WSI - ćwiczenie 6

Uczenie (się) ze wzmocnieniem

Levchenko Sofiia, nr. indeksu 308996

27.05.2002

Wstęp

Celem tego ćwiczenia było zaimplementowanie algorytmu Q-learning, a następnie wykorzystać go w celu stworzenia agenta, rozwiązującego problem taxi.

Struktura projektu

Ze względu na specyfikę wykonywanego zadania, zdecydowałyśmy się, że struktura projektu będzie wyglądała w następujący sposób:

- *q_learning.py* – implementacja algorytmu q_learning rozwiązującego problem taxi, możliwe do wywołania są trzy funkcje: pierwsza do trenowania modelu (uzupełnienie tablicy nagród), druga jest identyczna do pierwszej, za wyjątkiem tego, że też ona zbiera statystyki dotyczące kosztu, a trzecia wykorzystywana, aby wykorzystać wytworzony model do przejścia przez rozwiązywany problem
- *experiments.py* - funkcje z definicjami eksperymentów przeprowadzonych w trakcie zadania, w tym też funkcje pomocnicze do wizualizacji wyników
- *main.py* - zawiera wywołania eksperymentów przeprowadzonych w trakcie realizacji zadania

W ramach projektu były wykorzystywane różne biblioteki zewnętrzne, ułatwiające wykonanie zadania:

- *numpy* - do działań na różnych strukturach danych oraz przeprowadzenia na nich obliczeń/działań matematycznych.
- *pandas* - do przetwarzania danych w celu wizualizacji wyników
- *matplotlib* - biblioteka ułatwiająca tworzenie wykresów
- *gym* – biblioteka do zasymulowania rozwiązanego problemu taxi

Decyzje projektowe

Uczenie modelu polega na przejściu przez rozpatrywany problem (w naszym przypadku, problem taxi), który da się opisać przez jego stany, akcje i nagrody (jak w modelu decyzyjnym Markowa), wiele razy i na podstawie uzyskiwanych nagród, zmieniać zawartość tabeli nagród w taki sposób, aby reprezentowała ona wszystkie możliwe stany i akcje, możliwe do wykonania w danych symulowanym środowisku.

Algorytm Q-learning został dostosowany do rozpatrywanego problemu i zawiera w sobie zarówno jak inicjalizację środowiska (i dokonanie ruchów do przemieszczania się po nim), tak i sam wzór do aktualizowania wartości w tabeli nagród (oparty na równaniu Bellmana).

Funkcja *q_learning_train* potrzebuje do określenia takich hiperparametrów, jak:

- *observation*_space, do określenia wymiaru przestrzeni możliwych stanów środowiska
- *action_space*, do określenia wymiaru przestrzeni możliwych do dokonania akcji

- *iterations,* do określenia liczby iteracji, podczas których są zmieniane wagi tabeli kosztów
- *learning_rate,* do określenia tego, jak szybko algorytm uczy się na podstawie nagród powiązanych ze stanami (wartość tzw. kroku)
- *dyskonto,* do opisu tego, jak ważna jest natychmiastowa gratyfikacja w porównaniu z długoterminowymi korzyściami
- *random_action_prob*, do określenia wartości progu, kiedy należy wykonać losową czynność zamiast akcji z tabeli nagród

Opis wykonanych eksperymentów

Aby przetestować jakość tabeli nagród, wytworzonej przy pomocy algorytmu q_learning, została ona wykorzystana do przejścia przez problem i obliczenia, jaki był koszt takiego jednorazowego kosztu i na podstawie tego, zostało ocenione, czy dokonane kroki były optymalne.

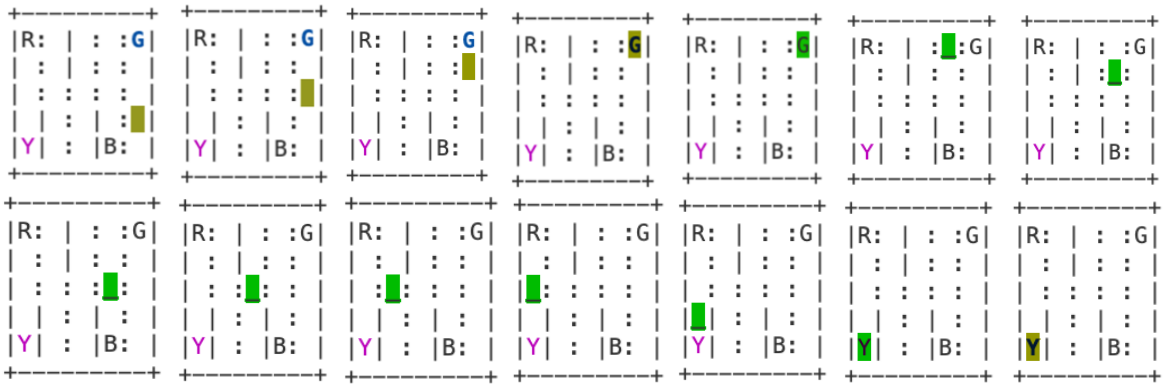
Następnie został zbadany wpływ wartości hiperparametrów na to, jaki był uzyskiwany koszt przejścia przez rozpatrywany problem. Badany był wpływ parametru learning_rate, dyskonto oraz random_action_prob.

