Umjetna inteligencija – Laboratorijska vježba 3

UNIZG FER, ak. god. 2015/16.

Zadano: 24.5.2016. Rok predaje: 30.5.2016. do 23.59 sati.

Uvod

U okviru ove laboratorijske vježbe rješavati ćete probleme iz područja potpornog učenja. Kod koji ćete koristiti možete skinuti kao zip arhivu na stranicama fakulteta ovdje, ili na github repozitoriju predmeta ovdje.

Pri rješavanju ove laboratorijske vježbe imate dostupan *autograder* - pokretanjem naredbe `python autograder.py` možete vidjeti valjanost vaših rješenja. Ukoliko želite pokrenuti autograder za neko specifično pitanje, možete to napraviti pomoću argumenta -q, kao u idućem primjeru: `python autograder.py -q q1`.

Kod za projekt se sastoji od Python datoteka, dio kojih ćete trebati pročitati i razumjeti za implementaciju laboratorijske vježbe, dio koji ćete trebati samostano uređivati te dijela koji ćete moći ignorirati. Nakon raspakiranja zip arhive, opis datoteka i direktorija koje ćete vidjeti je u nastavku:

Datoteke koje ćete uređivati:	
valueIterationAgents.py	Logika algoritma iteracije vrijednosti
qlearningAgents.py	Logika algoritama baziranih na Q-učenju
Datoteke koje trebate proučiti:	
mdp.py	Opis općenitog Markovljevog procesa odlučivanja
learningAgents.py	Logika koja opisuje agente
util.py	Korisne pomoćne strukture i metode
gridworld.py	Opis GridWorlda
featureExtractors.py	Datoteka koja ekstrahira značajke za Q-stanja
Pomoćne datoteke koje možete ignorirati:	
environment.py	"Apstraktna klasa" za probleme potpornog učenja
graphicsUtils.py	Pomoćne funkcije za grafičko sučelje
graphicsGridworldDisplay.py	Grafički prikaz GridWorlda
crawler.py	Logika "puzača"
graphicsCrawlerDisplay.py	Grafički prikaz puzača
autograder.py	Pomoćna datoteka koja pokreće testove

Kod laboratorijskih vježbi je preuzet s predmeta "Uvod u umjetnu inteligenciju" na Berkeleyu, koji je velikodušno ustupio funkcionalno okruženje drugim fakultetima na korištenje u njihovim predmetima. Kod je napisan u Pythonu 2.7, koji za pokretanje laboratorijske vježbe trebate instalirati.

Nakon skidanja i raspakiravanja koda te pozicioniranja konzolom u direktorij u kojemu ste raspakirali sadržaj arhive, možete isprobati igru s ručnim kontroliranjem pomoću strelica upisujući:

```
python gridworld.py -m
```

GridWorld je svijet u kojemu vam je kretanje otežano na način da imate šansu od 80% za napraviti akciju koju ste odabrali, te šansu od 20% za nasumičnu akciju. Vjerojatnost nasumičnog kretanja (šum, engl. noise) možete podesiti pomoću parametra '-n (noise)', na primjer:

```
python gridworld.py -m -n 0.5
```

S čime će šansa nasumičnog kretanja biti 50%. Ovo, kao i sve druge parametre možete vidjeti pozivom parametra '-h (help)'. Ispis je u nastavku:

```
Options:
  -h, --help            show this help message and exit
  -d DISCOUNT, --discount=DISCOUNT
                        Discount on future (default 0.9)
  -r R, --livingReward=R
                        Reward for living for a time step (default 0.0)
  -n P, --noise=P       How often action results in unintended direction
                        (default 0.2)
  -e E, --epsilon=E     Chance of taking a random action in q-learning
                        (default 0.3)
  -l P, --learningRate=P
                        TD learning rate (default 0.5)
  -i K, --iterations=K  Number of rounds of value iteration (default 10)
  -k K, --episodes=K    Number of epsiodes of the MDP to run (default 1)
  -g G, --grid=G        Grid to use (case sensitive; options are BookGrid,
                        BridgeGrid, CliffGrid, MazeGrid, default BookGrid)
  -w X, --windowSize=X  Request a window width of X pixels *per grid cell*
                        (default 150)
  -a A, --agent=A       Agent type (options are 'random', 'value' and 'q',
                        default random)
  -t, --text            Use text-only ASCII display
  -p, --pause           Pause GUI after each time step when running the MDP
  -q, --quiet           Skip display of any learning episodes
  -s S, --speed=S       Speed of animation, S > 1.0 is faster, 0.0 < S < 1.0
                        is slower (default 1.0)
  -m, --manual          Manually control agent
  -v, --valueSteps      Display each step of value iteration
```

Kao što smo obradili na predavanjima, problem koji rješavamo u GridWorldu i sličnim problemima je optimizacija ukupne korisnosti (engl. utility) kroz kretanje po mapi. Ono što vam otežava kretanje po mapi je šum pri kretanju - akcije koje odaberete vas ne moraju nužno voditi na ista mjesta. Ovo se u okviru GridWorlda manifestira na način da imate fiksnu šansu $(1 - n)$ za kretati se u smjeru koji ste odabrali, te šansu n za kretati se nasumično.

