Raport
Wprowadzenie do Sztucznej Inteligencji – Ćwiczenie 6.

Uczenie ze wzmocnieniem

Autor: Aleksandra Jamróz, nr albumu: 310 708

Treść zadania – *Q-Uber*

Elon Piżmo konstruuje autonomiczne samochody do swojego najnowszego biznesu. Dysponujemy planszą NxN (domyślnie N=8) reprezentującą pole do testów jazdy. Na planszy jako przeszkody stoją jego bezpłatni stażyści. Mamy dwa autonomiczne samochody: Random-car, który kierunek wybiera rzucając kością (błądzi losowo po planszy) oraz Q-uber, który uczy się przechodzić ten labirynt (używa naszego algorytmu). Samochody zaczynają w tym samym zadanym punkcie planszy i wygrywają, jeśli dotrą do punktu końcowego, którym jest inny punkt planszy. Istnieje co najmniej jedna ścieżka do startu do końca. Elon oszczędzał na module do liczenia pierwiastków, dlatego samochody poruszają się przy użyciu metryki Manhattan (góra, dół, lewo, prawo). Jeżeli samochód natrafi na stażystę to kończy bieg i przegrywa. Analogicznie jak wejdzie na punkt końca to wygrywa i również nie kontynuuje dalej swojej trasy. Celem agenta jest minimalizacja pokonywanej trasy.

Labirynt

Plansza, po której poruszają się samochody, to lista zawierająca N list o długości N, przy czym N to długość boku planszy. Wszystkie elementy inicjowane są wartością 0, oznaczającą pole, po którym samochody mogą się dowolnie poruszać. Następnie losowane są miejsca dla stażystów, które oznacza się poprzez zamianę zera na jedynkę. Poprawność wygenerowanego labiryntu jest sprawdzana za pomocą funkcji *dfs*, która znajduje pola na planszy, do których można dotrzeć z punktu początkowego. Jeżeli na liście odwiedzonych punktów znajduje się punkt końcowy, labirynt jest poprawny. Aby ułatwić przeprowadzanie testów, zaimplementowałam funkcje zapisujące i odczytujące labirynt do pliku tekstowego.

```
Wygenerowany labirynt:
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 1, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 1, 1, 1, 0]
[0, 1, 0, 1, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0]
```

Rysunek 1. Przykładowy wygenerowany labirynt oznaczony zerami i jedynkami

Taksówka Elona

Taksówka zawiera planszę do przejechania oraz współrzędne punktu początkowego i końcowego. Są one ustawione domyślnie na przeciwległe krańce planszy. Oprócz tego zawiera pole określające, czy taksówka zakończyła już swój bieg – dzieje się to w przypadku uderzenia w stażystę lub dojechania do punktu końcowego. Główna metoda tej klasy polega na wykonaniu ruchu. Mamy do wyboru 4 akcje: ruch w dół – 0, w górę – 1, w lewo – 2 oraz w prawo – 3. Podczas ruchu zmienia się jedna z aktualnych współrzędnych taksówki. Następnie sprawdzana jest poprawność ruchu i na tej podstawie zwracana jest nagroda. Jeżeli taksówka uderza w ścianę, taksówka nie kończy biegu, ale ustawiane są poprzednie współrzędne, a nagroda wynosi -10. Jeżeli uderza w stażystę – kończy jazdę a nagroda to również -10. Nagroda za dojechanie do miejsca docelowego wynosi 20, a za bezwypadkowy ruch: -1. Nagroda wynosząca -1 za teorytycznie poprawny ruch wynika z minimalizacji liczby wszystkich ruchów wykonanych na planszy.