Pod koniec, algorytm został uruchomiony 50 razy, aby przetestować, jak bardzo czynnik losowy wpływa na jakość uzyskiwanych wyników.

Wyniki

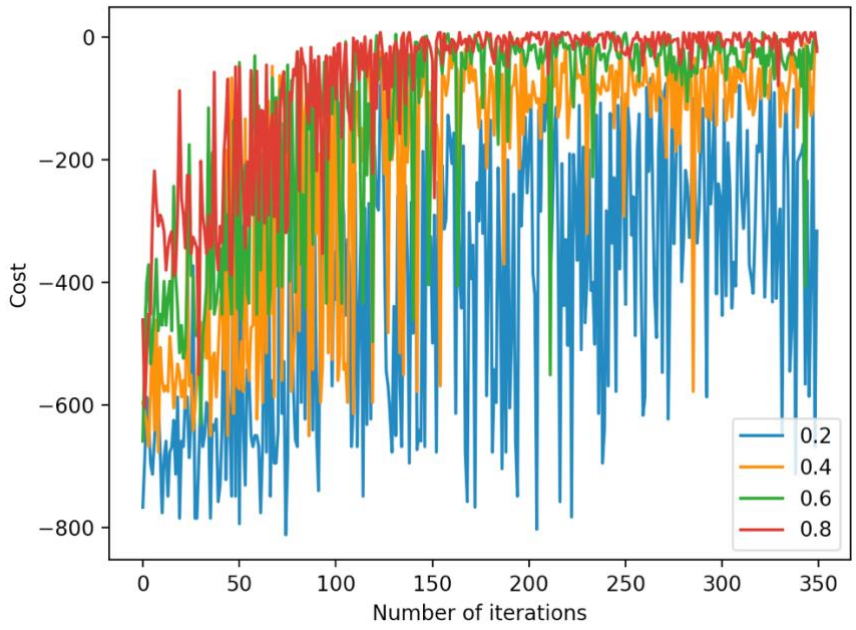
Możemy zobaczyć, że rozpatrywana taksów dokonywała wyłącznie optymalnych kroków, aby odebrać pasażera z punktu G i dostarczyć go do punktu Y.

Koszt takiego przejścia jest równy 8, co jest bardzo dobrym wynikiem dla problemu tego typu.



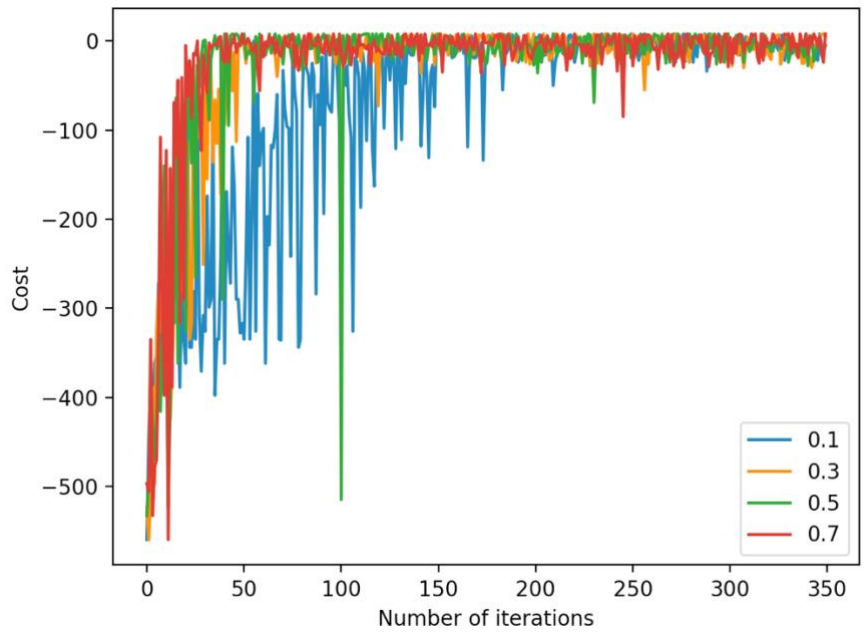
Następnie został zbadany wpływ prawdopodobieństwa dokonania losowej akcji (współczynnika eksploracji) na uzyskiwane wartości kosztu przejścia przez problem.

Możemy zobaczyć, że dla zbyt małych wartości model praktycznie się nie uczy i rozrzut uzyskiwanych wartości jest na dość wysokim poziomie.