Kao i inače, pozicije su definirane pomoću koordinata (x,y), dok su prijelazi između pozicija definirani pomoću strana svijeta (south, east, north, west).

Laboratorijska vježba se sastoji od četiri podzadatka, od kojih svaki nosi po pet bodova. Ukupna suma od 20 konačnih bodova će se potom skalirati na 6.25.

Problem 1: Iteracija vrijednosti (5 bodova)

U datoteci valueIterationAgents.py nadopunite kod za iteraciju vrijednosti. Podsjetimo se, iteracija vrijednosti je algoritam koji pokušava izračunati vrijednosti svakog stanja na mapi pomoću rekurzivne formule za vrijednost stanja:

$$V_0(s) = 0$$
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

Kao parametar u konstruktoru vašeg *ValueIterationAgent*-a dobivate argument *mdp*, tj. definiciju markovljevog procesa odlučivanja (u slučaju GridWorlda, to je implementacija apstraktne klase *mdp.MarkovDecisionProcess* se nalazi u *gridworld.py*. Metode koje opisuju markovljev proces odlučivanja su:

`mdp.getStates():`

Vraća listu svih mogućih stanja (list(tuple(int, int))) u problemu

`mdp.getPossibleActions(state):`

Vraća listu svih mogućih akcija (list(str)) za stanje state

`mdp.getTransitionStatesAndProbs(state, action):`

Vraća listu parova (tuplea) koji se sastoje od elemenata (idućeStanje : tuple(int, int), vjerojatnostPrijelaza: float)

`mdp.getReward(state, action, nextState):`

Vraća float preciznost vrijednost funkcije nagrade za prijelaz iz stanja state uz akciju a u stanje nextState

`mdp.isTerminal(state):`

Provjerava je li neko stanje konačno (vraća True ili False). Konačna stanja nemaju prijelaza (dakle lista prijelaza može biti prazna)!

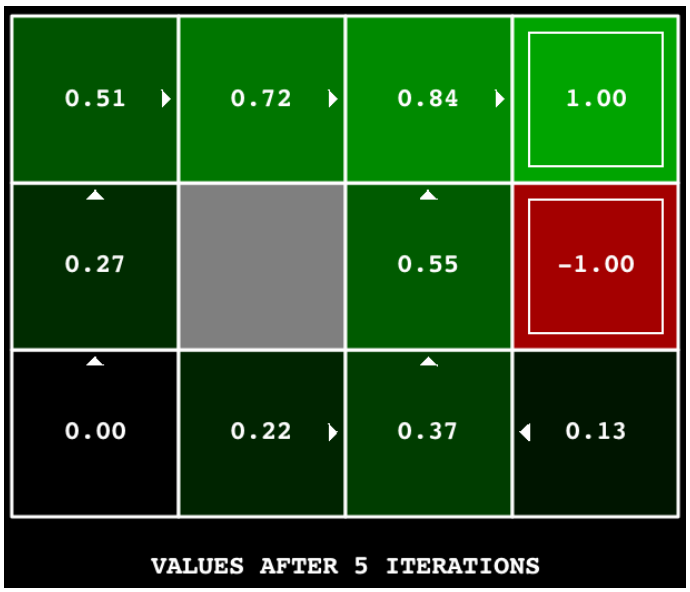
Metode koje trebate dovršiti su: *__init__*, *computeQValueFromValues*, *computeActionFromValues*. Kako imate preddefinirane vrijednosti prijelaza i nagrada, većina računanja bi trebala biti izvršena u inicijalizaciji. Pri rješavanju koristite prethodno inicijalizirani razred *util.Counter*, koji je zapravo implementacija Pythonovog riječnika (engl. dictionarya), s početnim vrijednostima postavljenim na nulu - dakle, ne morate inicijalizirati vrijednosti za svako stanje.

Opaska 1.1 Pazite na to da pri ažuriranju vrijednosti koristite pomoćnu strukturu u koju ćete spremati međurezultat - u koraku $k+1$ iteracije vrijednosti V_{k+1} ovise samo o vrijednostima iz koraka V_k - pazite da ne koristite vrijednosti koje ste ažurirali u tom koraku.

