Sztuczna inteligencja
Ćwiczenia 4
Zajęcia w tygodniu 14-18 maja
(w kolejnym tygodniu dla grupy środowej)

Każde zadanie warte jest 1 punkt.

Zadanie 1. W grze Reversi pewne pola są *bezpieczne*, tzn. raz zajęte nie zmienią już w danej grze przynależności. Oczywistym przykładem są pola narożne. Jeżeli jednak o bezpieczeństwie pola będziemy mówić w kontekście konkretnej sytuacji na planszy, to pola narożne są potencjalnie niejedynymi bezpiecznymi: przykładowo, jeżeli czarny ma zajęty narożnik, to bezpieczne dla czarnego są dwa przylegające do tego narożnika pola brzegowe.

a) Jak wykorzystać wiedzę na temat bezpieczeństwa pola w tworzeniu heurystycznej funkcji oceniającej?

b) Zdefiniuj precyzyjnie warunek bezpieczeństwa pola (w sposób umożliwiający napisanie programu sprawdzającego bezpieczeństwo pola).

c) Zastanow sie, jak w praktyce najlepiej sprawdzac bezpieczenstwo pola, tak zeby funkcja heurystyczna nie zwolniła zbytnio. Czy warto definiowac jakąś podklasę pól bezpiecznych, które da się szybko wyznaczać?

Zadanie 2. Dla algorytmów Alpha-Beta-Search oraz MCTS odpowiedz na pytania:

a) W jaki sposób można wykorzystać czas poświęcony na obliczenia najlepszego *poprzedniego* ruchu do obliczenia najlepszego *bieżącego* ruchu (zakładamy, że rozgrywamy tylko jedną partię).

b) W jaki sposób wykorzystać możliwość równoległego wykonywania kodu w celu poprawy jakości gry?

Zadanie 3. Rozważamy grę „kółko i krzyżyk z buczeniem". Mamy zwykłą planszę do kółka i krzyżyka, dwóch graczy i serwer gry. Gracze zgłaszają propozycje ruchu do serwera gry (gracz A nie widzi propozycji gracza B i odwrotnie) i serwer albo przyjmuje ten ruch (jeżeli zaproponowane pole jest puste), albo buczy i gracz ma prawo do kolejnej próby (dowolnie wiele razy). Buczenie jest słyszalne dla obu graczy, niedozwolone jest powtarzanie jakiegoś ruchu (tzn. jak przy ruchu przeciwnika usłyszałem buczenie, to mogę założyć, że zdobył on dodatkową wiedzę, a nie że próbuje mnie zmylić). Oprócz tego to normalne kółko i krzyżyk (tzn. celem jest ustawienie 3 swoich znaków w wierszu, kolumnie lub na przekątnej).

Zaproponuj agenta, który grałby w tę grę (możliwie dobrze). Oczywiście pierwszy gracz ma dużą przewagę (dlaczego?), więc zakładamy, że rozgrywka składa się z $2K$ partii, w których gracze rozpoczynają na zmianę.

Zadanie 4. Gry karciane są dobrym przykładem sytuacji, w której nie tylko występuje element losowy (rozdawanie kart), ale również gracz musi podejmować decyzje w sytuacji, w której nie zna w pełni stanu gry (nie wie, co otrzymali inni gracze). Gdy programuje się takiego agenta, często używanym pomysłem jest wielokrotne losowanie możliwego stanu gry (czyli bieżącego układu kart), znajdywanie najlepszego ruchu w tym stanie (używając standardowych metod rozwiązywania gier z jawnym stanem) i ostateczny wybór ruchu, który był najlepszy w największej liczbie losowań.

Wyjaśnij następujące kwestie:

a) Co może oznaczać: losowanie **możliwego** stanu?

b) W jakich sytuacjach losowanie z poprzedniego punktu chcielibyśmy wykonywać przypisując stanom niejednakowe prawdopodobieństwa? Co można w ten sposób uzyskać i jakie to rodzi problemy? Uwaga dla osób mało grających w gry karciane: wiele gier ma element licytacji, w której gracze deklarują (często wielokrotnie) jaki wariant przyszłej rozgrywki wydaje im się odpowiedni.

c) Jaki istotny aspekt gier karcianych jest pomijany w tym podejściu?

