UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Reinforcement Learning

Inteligenta Artificiala

Autori: Laura-Luisa Voicu
Grupa: 30236

Facultatea de Automatica
si Calculatoare

11 Ianuarie 2024

Cuprins

1 Introducere

1.1 Overview

1.1.1 Despre Reinforcement Learning

Reinforcement Learning = tehnica de invatare automata prin care un software e invatat sa ia decizii pentru a obtine rezultatele cele mai optime.

Actiunile software-ului ce contribuie la atingerea obiectivului vor fi intarite, in timp ce actiunile ce indeparteaza obiectivul sunt ignorate.

Reinforcement Learning - types:

1. Passive Reinforcement Learning: how to learn from already given experiences
   (a) Model-based: modelul MDP invata din experiente, dupa care rezolva MDP-ul
   (b) Model-free:
      i. Value learning: invata valori dintr-un policy fix
      ii. Q Learning: invata Q valori din policy-ul optimal
2. Active Reinforcement Learning: hot to collect new experiences
3. Approximate Reinforcement Learning: to handle large state spaces

1.2 MDP - Markov Decision Process

Reinforcement Learning foloseste MDP

- Are un set de stari
- Are un set de actiuni per stare
- Are un model T(s,a,s')
- Are o functie de recompensa R(s,a,s')

1.3 MDPs - Instructiuni

Rulare Gridworld (foloseste arrow keys)

```
python3 gridworld.py -m

# agentii se muta random
python3 gridworld.py -g MazeGrid
```

In Gridworld MDP pozitiile sunt reprezentate ca si coordonate (x,y). Default, recompensa este 0; ea poate fi schimbata cu optiunea -r.

2 Question 1: Value Iteration

2.1 Update State

Pentru a updata starea curenta, se va folosi formula:

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

Figura 1: Update next state

Q1 presupune scrierea unui value iteration agent ce reprezinta un **offline planner**, nu un agent de reinforcement learning.

Clasa ValueIterationAgent ca un MDP si ruleaza iteratiile de atatea ori cat a fost specificat.

Se vor computa k pasi ce estimeaza valoarea optimala Vk (in metoda **runValueIteration**. E necesara implementarea :

- computeActionFromValue(state) - are rolul de a computa cea mai buna actiune corespunzatoare valorii din self.values
- computeQValueFromValues(state, action) - returneaza Q-value al tuplei (state, action) corespunzatoare valorii din self.values.

2.2 Comenzi de rulare

```
python3 gridworld.py -a value -i 100 -k 10
python3 gridworld.py -a value -i 5

#Autograder:
python3 autograder.py -q q1
```

