Sztuczna inteligencja
Pracownia 5
Termin: koniec semestru
(z dodatkowym terminem w sesji)

Możliwe jest oddawanie zadań dotyczącuch jednej gry z poprzedniej listy bez straty punktowej. Jako 'punkty do zdobycia' liczymy dla tej listy 15 (ale można zdobyć ich oczywiście dużo więcej[1]).

Zadanie 1. (Xp) Rozwiąż zadanie z wcześniejszych list, którego nie robiłeś (lub popraw punktację zadania, które robiłeś). Uwaga: to zadanie nie jest zadaniem z listy p5 w rozumieniu zasad zwalniania z egzaminu.

Zadanie 2. (5+p) Miejsce na obiecane łatwe zadanie

Zadanie 3. (4+4p) Napisz agenta, grającego w szachy. Do generacji ruchów wykorzystaj jakąś bibliotekę szachową (np. python-chess). Do zaliczenia zadania wymagane są jedynie dwie rzeczy:

a) Musisz napisać funkcję heurystyczną, oceniającą ruchy. Funkcja ta powinna oceniać **nie tylko** przewagę materialną. Funkcja powinna być wykorzystana (albo w MiniMax-ie, albo w MCTS – do symulacji, lub jak w AplhaZero[2])

b) Dołącz do agenta jakąś formę przeszukiwania drzewa gry (np. MCTS lub MiniMax)

Dodatkowe punkty są za:

- Wykorzystanie księgi otwarć (1p)
- Wykorzystanie bazy końcówek szachowych (1p)
- Dodatkowe elementy funkcji heurystycznej (0.5p za każdy element, max 2p). Przykładowo, student, w którego funkcji oceniającej będzie przewaga materialna, ruchliwość, ocena struktury pionów i szacowanie zagrożenia króla dostanie 1p (bo przewaga materialna i 1 dodatkowy element są wymagane, pozostałe dwa – dodatkowe)

Oczywiście nie wolno wywoływać zewnętrznych silników szachowych.

Zadanie 4. 1-6p Wystartuj w turnieju z programem grającym w grę X (gdzie X to szachy, dżungla lub reversi). Można startować w większej liczbie turniejów, wówczas punkty się sumują[3]. W turnieju uczestniczy program studenta, bot (lub boty) dostarczone przez wykładowcę oraz (opcjonalnie) programy innych studentów. Uczestnictwo w turnieju jednoosobowym daje 1p, w takim, w którym jest więcej uczestników – dwa punkty. Turnieje organizują studenci, podając następnie w odpowiednim formacie ich wyniki. Wykładowca ustala ranking uczestników stosując ranking szachowy Elo (uwzględniający wszystkie wyniki meczów). Znalezienie się wyżej w rankingu od botu wykładowcy daje 2p (jeżeli botów będzie więcej, mogą być tańsze i droższe, ale na pewno przynajmniej 1 będzie 2-punktowy). Oprócz tego pozycje w rankingu są dodatkowo punktowane, a złoty medal daje zwolnienie z egzaminu z oceną (ćwiczenia + 0.5), srebrny – z oceną z ćwiczeń, a brązowy – z oceną z ćwiczeń, o ile jest co najmniej 4.0. **Uwaga:** boty też dostają medale!

Szczegóły będą podane w osobnym dokumencie.

Zadanie 5. (5p) Rozważamy prosty model autka, poruszającego się po dyskretnym torze (czyli podczas ruchu auto przeskakuje z kratki na kratkę). Stan autka dany jest przez cztery liczby całkowite: pozycję (x,y) oraz prędkość (vx, vy), przy czym każda składowa prędkości należy do zbioru $\{-3,-2,-1,0,1,2,3\}$. Akcja jest parą (dvx, dvy) mówiącą o tym, jak chcemy zmienić prędkość. Przyrost składowej prędkości należy do zbioru $\{-1,0,1\}$, czyli mamy 9 możliwych akcji.

Podstawowa mechanika autka dana jest zatem programem:

---

[1] Na liście C5 będzie zadanie, z treścią: przeczytaj listę p4 i wyjaśnij ewentualnym zainteresowanym, o co chodzi w kolejnych zadaniach :)

[2] Na wykładzie 12 będzie o tym mowa pod koniec

[3] A dokładniej, sumują się maksima punktów dla pojedynczych rodzajów gier. Jeżeli Ania wystartowała w dwóch turniejach szachowych: jednym z Bartkiem, a drugim z Celiną i Darkiem, a oprócz tego zmierzyła jakość swojego agenta grającego w Dzunglę konfrontując go z botem Wykładowcy, wówczas dostanie 3 punkty

```
x,y,vx,vy = state
dvx,dvy = action   # (*)
vx += dvx
vy += dvy
if vx > 3: vx = 3
if vy > 3: vy = 3
if vx < -3: vx = -3
if vy < -3: vy = -3
x += vx
y += vy
new_state = x,y,vx,vy
```

