AI-programmering

Inlämningsuppgift 4

Deadline: *11e Oktober - 23:59*

0. Introduktion

I denna inlämningsuppgift skall ni implementera Q-learning på exempel från python biblioteket Gymnasium. Eftersom vi använder Gymnasium så skall ni använda Python som programmeringsspråk. Implementationen skall vara eran från grunden, således får ni inte använda några biblitotek för maskininlärning. Mer information nedan.

Denna uppgift är uppdelad i två delar:

- För betyget 3 skall:
  - Q-learning implementeras för att lösa "Frozen Lake 8x8" exemplet i Gymnasium
  - Agenten skall kunna hitta optimal lösning (den skall alltså kunna hitta snabbaste rimliga vägen)
- För betyget 5 skall:
  - Q-learning implementeras för att lösa antingen "Acrobot" eller "Lunar lander" exemplet i Gymnasium, båda har kontinuerlig observation (state) space
  - En linjär (eller icke-linjär) funktionsapproximator skall implementeras från grunden (en simpel funktion för att ta intervaller av kontinuerliga värden och göra om dem till heltal räcker t.ex.) för att göra om observation space till diskret.

0.1 Administrativt

- För inlämning skall källkod skickas in på Canvas
- Python skall användas
- Ingen särskild prestanda metrik behöver implementeras. Det räcker med att visa att agenten gör rätt i spelfönstret.
- Maskininlärningsbibliotek får ej användas (t.ex. Sci-kit, Torch, Tensorflow, Caffe, mlpack, OpenNN, OpenCV etc.)
  - Algoritmerna och ekvationer skall vara egenskrivna
  - Bibliotek för simpla matematiska operationer får användas (t.ex. standardbibliotek, Numpy etc.)
- För betyg 3-uppgiften krävs att agenten hittar optimal lösning
- För betyg 5-uppgiften krävs inte optimal lösning, men den skall kunna lösa uppgiften ofta nog att det är tydligt att den har lärt sig något nyttigt

0.2 Open AI Gymnasium

OpenAI Gymnasium är ett Python bibliotek för reinforcement learning. Biblioteket har definierade omgivningar med skapade rewardfunktioner som är redo att köra. Allt ni behöver för att använda biblioteket finns på denna länk:

https://gymnasium.farama.org/content/basic_usage/

```python
import gymnasium as gym
env = gym.make("LunarLander-v2", render_mode="human")
observation, info = env.reset()

for _ in range(1000):
    action = env.action_space.sample()  # agent policy that uses the observation and info
    observation, reward, terminated, truncated, info = env.step(action)

    if terminated or truncated:
        observation, info = env.reset()

env.close()
```

Ovan syns den grundläggande strukturen av att använda Gymnasium. Omgivningen skapas av *gym.make* och *render_mode* bestämmer om körningen skall visas eller inte. $render\_mode =$ "*human*" visar körningen men är långsam och $render\_mode = None$ visar den ej och är snabb så den passar bättre för träning.

env.reset() återställer omgivningen tillbaka till grundförutsättningarna. $env.action_space.sample()$ returnerar en random action från action space. *env.step*(*action*) stegar omgivningen till nästa state med action. *terminated* och *truncated* är flaggor för att avsluta en körning.

0.3 Labbkod

För labben får ni har ni 4 filer att jobba med. *qlearning.py* innehåller skelettkod för eran algoritm. *frozen_lake.py* innehåller skelettkod för Uppgift 1 (betyg 3). *continous.py* innehåller skelettkod för Uppgift 2 (betyg 5). qlearning.py importeras smidigast in i frozen_lake.py och continous.py när ni vill använda eran algoritm.

requirements.txt innehåller de bibliotek ni behöver installera för labben. Om ni använder Anaconda är det smidigast att skapa en ny conda environment och installerar biblioteken där, instruktioner finns i "IF YOU ARE USING ANACONDA.txt".

0.4 Q-learning pseudokod

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return a

Figure 21.8 An exploratory Q-learning agent. It is an active learner that learns the value $Q(s, a)$ of each action in each situation. It uses the same exploration function f as the exploratory ADP agent, but avoids having to learn the transition model because the Q-value of a state can be related directly to those of its neighbors.