Kako biste testirali implementaciju, rezultat pokretanja iduće naredbe:

```
python gridworld.py -a value -i 5
```

bi vam trebao izgledati kao na idućoj slici:



Slika 1: Primjer izvođenja iteracije vrijednosti

Problem 2: Q-učenje (5 bodova)

U datoteci qlearningAgents.py nadopunite kod za algoritam Q-učenja. Podsjetimo se, algoritam Q-učenja radi na principu pokretnog prosjeka (engl. running average), te formula za ažuriranje izgleda kao u nastavku:

$$Q_0(s,a) = 0$$
$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + (\alpha)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)]$$

Pri čemu je α stopa učenja (engl. learning rate), dok je γ faktor propadanja (engl. decaying factor).

Metode koje trebate nadopuniti u razredu QLearningAgent su: *__init__* , *getQValue*, *computeValueFromQValues*, *computeActionFromQValues*, *getAction* i *update*. Razred QLearningAgent ima varijable koje nasljeđuje od ReinforcementAgenta, koje eksplicitno ne vidite iako postoje. Varijable koje su nasljeđene, a trebaju vam su: *self.epsilon* - vjerojatnost za istraživanje u epsilon-greedy pristupu (Tek u zadatku 3.), *self.alpha* - stopa učenja, te *self.discount* - stopa propadanja. Sve vrijednosti su u float preciznosti.

Funkcija koja je dostupna kroz nasljeđivanje, iako ju eksplicitno ne vidite je `self.getLegalActions(state)`, koja vam za predano stanje vraća listu mogućih akcija.

Opaska 2.1 Pripazite na činjenicu da ako u nekom stanju Q-stanje koje ste jedino istražili ima negativnu vrijednost, optimalan potez je jedan od onih koji još niste istražili! Ti potezi ne moraju nužno biti inicijalizirani u vašem riječniku (Counteru). Ukoliko imate više Q-stanja s istim vrijednostima, za optimalan potez odaberite bilo koje od njih s uniformnom vjerojatnošću. Za ovo je korisna funkcija `random.choice()`, koja za argument prima niz elemenata i vraća jedan od njih s uniformnom vjerojatnošću.

Opaska 2.2 Pripazite na to da kad god pristupate Q-vrijednostima, koristite metodu `getQValue` budući da će vam to olakšati rješavanje približnog Q-učenja.

Opaska 2.3 Kao ključeve vašeg riječnika (dictionary / Counter) možete koristiti i tupleove (parove stanje, akcija).

Problem 3: ϵ-pohlepan pristup (5 bodova)

Modificirajte vaš algoritam Q-učenja pomoću implementacije ϵ-pohlepnog pristupa (engl. ϵ-greedy). Podsjetimo se, ϵ-pohlepnost znači da s malom vjerojatnosti ϵ odabirete nasumičnu akciju, koja može biti bilo koja od legalnih akcija, dok s vjerojatnošću $1 - \epsilon$ odabirete trenutno optimalnu akciju.

Vrijednost ϵ vam je dostupna kao varijabla razreda *self.epsilon*, dok bi vam se korisnom trebala pokazati metoda util.flipCoin(p), koja vraća True s vjerojatnošću p, te False s vjerojatnošću $1 - p$.

Bez ikakvih dodatnih izmjena u kodu nakon dodavanja ϵ-pohlepnog pristupa, provjerite kako vam funkcionira puzač (engl. crawler) pokretanjem idućeg programa:

```
python crawler.py
```

Problem 4: Približno Q-učenje (5 bodova)

Implementirajte približno Q-učenje u datoteci qlearningAgents.py u razredu *ApproximateAgent*.

Prisjetimo se, približna Q-funkcija je u obliku:

$$Q(s,a) = \sum_{i=1}^{n} f_i(s,a) w_i$$

Pri čemu je W vektor težina, dok je $f(s,a)$ vektor značajki za Q-stanje (s,a), a svaka težina w_i iz vektora se mapira na jednu značajku. Funkcije koje će računati značajke su već definirane i implementirane u datoteci *featureExtractors.py*, te varijabla razreda *self.featExtractor* koja vam je dostupna na poziv metode *getFeatures(state, action)* vraća Counter značajki za to Q-stanje.

U početku postupka težine možete inicijalizirati na 0 (kako je ovo najlakše riješiti? hint:Counter), te ih potom ažurirajte po jednadžbi s predavanja:

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$
$$difference = (r + \gamma \max_{a'} Q(s',a')) - Q(s,a)$$

Ukoliko vaša implementacija radi, bez problema bi trebali pobjeđivati iduću mapu:

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor
-x 50 -n 60 -l mediumGrid
```

Dok bi s (moguće negdje do minutu treniranja), vaš agent trebao bez problema pobjeđivati i nešto veću mapu:

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor
-x 50 -n 60 -l mediumClassic
```