Autko, które zakończy ruch na polu `e` kończy jazdę, wygrywa wyścig (i dostaje nagrodę 100), auto, które zakończy ruch na poza drogą (na polu `'.'`), również kończy jazdę, ale z nagrodą równą -100. Jeżeli autko rozpoczyna ruch na polu z rozlanym olejem (`'o'`), wówczas jego prędkość ulega zaburzeniu, realizowanemu w ten sposób, że w miejscu `(*)` wykonywany jest kod:

```
dvx += random.choice( [-1,0,1])
dvy += random.choice( [-1,0,1])
```

Auto startuje na wskazanym polu (na mapce oznaczonym przez `'s'`), z zerową prędkością. Napisz program, który wyznacza *optymalną politykę* dla autka, czyli funkcję, która dla każdego możliwego stanu wyznacza akcję. Politykę zapisujemy w formacie tekstowym składającym się z wierszy zawierających 6 liczb całkowitych (stan i akcję, którą należy wykonać w danym stanie):

```
x y vx vy   dvx dvy
```

Zakładamy ponadto, że $\gamma = 0.99$ (przypominam, że wypłata jest mnożona przez γ^t), a koszt jednego ruchu to 1.0.

Procedura testowania programu będzie następująca:

a) Program studenta wyznacza politykę dla konkretnego toru i zapisuje ją do pliku (w utworzonej przez studenta kartotece, w której są również mapy torów). Plik z polityką ma nazwę `policy_for_<task>.txt`.

b) Program `test.sh` przegląda wszystkie polityki (i tory) w danej kartotece,

c) wykonując dla każdego 20 testów i informując, czy wyniki są satysfakcjonujące. Przebiegi testów są wizualizowane.

d) Dodatkowo zakładamy, że brak informacji o jakimś stanie powoduje wykonanie akcji (0,0). Wykonuje się co najwyżej 1000 kroków.

Na stronie wykładu znajdują się (spakowane) tory, wraz ze skryptami oceniającymi. Jest też przykładowa polityka dla jednego z zadań. Polityka wyznaczana jest off-line (sprawdzaczka nie kotroluje czasu). Można politykę wyznaczyć przed zajęciami, wymaga się jedynie, by dla każdego przypadku testowego student mógł powtórzyć obliczenia polityki przy prowadzącym (czyli w czasie zajęć).

Zadanie 6. (5p) W zadaniu tym rozważamy inny niż w poprzednim zadaniu, deterministyczny mechanizm działania samochodu. Stanem samochodu jest również położenie (x,y) oraz prędkość (v, α), gdzie α jest kątem między osią X a wektorem prędkości. Prędkość i kąt są dyskretne (w jednym ruchu można zmienić każdą z nich o co najwyżej 1), ale położenie wyrażane jest liczbami rzeczywistymi, co sprawia, że wyliczenie stanów najprawdopodobniej przestaje wchodzić w grę. Mamy następujące akcje:

```
.   - toczenie się
r.  - skręt w prawo
```

```
l.  - skręt w lewo
a.  - przyśpieszenie
ar. - przyśpieszenie i skręt w prawo
al. - przyśpieszenie i skręt w lewo
b.  - hamowanie
br. - hamowanie i skręt w prawo
bl. - hamowanie i skręt w lewo
```

które, jak widać, można sklejać w 1 napis i dzielić ze względu na kropki. Ponieważ model jest deterministyczny, a obliczenia mogą być długotrwałe, oddając zadanie należy:

a) zaprezentować ciąg akcji (w pliku `actions_for_<tast-name>`), który doprowadza do mety dla każdego przypadku testowego (będzie udostępniony symulator, który sprawdzi poprawność ścieżki i jej długość)

b) umieścić wszystkie pliki z akcjami i torami w jednej kartotece, uruchomić plik `test.sh`.

c) być gotowym, jeżeli prowadzący poprosi, o powtórzenie części obliczeń

Dokładny opis mechaniki samochodu znajduje się w pliku `char_model2.py`.

Zadanie 7. (od 10p do 15p) Rozważamy inny model autka (`char_model3.py`), w którym położenie zmienia się w nieco bardziej realistyczny sposób (choć autko sterowane jest tymi samymi poleceniami, co w poprzednim zadaniu). Najważniejsze różnice:

a) Zarówno prędkość, jak i kąt są również zmiennymi rzeczywistymi,

b) uwzględnione są opory, związane z tarciem (składowa stała) oraz z oporami powietrza (zależnymi od prędkości i kwadratu prędkości)

c) to, o ile skręcimy wykonując akcję `l` lub `r` zależy od prędkości

d) Akcje mają b.dużą rozdzielczość czasową: na przykład do rozpędzenia autka do prędkości rzędu pół piksela na cykl potrzebne jest kilkadziesiąt cykli przyśpieszania.

Jest też jeszcze jedna różnica: zadanie tymczasowo nie ma rozwiązania wzorcowego. Jest wyznaczona ręcznie trasa dla `task2.txt`, która jest mocno nieoptymalna (kilka razy autko prawie się zatrzymuje, żeby zmieścić się w zakręcie), a ma długość 1800 akcji.