Q-learning

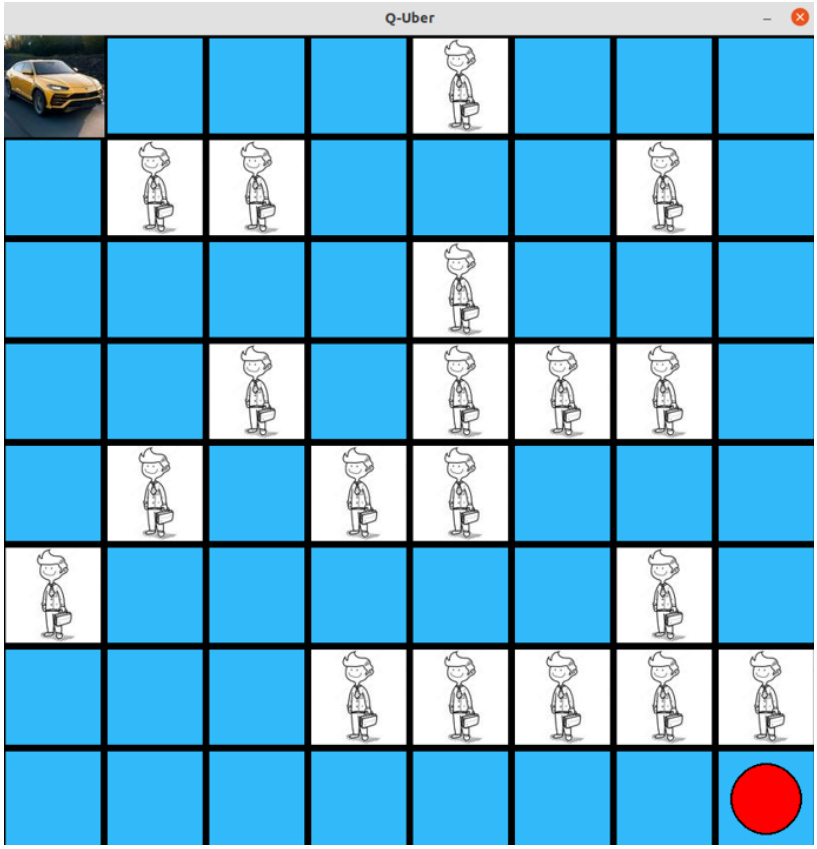
Liczba stanów taksówki jest równa liczbie pól na planszy. Dla wszystkich możliwych kombinacji współrzędnych numer stanu liczy się poprzez pomnożenie wartości współrzędnej x przez liczbę rzędów i dodanie do otrzymanej wartości współrzędnej y. W ten sposób każdy stan otrzymuje unikalny numer. Główny zamysł uczenia ze wzmocnieniem ma miejsce w metodzie *drive*. Odpowiada ona za wykonanie całej trasy taksówki od punktu początkowego do wydarzenia kończącego jazdę. Jazda odbywa się wedle podanych parametrów: epsilon, beta i gamma oraz podanej tablicy q-learningu. Tablicę podaję jako parametr, ponieważ do wykształcenia poprawnej tabeli potrzeba wielu taksówek. Metoda zwraca listę wykonanych po kolei akcji oraz zmodyfikowaną tablicę. Epsilon wyznacza prawdopodobieństwo wykonania ruchu losowego lub ruchu zgodnego z odczytem tabeli. Jeżeli wlosowana wartość z przedziału (0,1) jest od niego mniejsza, numer akcji jest przypadkowy. W innym przypadku wybiera maksymalną wartość z odpowiedniego rzędu w tabeli. Następnie wykonywany jest ruch oraz wyznaczana jest nagroda. Za pomocą funkcji *update_q_table* modyfikuję pole poprzedniego stanu i wykonanej akcji w tabeli. Wzór tej operacji wygląda następująco:
*q_table[state, action] += beta * (reward + gamma * next_state_max - q_table[state, action])*

Random taxi

Samochód poruszający się po labiryncie w losowy sposób ma nikłe szanse na jego przejście, zwłaszcza zaczynając za każdym razem od zera, to znaczy nie posiadając żadnej wiedzy na temat labiryntu. Aby przejść labirynt, którego używałam do testów taksówki Elona, musiałby wykonać bezbłędnie ponad 15 ruchów pod rząd. Prawdopodobieństwo takiego zdarzenia wynosi ok. 9.31e-10, czyli jest mniejsze niż szansa na trafienie szóstki na loterii lotto. Z tego powodu zdecydowałam, że samochód losowy również będzie się uczył jazdy po labiryncie. Podobnie do inteligentnej taksówki posiada tabelę do jazdy, która dla każdego stanu na początku przechowuje 4 możliwe ruchy. Jeżeli samochód po wykonaniu akcji wjedzie w stażystę bądź wyjedzie poza obszar labiryntu, akcja ta zostaje usunięta z możliwych akcji dla danego stanu. W wyniku serii takich operacji taksówka ma do dyspozycji tylko poprawne ruchy, ale ich wybór jest za każdym razem losowy. Po jeździe zakończonej sukcesem do listy wykonanych akcji dodawana jest na końcu akcja nr „4”, ułatwiająca potem testowanie taksówek.

