UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Reinforcement learning

Inteligenta Artificiala

Autori: Adriana Mihnea
Grupa: 30236

Facultatea de Automatica
si Calculatoare

3 ianuarie 2024

Cuprins

1 Question 1: Value Iteration

```
def runValueIteration(self):
    for _ in range(self.iterations):
        newValues = util.Counter()
        for state in self.mdp.getStates():
            if not self.mdp.isTerminal(state):
                # Compute the Q-value for each action
                Q_values = [self.computeQValueFromValues(state, action) for
                ↪  action in self.mdp.getPossibleActions(state)]
                newValues[state] = max(Q_values)
        self.values = newValues
```

RunValueIteration este metoda principala ce contine algoritmul pentru un anumit numar de iteratii (self.iterations). Initializeaza un nou set de valori (newValues), dictionar care va pastra valorile actualizate pentru fiecare stare. Pentru fiecare stare in MDP:

1. Daca starea nu este terminala:
   - calculeaza Q-value pentru fiecare actine posibila din starea curenta folosind metoda computeQValueFromValues
   - Q-value reprezinta recompensa cumulativa asteptata in urma luarii unei anumite actiuni din starea curenta si urmand apoi politica optimala
   - noua valoare pentru stare este setata ca fiind Q-value maxim din toate actiunile posibile
2. In momentul in care se completeaza toate iteratiile, valorile agentului vor fi completate cu noile valori calculate

```
def computeQValueFromValues(self, state, action):
    qValue = sum(prob * (self.mdp.getReward(state, action, nextState) +
    ↪  self.discount * self.values[nextState])
                 for nextState, prob in
                 ↪  self.mdp.getTransitionStatesAndProbs(state, action))
    return qValue
```

ComputeQValueFromValues calculeaza Q-value pentru o pereche data de stare-actiune, folosind functia curenta de valoare (self.values). Itereaza toate posibilele stari urmatoare si probabilitatile lor de tranzitie (prob) date in starea si actiunea curenta. Pentru fiecare stare urmatoare, calculeaza recompensa imediata (self.mdp.getReward) si viitoarea valoare discounted (self.discount * self.values[nextState]). Q-value este suma acestor valori pentru toate starile urmatoare posibile.

```
def computeActionFromValues(self, state):
    if self.mdp.isTerminal(state):
        return None

    possibleActions = self.mdp.getPossibleActions(state)
    return max(possibleActions, key=lambda action:
    ↪  self.computeQValueFromValues(state, action))
```

ComputeActionFromValues calculeaza cea mai buna actiune a unei stari date bazat pe functia de valoare curenta. Daca starea este terminala, returneaza "None" deoarece nu

exista actiuni legale in stari terminale. Altfel, ia lista actiunilor posibile pentru starea curenta si returneaza actiunea care maximizeaza Q-value pentru perechea stare-actiune data folosind metoda computeQValueFromValues. Argumentul "key" ii specifica functiei folosite sa extraga o cheie de comparare (in acest caz, Q-value) de la fiecare element.

2 Question 3: Policies

Acest exercitiu presupune alegerea urmatoarelor 3 valori pentru a alege cai optime in diferite conditii:

- **Discount** determina importanta recompenselor viitoare (mai aproape de 1) in comparatie cu recompensele imediate (mai aproape de 0).
- **Noise**: o valoare mai mare creste probabilitatea unor actiuni aleatorii, in timp ce o valoare mai mica pastreaza actiunile deterministice. Aceasta valoare este importanta cand vine vorba de a lua decizia de a urma sau nu cai riscante
- **Living Reward** reprezinta recompensa (sau penalizarea) pe care o primeste agentul pentru fiecare pas ce nu face parte dintr-o stare terminala. Aceasta valoare afecteaza tendinta agentului de a explora, a evita sau a prioritiza anumite cai

Mai jos sunt prezentate conditiile care trebuie indeplinite:

2.1 Prefer the close exit (+1), risking the cliff (-10)

```
def question3a():
    answerDiscount = 0.9
    answerNoise = 0.2
    answerLivingReward = -1.9
    return answerDiscount, answerNoise, answerLivingReward
```

Am setat discount-ul la 0.9 pentru a face agentul să acorde o importanță semnificativă recompenselor imediate, dar fără a neglija complet cele pe termen lung. Am setat noise-ul la 0.2 astfel incat agentul sa nu se indeparteze prea mult de la calea cea mai scurta. Am setat living reward-ul la -1.9 astfel incat sa se aleaga cu prioritate calea cea mai scurta, orice valoare mai mica de atat facand testul sa esueze.

2.2 Prefer the close exit (+1), but avoiding the cliff (-10)

```
def question3b():
    answerDiscount = 0.5
    answerNoise = 0.2
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
```

Am setat discount-ul la 0.5 pentru a acorda atentie recompensei apropiate decât celei pe termen lung. Am setat noise la 0.2 pentru ca agentul sa faca actiuni aleatoare, dar nu suficient de multe incat sa ajunga pe o cale riscanta. Am setat living reward la -1 pentru a face agentul sa ajunga cat mai rapid la starea finala.

2.3 Prefer the distant exit (+10), risking the cliff (-10)

```
def question3c():
    answerDiscount = 0.95
    answerNoise = 0.05
    answerLivingReward = -0.1
    return answerDiscount, answerNoise, answerLivingReward
```

Am setat discount-ul la 0.95 pentru a acorda atentie recompensei indepartate. Am redus noise la 0.05 pentru a face acțiunile agentului mai deterministe și a reduce explorarea aleatorie. Am setat living reward la -0.1 pentru a oferi o mică penalizare pentru timpul petrecut în stări non-terminale, fără a încuraja explorarea inutilă.

2.4 Prefer the distant exit (+10), avoiding the cliff (-10)

```
def question3d():
    answerDiscount = 0.8
    answerNoise = 0.4
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
```

Am setat discount-ul la 0.8 pentru a echilibra importanța recompenselor imediate și cele pe termen lung. Am cresut noise la 0.4 pentru a permite explorarea aleatorie. Totusi, am setat living reward la -1 pentru a încuraja ajungerea rapidă la stările terminale.

2.5 Avoid both exits and the cliff (so an episode should never terminate)

```
def question3e():
    answerDiscount = 0.8
    answerNoise = 0.4
    answerLivingReward = 2
    return answerDiscount, answerNoise, answerLivingReward
```

Am mentinut discount-ul la 0.8 pentru a echilibra importanța recompenselor imediate și cele pe termen lung. Am mentinut noise la 0.4 pentru a permite explorarea aleatorie. Am setat living reward la 2 pentru a a oferi o recompensă mare pentru timpul petrecut în stări non-terminale, încurajând astfel explorarea extinsă.

3 Question 5: Q-Learning (Neimplementat)