschimbate, iar agentul nu învață din recompensele primite.
    - Fără explorare și fără ajustarea valorilor Q, agentul nu are modalitatea de a descoperi politica optimă sau de a îmbunătăți învățarea sa.
    - Prin urmare, nu există o combinație de epsilon și rată de învățare care să permită agentului să învețe politica optimă într-un număr redus de episoade, precum cele 50 menționate în cerință.

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Marian Iosif Andrei
Grupa: 30232

Q8: Q-Learning and Pacman:

1. getAction(self, state):
    - Această funcție decide acțiunea pe care Pacman ar trebui să o întreprindă în starea curentă.
    - Cu o probabilitate “epsilon”, se alege o acțiune aleatoare (explorare).
    - În caz contrar, se alege cea mai bună acțiune conform politicii curente (exploatare).
    - Dacă nu există acțiuni legale disponibile (stare terminală), se returnează “None”.
2. computeActionFromQValues(self, state):
    - Această funcție calculează cea mai bună acțiune de luat într-o stare dată.
    - Dacă nu există acțiuni legale disponibile (stare terminală), se returnează “None”.
    - Se utilizează funcția “max” pentru a găsi acțiunea cu cea mai mare valoare Q pentru starea dată.
    - Funcția “key=lambda action: self.getQValue(state, action)” specifică criteriul de ordonare bazat pe valorile Q.

Q9: Approximate Q-Learning:

1. __init__(self, extractor='IdentityExtractor', **args):
    - Constructorul inițializează un obiect ApproximateQAgent.
    - “extractor” este specificatorul de extragere a caracteristicilor (feature extractor) și este implicit setat la 'IdentityExtractor'.
    - “**args” permite orice alt argument să fie transmis și este folosit pentru inițializarea obiectului de bază PacmanQAgent.
    - “featExtractor” este un obiect al extractorului specificat de la care se vor obține caracteristicile.
    - “weights” este un obiect Counter care va stoca valorile ponderilor pentru caracteristici.
2. getQValue(self, state, action):
    - Calculează valoarea Q pentru o pereche (stare, acțiune) folosind produsul scalar între vectorul de caracteristici și ponderile corespunzătoare.
    - Se obțin caracteristicile pentru perechea (stare, acțiune) cu ajutorul extractorului de caracteristici.
    - Se înmulțește vectorul de caracteristici cu ponderile și se returnează rezultatul.

3. update(self, state, action, nextState, reward):
    - Actualizează ponderile în funcție de tranziția de la “state” la “nextState” cu recompensa “reward” și acțiunea “action”.
    - Se calculează diferența dintre recompensa anticipată și Q-value curent.
    - Pentru fiecare caracteristică asociată cu (stare, acțiune), se actualizează ponderile în funcție de diferență și valoarea caracteristicii.
    - Ponderile actualizate sunt stocate înapoi în obiectul “weights”.