Wizualizacja

Wykorzystałam bibliotekę pygame do wizualizacji trasy pokonywanej przez samochód. Żeby włączyć odpowiednią trasę, należy ręcznie wkleić uzyskaną liczbę ruchów lub zastosować listę zwróconą bezpośrednio przez funkcję *drive* taksówki, po czym uruchomić program.
W przykładowym kodzie zamieściłam listy *interval_steps*, które po odkomentowaniu zbierają kolejne listy kroków w określonych odstępach. W ten sposób możemy prześledzić generowane kroki i zaobserwować proces nauki algorytmu.



Rysunek 2. Plansza przed wyruszeniem taksówki z zaznaczonym punktem końcowym

Testowanie algorytmu

Testy dla Q-Ubera różniły się od testów przeprowadzanych dla taksówki typu Random Car. Założyłam, że w przypadku Q-Ubera interesować mnie będzie znalezienie ścieżki prowadzącej do końca labiryntu, a w przypadku istnienia kilku ścieżek - najkrótsza z nich. Testy losowej taksówki polegały praktycznie na wyeliminowaniu błędnych ruchów, ponieważ od tego momentu każda ścieżka była poprawna, różniąc się jedynie długością.

Jakość działania algorytmu q-learningu zależała od parametrów beta, gamma oraz epsilon. Zdecydowałam przeprowadzić serię prób dla różnych wartości tych parametrów. Wynikiem jednej próby była liczba iteracji potrzebnych do ukształtowania wystarczająco poprawnej tablicy, żeby na jej podstawie taksówka przejechała do punktu docelowego najkrótszą trasą. Wyniki w największym stopniu zależały od wartości epsilon. Im był większy, tym gorsze były osiągi algorytmu. Oznacza to, że program działa lepiej, korzystając z kształtowanej tabeli, niż wybierając ruchy losowo. Na drugim miejscu pod względem wagi plasuje się współczynnik

beta. W jego przypadku osiągi poprawiały się proporcjonalnie do jego wzrostu, ponieważ algorytm mógłuczyć się szybciej. Najmniejszy wspływ na wynik miała wartość dyskonta. Nie zaoserwowałam przewagi żadnej z wartości, które testowałam. Dodawane liczby nie różniły się znacząco, biorąc również ich skalę, co przełożyło się na taki wynik testów.
Poniższe tabele są posortowane rosnąco względem uzyskanej liczby iteracji.

epsilon	gamma	beta	iterations
0.05	0.6	0.5	119
0.05	0.8	0.5	124
0.05	0.4	0.5	135
0.05	0.6	0.3	169
0.05	0.8	0.3	170
0.05	0.4	0.3	195
0.05	0.6	0.1	352
0.05	0.4	0.1	458
0.05	0.8	0.1	462

Rysunek 3. Liczba iteracji dla epsilona = 0,05

epsilon	gamma	beta	iterations
0.15	0.8	0.5	195
0.15	0.6	0.5	243
0.15	0.4	0.5	254
0.15	0.6	0.3	342
0.15	0.8	0.3	385
0.15	0.4	0.3	444
0.15	0.8	0.1	684
0.15	0.6	0.1	798
0.15	0.4	0.1	1092

Rysunek 4. Liczba iteracji dla epsilona = 0,15

epsilon	gamma	beta	iterations
0.3	0.8	0.3	665
0.3	0.8	0.5	740
0.3	0.6	0.3	1003
0.3	0.4	0.5	1060
0.3	0.4	0.3	1344
0.3	0.6	0.5	1789
0.3	0.6	0.1	2112
0.3	0.8	0.1	2114
0.3	0.4	0.1	2228

Rysunek 5. Liczba iteracji dla epsilona = 0,3

Taksówka poruszająca się losowo po planszy, ucząc się błędnych ruchów zgodnie z wcześniejszym opisem, potrzebowała średnio 49 prób, żeby znaleźć pierwszą poprawną trasę. Następnie dawałam algorytmowi pewną liczbę prób na znalezienie jak najkrótszej drogi. Ani razu nie udało mi się osiągnąć w ten sposób najkrótszej możliwej drogi. Najkrótsza wygenerowana droga była 2 razy dłuższa od oczekiwanej i była wynikiem 10000 przeprowadzonych prób. Jest to sposób zdecydowanie nieopłacalny biorąc pod uwagę, że przy dobrych ustawieniach parametrów, najkrótszą drogę możemy uzyskać już po średnio 115 iteracjach.