Zadanie 5. Rozważmy grę oszust (zob. Cheat_(game) w Wikipedii). Gramy standardową talią 52 kart (asy są najniższymi kartami). Mamy k graczy, którym rozdane są wszystkie karty. Zaczyna rozdający, po czym gracze na zmianę „zagrywają" od 1 do 4 kart (przy czym zagrywają je koszulkami do góry, tak że nie są widoczne dla innych graczy). Celem jest pozbycie się wszystkich kart. Gracz zagrywając

kartę (karty) mówi o ich wartości (na przykład mówi: zagrywam dwie siódemki). Można deklarować jedynie karty o równej wielkości, dodatkowo deklaracja powinna być starsza (bądź równa) deklaracji poprzedniego gracza, czyli na „dwie siódemki" można rzucić „trzy dziewiątki", ale nie odwrotnie. Gracze mogą kłamać odnośnie tego, co rzucili. Po każdej zagrywce jest faza sprawdzania, w której kolejni gracze mogą spasować lub sprawdzić zagrywającego (ta faza albo zawiera $k-1$ pasów, albo pewną liczbę pasów i pierwsze sprawdzenie). Jeżeli zagrywający kłamał, to zabiera wszystkie karty[1]. Jeżeli zagrywający powiedział prawdę, to karty bierze sprawdzający. W obu przypadkach kolejny gracz kontynuuje rozgrywkę (ponieważ stos jest pusty, może wyrzucić karty o dowolnej wysokości, oczywiście wyrzucając karty może skłamać o ich wartości).

Zaproponuj kilku (co najmniej 2) przykładowych prostych agentów, grających w tą grę.

Zadanie 6. Potestuj playground.tensorflow.org. Odpowiedz na pytania:

a) Dla jakich zbiorów danych i jakich cech wystarcza 1 neuron do poprawnej klasyfikacji? (i dlaczego)

b) Co dzieje się, gdy dla bardziej złożonych sieci damy zbyt duży *Learning rate*?

c) W którym zadaniu przydają się cechy sin i cos?

d) Dla każdego zbioru danych (oprócz spirali) powiedz, jaka najprostsza[2] sieć neuronowa korzystająca tylko z cech x_1 i x_2 poprawnie klasyfikuje ten zbiór danych.

Zadanie 7. Rozważamy sieć neuronową z prostą funkcją schodkową (równą 1 dla liczb dodatnich, 0 w przeciwnym przypadku). Wejściem do tej sieci będą zera i jedynki, zatem sieć będzie obliczała jakąś funkcję boolowską.

a) Podaj sieci neuronowe (złożone z jednego neurona) obliczające $x \vee y$, $x \wedge y$, $\neg x$.

b) Podaj sieć neuronową dla x **xor** y

c) Uzasadnij, że nie jest możliwa sieć z punktu b), która ma tylko 1 neuron

Zadanie 8. Rozważamy takie same sieci, jak w poprzednim zadaniu. Uzasadnij, że za pomocą sieci neuronowej możemy wyrazić dowolną funkcję boolowską:

a) starając się, by sieć miała możliwie mało neuronów;

b) starając się, by sieć miała możliwie mało warstw neuronów.

Zadanie 9. Przypominamy, że $\mathrm{ReLU}(x) = \max(0, x)$. Podaj sieci neuronowe, biorące jako wejście liczby od -1 do 1, z funckją aktywacji równą ReLU, obliczające:

a) **if** $x > T$ **then** a **else** c

b) $\max(a, b)$

c) $a + b$

Zadanie 10. Niech $f : [0,1] \times [0,1] \to [0,1]$. Przyjmijmy, że f jest ciągła. Naszkicuj dowód, że za pomocą sieci neuronowej z funkcją schodkową[3] jako funkcją aktywacji możemy przybliżać f z dowolną dokładnością[4]. Wskazówka (rot13.com): pml hzvrfm fgjbemlć fvrć, xgóen zn fgnłą jnegbść an xjnqenpvr? Ile warstw ma Twoja sieć?

Zadanie 11. Czy powyższy wynik daje się uogólnić na więcej wymiarów (tzn. na funkcję określoną na $[0,1]^k$)? Czy ma on praktyczne znaczenie?

[1] Zasady gry tego nie prezycują, ale możemy przyjąć, że gracz zabiera karty zachowując ich kolejność i jeżeli ma odpowiednio dobrą pamięć, to może po fakcie sprawdzić, kto kłamał, a kto mówił prawdę w podczas budowy tego stosu.

[2] Mająca najmniej warstw i (w drugiej kolejności) najmniej neuronów.

[3] równą 0 dla argumentów nieujemych, 1 dla dodatnich

[4] Czyli, że dla zadanej f i zadanego ε możemy stworzyć sieć, której wartości będą różniły się od f o co najwyżej ε