

Sztuczna inteligencja
Pracownia 5
Termin: koniec semestru
(z uwzględnieniem dodatkowego terminu podczas sesji)

Jako 'punkty do zdobycia' liczymy dla tej listy 15 (ale można zdobyć ich oczywiście dużo więcej¹).
Uwaga: co najmniej jedno zadanie egzaminacyjne będzie miało związek z tą listą pracowniową.
Zadania podzielone są na kilka podsekcji

1 Zadania bez szczególnego tematu

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) Mamy 24 kwadraty o bokach 1, 2, 3, ..., 24. Należy napisać program, który rozmieszcza je na kwadratowej planszy o boku 70, w ten sposób, aby były spełnione następujące warunki:

1. współrzędne lewego górnego rogu każdego kwadratu są liczbami całkowitymi
2. każdy kwadrat w całości mieści się na planszy
3. kwadraty nie nachodzą na siebie (ale mogą się stykać, rogami lub bokami)
4. boki rozmieszczanych kwadratów są równoległe do odpowiednich boków planszy
5. możliwie jak najmniejsza powierzchnia planszy zostaje niepokryta kwadratami

Twój program powinien wypisywać proponowane rozwiązanie w następującym formacie:

w pierwszym wierszu powinna być wypisana liczba niezapełnionych kwadracików po niej powinno być 70 wierszy po 70 znaków przedstawiających rozmieszczenie kwadratów każdy znak jest kropką (oznaczającą brak kwadratu) lub literą 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' ('A' oznacza przynależność do kwadratu o boku 1, 'B' – o boku 2, itd)

Punktacja za zadanie jest następująca (punkty się sumują):

- 2p – za rozwiązanie zgodne ze specyfikacją
- 1p – za 1000 lub mniej wolnych pól,
- 1p – za 800 lub mniej wolnych pól, czas działania programu kilkanaście sekund
- $2 * 0.5$ – jeżeli Twój program wykorzystuje 1 lub 2 idee/algorytmy z wykładu (i potrafisz je wskazać)

Dodatkowo dostajesz $\max(0, (200 - W) / 100)$, gdzie W jest liczbą wolnych pól. Dla uzyskania tego bonusu możesz pozwolić programowi działać dłużej (ale na tyle krótko, byś mógł go uruchomić przy prowadzącym pracownię

Zadanie 3. (do 15p) Miniprojekt: Zaproponuj prowadzącemu trudne zadanie, które chciałbyś rozwiązać. Na stronie wykładu znajdziesz przykładowe propozycje, ale możesz też zgłosić inną

¹Na liście C5 będzie zadanie, z treścią: przeczytaj listę p5 i wyjaśnij ewentualnym zainteresowanym, o co chodzi w kolejnych zadaniach :)

2 Gry planszowe

Zadanie 4. 5p+1 Będziemy rozważać szachy błyskawiczne, w którym agenci co prawda wykonują algorytm minimaxowy (lub $\alpha\beta$ -search), ale po pierwszym ruchu rozważają jedynie ruchy *dynamiczne*, czyli bicia, szachy i obrony przed szachem (czyli kolejny ruch po szachu). Głębokość może być duża, bowiem zdecydowana większość gałęzi ma niewielki branching factor (często równy 1), a ponadto w naturalny sposób szybko się kończy (wyjątkiem są sekwencje szach-ucieczka-szach-ucieczka), które mogą potencjalnie trwać długo². W momencie, gdy dojdziemy do sytuacji spokojnej (bądź do granicy przeszukiwania), wówczas korzystamy z prostej funkcji heurystycznej, która jest sumą następujących składników

- waga moich bierek (pion ma wagę 1, waga pozostałych bierek jest parametrem agenta)
- waga bierek przeciwnika (ze znakiem przeciwnym)
- $\alpha \times$ liczba-moich-możliwych-ruchów
- $-\alpha \times$ liczba-możliwych-ruchów-przeciwnika

Dodatkowo, jeżeli ruch kończy się matem, to ma on wartość nieskończoną (czyli inne czynniki nie mają znaczenia).

Naszym celem będzie analizowanie partii rozgrywanych przez agentów. Przyjmujemy, że każda partia trwa co najwyżej K ruchów ($K=100?$). Agent wygrywa partię przez knock-out, jeżeli udało mu się zamatować przeciwnika, jeżeli nie – to oceniamy sytuację na planszy i przyznajemy zwycięstwo 'na punkty'.