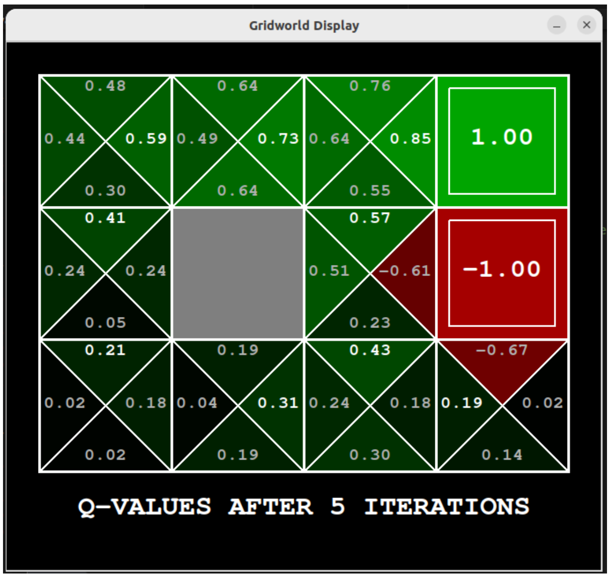

Figura 2: Value Iteration

2.3 Cod

:

```python
class ValueIterationAgent(ValueEstimationAgent):

    def runValueIteration(self):
        for _ in range(self.iterations):
            aux_values = util.Counter()
            for state in self.mdp.getStates():
                if not self.mdp.isTerminal(state):
```

```
                    actions = self.mdp.getPossibleActions(state)
                    max_value_state = self.computeQValueFromValues(state, actions[0])
                    for action in actions:
                        value = self.computeQValueFromValues(state, action)
                        if value > max_value_state:
                            max_value_state = value
                    aux_values[state] = max_value_state
                else:
                    aux_values[state] = 0
            self.values = aux_values


    def computeQValueFromValues(self, state, action):

        #V(k+1)(s) = max(Sum(T(s,a,s')*[R(s,a,s') + gama*V(k)(s')]))
        # s' -  noua stare, V - values, R - reward, gama - factor discount
        q_value = 0
        R = self.mdp.getReward
        gama = self.discount
        V = self.values
        all_states_probs = self.mdp.getTransitionStatesAndProbs(state, action)
        for new_state, prob in all_states_probs:
          q_value += prob * ( R(state, action, new_state) + gama * V[new_state] )
        return q_value


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

        # state neterminal
        policies = [(self.computeQValueFromValues(state, action), action) for action in self

         # returnam cheia pentru care policy e maxim
        return max(policies, key = lambda x : x[0])[1]
```

2.4 Dificultati la implementare

- Intelegerea conteptelor: MDP, Reinforcement Learning, Values / QValues
- Integrarea functilor deja existente prezentate pentru MDP (getTransitionStatesAndProbs, etc.)
- Integrarea efectiva a functiei prezentate mai sus pentru calculul urmatoarei stari.

3 Question 3: Policies

3.1 Context

DiscountGrid - Ce stare terminala alegem?

Exista 2 tipuri de path-uri:

- Risk the cliff (1) - mai scurte dar risca sa castige un payoff negativ mai mare.
- Avoid the cliff (2) - mai lungi dar e mai putin probabil sa gastige payoff negativ mare.

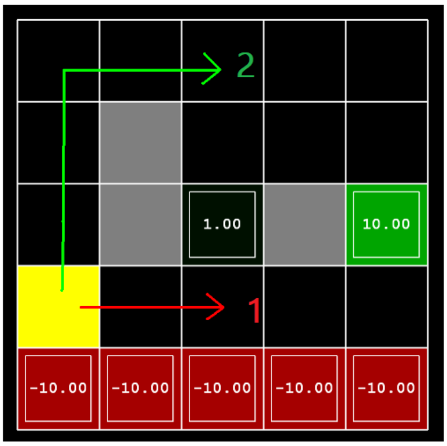

Figura 3: Cliffs

3.2 Cerinta

Setarea discount-ului, noise-ului si a reward-ului astfel incat sa se produca policy-ul optimal pentru diverse tipuri.

Optimal policy types

- Prefer the close exit (+1), risking the cliff (-10)
- Prefer the close exit (+1), but avoiding the cliff (-10)
- Prefer the distant exit (+10), risking the cliff (-10)
- Prefer the distant exit (+10), avoiding the cliff (-10)
- Avoid both exits and the cliff (so an episode should never terminate)

3.3 Comenzi de rulare

```
python3 gridworld.py -g DiscountGrid -a value --discount [YOUR_DISCOUNT] --noise [YOUR_NOISE


#Autograder:
python3 autograder.py -q q3
```

3.4 Cod

```
ef question3a():
    answerDiscount = 0.5
    answerNoise = 0
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward

def question3b():
    answerDiscount = 0.5
```

```python
    answerNoise = 0.25
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward

def question3c():
    answerDiscount = 1
    answerNoise = 0
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward

def question3d():
    answerDiscount = 0.5
    answerNoise = 0.5
    answerLivingReward = 0.5
    return answerDiscount, answerNoise, answerLivingReward

def question3e():
    answerDiscount = 1
    answerNoise = 0
    answerLivingReward = 1
    return answerDiscount, answerNoise, answerLivingReward

def question8():
    answerEpsilon = None
    answerLearningRate = None
    return answerEpsilon, answerLearningRate

```

3.5 Justificari

- Question3a
  - answerDiscount: 0.5 - accent moderat pe reward imediat
  - answerNoise: 0 - actiuni cunoscute, nu random
  - abswerLivingReward: -1 penalizarea agentului daca continua
  - concluzie: Agentul e focusat pe reward imediat si primeste penalizare daca contiuna. Setarile implica tinderea spre castiguri de scurta durata, dar adoptand o abordare prudenta (answerNoice e 0).
- Question3b
  - answerDiscount: 0.5 - accent moderat pe reward imediat
  - answerNoise: 0.25 - pot aparea decizii random in actiunea agentului
  - abswerLivingReward: -1 - penalizarea agentului daca continua
  - concluzie: Agentul manifesta un echilibru intre recompensele imediate si cele indepartate.
- Question3c
  - answerDiscount: 1 - trateaza in mod egal recompensele imediate si cele indepartate
  - answerNoise: 0 - nu exista actiuni random in actiunile agentului.
  - abswerLivingReward: -1 - penalizarea agentului daca continua

- concluzie: Agentul acorda aceeasi prioritate recompenselor imediate si indepartate, dar absenta Noise-ului poate duce la abordare prudenta intrucat actiunile viitoare sunt cunoscute.