Warunkiem zaliczenia zadania jest napisanie programu, który działa dla różnych torów, a dla toru z `task2.txt` generuje co najwyżej 1700 akcji. Zadanie warte jest wówczas 10 punktów. Za każdą kolejną pięćdziesiątkę poniżej 1700 dla tego zadania student dostaje 0.25p premii, premia nie może przekroczyć 5 punktów.

Zadanie 8. (5p) Rozważamy grę w Dżunglę, potraktowaną jako MDP w dwóch wariantach:

a) oponent wykonuje ruch losowy (losowany z rozkładem jednostajnym ze wszystkich możliwych ruchów)

b) sprawdzamy, czy istnieje ruch w stronę jamy (tzn. taki, że odległość manhatańska pewnej bierki od jamy przeciwnika w wyniku tego ruchu ulegnie zmniejszeniu). Jeżeli takiego ruchu nie ma, to wykonujemy ruch losowy. Jeżeli taki ruch jest, to rzucamy uczciwą monetą. Jeżeli wypadnie orzeł, to wykonujemy ruch losowy, jeżeli wypadnie reszka to wykonujemy ruch wylosowany z ruchów w stronę jamy.

W obu wariantach gramy do końca gry, nagroda wynosi $+1$ lub -1 (zależnie od tego, kto wygrał), współczynnik $\gamma = 0.99$, oponent porusza się jako drugi.

Twoim zadaniem będzie zastosowanie algorytmu Q-learning w celu wyuczenia agenta (agentów?) osiągającego dodatnią wartość oczekiwaną wypłaty w obu wariantach. Powinieneś przybliżać funkcję Q za pomocą funkcji liniowej określonej na cechach pary (stan, ruch). Cechy powinny uwzględniać bicie i dodatkowo w jakiś sposób różnicować bierki.

Zaprezentuj nauczone parametry i skomentuj ich zgodność (lub niezgodność) z Twoimi intuicjami dotyczącymi tej gry.

Zadanie 9. (5p) Rozważamy grę Connect 4[4]. Na stronie znajduje się kod w Pythonie, realizujący losowe pojedynki w tej grze (możesz, jeżeli chcesz, z niego korzystać). Twoim zadaniem będzie zastosowanie algorytmu *TD-learning* w celu wyuczenia funkcji oceniającej sytuację na planszy (podobnie jak w poprzednim zadaniu, funkcja powinna być liniowa). Przy generowaniu rozgrywek obaj gracze powinni posługiwać się aktualnym wariantem funkcji oceniającej, wybierając albo ruch o maksymalnej wartości (z p-stwem $1-\varepsilon$), albo ruch losowy (z p-stwem ε). Podobnie jak w zadaniu z Reversi z poprzedniej listy, aby zaliczyć zadanie trzeba wyuczyć agenta[5], który będzie wygrywał co najmniej 75% pojedynków z agentem grającym losowo (na początku każdej rozgrywki jest losowany gracz rozpoczynający). Można dostać punkt premii, jeżeli nasz program będzie wygrywał w 99% przypadków (co może wymagać lekkiej modyfikacji schematu uczącego, w stosunku do oryginalnego sformułowania TD-learning).

Zadanie 10. (5p) W zadaniu tym powinieneś opracować (być może korzystając ze źródeł, albo z uwag na wykładzie 12) algorytm wykrywania nieprawidłowości (anomaly detection) i przetestować go w dwóch przypadkach:

a) danych MNIST

b) danych MNIST, do których dorzuciłeś kilka celowo niepasujących obrazków (domek, buźka, jakaś literka, grzybek, samolocik, pozioma kreska, cztery kółka, ...)

W obu przypadkach zaprezentuj znalezione „anomalie" (możesz korzystać z biblioteki matplotlib, albo dowolnego innego narzędzia). Dla przypadku drugiego określ skuteczność w wynajdywaniu „obcych" obrazków wśród cyfr.

Jeśli chodzi o obsługę danych MNIST możesz korzystać z dowolnej biblioteki i/lub kodu znalezionego w Internecie (oczywiście powinieneś go rozumieć). Jeżeli chcesz, możesz korzystać z zaimplementowanego w dowolnej bibliotece (np. sklearn) algorytmu klasteryzacji (takiego jak K-średnich). Samo wykrywanie nieprawidłowości powinno być wykonane za pomocą Twojego kodu (oczywiście nie musisz wykonywać klasteryzacji, jeżeli Twój algorytm jej nie wymaga).

Zadanie 11. (do 15p) Miniprojekt: Zaproponuj prowadzącemu trudne zadanie, które chciałbyś rozwiązać. Na stronie wykładu znajdziesz przykładowe propozycje, ale możesz też zgłosić inną

[4] https://en.wikipedia.org/wiki/Connect_Four

[5] W teście agent ma prawo cały czas wykonywać optymalne ruchy, losowanie dotyczy jedynie procesu uczenia.