Ten drugi przypadek ma 2 warianty (do wyboru przez studenta):

- a) Używamy 'standardowego' sposobu oceniania (czyli 1 dla piona, 3 dla skoczka, 3 dla gońca, 5 dla wieży, 9 dla hetmana).
- b) Używamy zewnętrznego, popularnego silnika szachowego, jak np. stockfish jako arbitra³. Ten wariant jest bardziej wiarygodny i daje dodatkowy punkt.

Twoim celem w tym zadaniu jest wylosować populację około 100 agentów (określanych przez wagi bierek i wartość α) i przeprowadzić pojedynki 'każdy z każdym'. Zliczyć zwycięstwa i porażki agentów i sporządzić raport do pliku, zawierający specyfikację i osiągi każdego agenta (oczywiście warto posortować agentów od najlepszego do najgorszego).

Zadanie 5. 2+X Cel i zasady są takie same jak w poprzednim zadaniu. Jediną różnicą jest wymaganie, żeby szukać optymalnego agenta za pomocą jakiejś heurystycznej metody przeszukiwania przestrzeni stanów (stan == specyfikacja agenta). Przy czym wartością agenta będzie średni wynik z 5-cioma najlepszymi agentami z poprzedniego zadania. Premia jest uznaniowo przyznawana ze względu na pracowitość i jakość rozwiązania.

Zadanie 6. 1-6p Wystartuj w turnieju z programem grającym w grę X (gdzie X to szachy, dżungla lub reversi). Szczegóły będą podane w osobnym dokumencie.

Zadanie 7. (8p) 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.

²Konieczne może być zatem jakieś zabezpieczenie się przed potencjalnym rozrostem drzewa związanym z wykonywaniem przez jedną stronę szachów

³Zwróć uwagę, że python-chess umożliwia odwołanie się do stockfisha, za pomocą `chess.engine.analyse`

W obu wariantach gry 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?) osiagającego dodatnią wartość oczekiwaną wypłaty w obu wariantach. Powinieneś przybliżyć 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 8. (6p) Rozważamy grę Connect 4⁴. 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 agenta⁵, 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).

3 O uczeniu

Zadanie 9. (7p) W zadaniu tym powinieneś opracować (być może korzystając ze innych źródeł wiedzy niż nasze wykłady) 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 wybijaniu „obcych” obrazków wśród cyfr.

Jeśli chodzi o obsługę danych MNIST możesz korzystać z dowolnej biblioteki i/lub kodu znanego 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 10. 8+X W tym zadaniu powinieneś napisać program analizujący ręcznie pisane cyfry, realizujący poniższy algorytm (którego szczegóły musisz samodzielnie dopracować, możesz też oczywiście zadać pytanie wykładowcy na temat tego zadania):

1. Stwórz zbiór kwadratowych fragmentów obrazów z części uczącej. Fragmenty powinny być niezbyt duże ($D \times D$, $D \approx 5$, ew. trochę większe). Powinny zawierać dość dużo pikseli cyfr (czyli nie być w całości wypełnione kolorem tła). Jak duży ten zbiór ma być, to jest Twoja decyzja.
2. Przeprowadź klasteryzację tego zbioru, dobierając samodzielnie liczbę klastrów K . Średnią obrazów w danym klastrze nazwiemy *wzorcem*.
3. Powiemy, że wzorec występuje w obrazie, jeżeli obraz zawiera fragment $D \times D$ wystarczająco podobny do wzorca (musisz samemu określić, co to znaczy)

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

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

4. Zauważ, że mamy w ten sposób metodę charakteryzowania obrazu cyfry za pomocą K bitów (bit i mówi, czy i -ty wzorzec jest znajduje się w obrazku). Ten ciąg bitów nazwiemy charakterystyką obrazu.
5. Dla każdej charakterystyki policz jakie obrazy (o jakich etykietach) ją mają. Wykorzystaj te statystyki do utworzenia klasyfikatora i sprawdź jego jakość.

Uwaga: $X \in \{0, 1, \dots, 7\}$ jest uznaniową premią związaną z jakością rozwiązania i ewentualnymi jego rozwinięciami.

4 Zadania o samochodzikach

Zadanie 11. (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 (v_x, v_y) , przy czym każda składowa prędkości należy do zbioru $\{-3, -2, -1, 0, 1, 2, 3\}$. Akcja jest parą (dv_x, dv_y) 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:

```
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 kontroluje 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 12. (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_<task-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 `car_model12.py`.

Zadanie 13. (od 10p do 15p) Rozważamy inny model autka (`car_model13.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 tarcie (składowa stała) oraz z oporami powietrza (zależnymi od prędkości i kwadratu prędkości)
- c) to, o ile skrećimy 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.