- Question3d
  - answerDiscount: 1 - trateaza in mod egal recompensele imediate si cele indepartate
  - answerNoise: 0.5 - apar decizii random in actiunea agentului
  - abswerLivingReward: 0.5 - se ofera o recompensa povitiva pentru existenta continua a a gentului
  - concluzie: Agentul explora ma mult si e mai dispus la riscuri.
- Question3e
  - answerDiscount: 1 - trateaza in mod egal recompensele imediate si cele indepartate
  - answerNoise: 0 - actiuni cunoscute, nu random
  - abswerLivingReward: 1 - se ofera o recompensa povitiva pentru existenta continua a a gentului
  - concluzie: Agenntul are o existenta prelungita, iar recompensele indepartate si imeditate sunt tratate in mod egal.

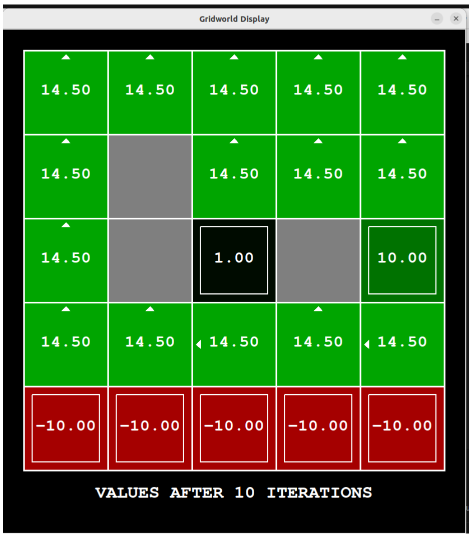

Figura 4: Values

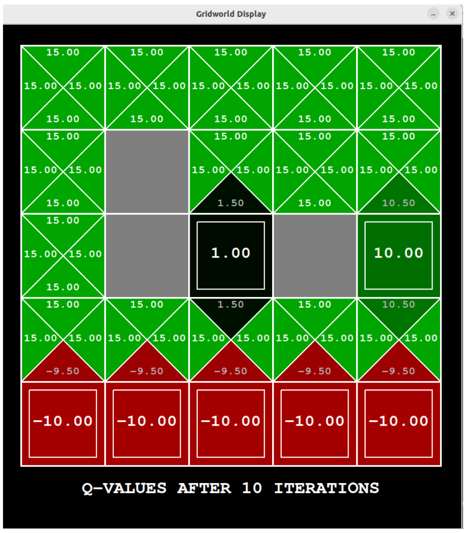

Figura 5: QValues

4 Question 5: Q-Learning

4.1 Context

Agentul nu invata efectiv din experienta, ci doar din modelul MDP. Cand agentul interactioneaza cu environment-ul, el doar urmeaza policy-urile setate.

Agentul Q-learning invata din incercarile si erorile interactiunilor cu environmen-ul prin metoda **update**.

4.2 Comenzi de rulare

```
python3 gridworld.py -a q -k 5 -m



#Autograder:
python3 autograder.py -q q5
```

4.3 Cod

```python
class QLearningAgent(ReinforcementAgent):

    def __init__(self, **args):
        self.values = util.Counter()

    def getQValue(self, state, action):
        return self.values[state, action]

    def computeValueFromQValues(self, state):
        actions = self.getLegalActions(state)
```

```
        if not actions:
            return 0
        return self.getQValue(state, self.getPolicy(state))

    def computeActionFromQValues(self, state):
        actions = self.getLegalActions(state)
        if not actions:
            return 0
        return max(actions, key=lambda action: self.getQValue(state, action))

    def getAction(self, state):
        legalActions = self.getLegalActions(state)
        action = None
        return action

    def update(self, state, action, nextState, reward):
        newValue = (1 - self.alpha) * self.getQValue(state, action) + self.alpha * (reward +
        self.values[state, action] = newValue

    def getPolicy(self, state):
        return self.computeActionFromQValues(state)

    def getValue(self, state):
        return self.computeValueFromQValues(state)
```