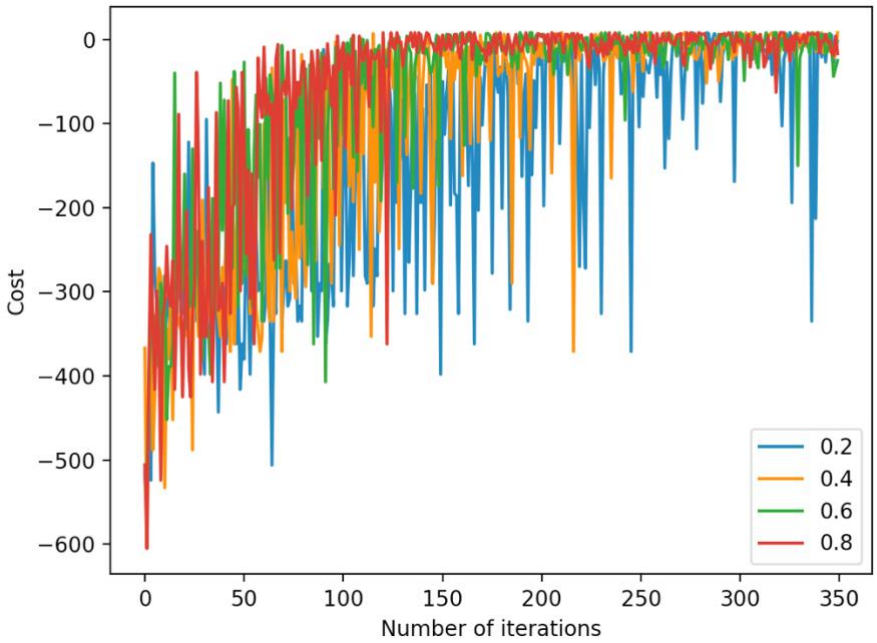


Po tym został zbadany wpływ współczynnika uczenia dla podobnej sytuacji.

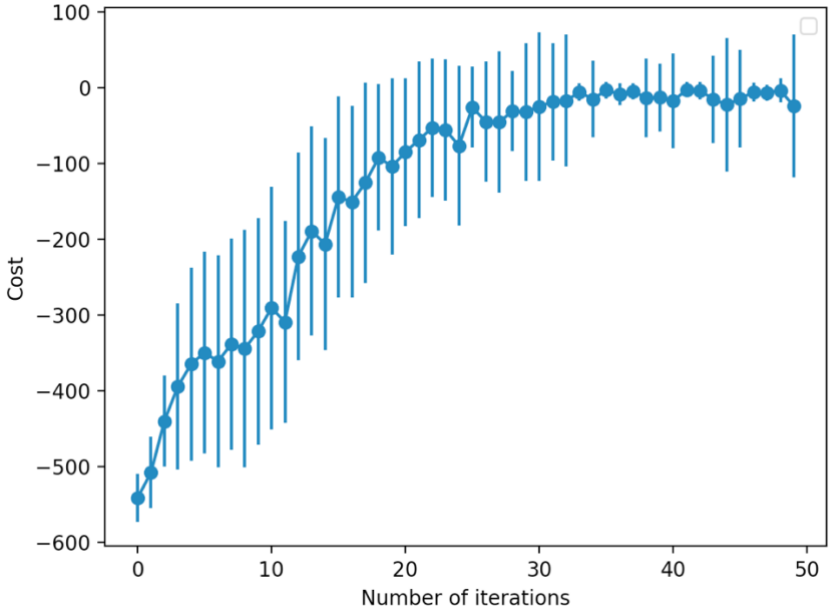
W tym przypadku akurat możemy zauważyć, że wyniki dla bardzo małych wartości polepszają się wolniej, niż dla wartości większych (co się dzieje bardzo szybko), natomiast dla dużych wartości parametru określającego liczbę iteracji, wszystkie uzyskiwane wartości kosztu są równie dobre.



Ostatnim wśród badanych hiperparametrów był współczynnik dyskonto. Tutaj możemy zaobserwować sytuację podobną do sytuacji ze współczynnikiem uczenia, jednak z taką różnicą, że dla zbyt małych wartości, nawet dla dużych liczby iteracji rozrzut uzyskiwanych wyników jest dość duży.



Też możemy zobaczyć, że tendencja polepszania się kosztu wraz ze wzrostem liczby iteracji została zachowana i wyniki dla najlepszych współczynników (wyznaczonych na podstawie przeprowadzonych eksperymentów) bardzo szybko osiągają wartości bliskie 0 (co jest dobrym wynikiem).



Wnioski

Zaimplementowany algorytm Q-learning okazał się bardzo pomocnym w przypadku rozwiązywanego problemu. Uzyskiwane wyniki są bardzo dobre, akcje do wykonania dobierane są w optymalny sposób.

W przypadku rozpatrywanego problemu, najlepsze wyniki uzyskiwane dla wartości parametrów bliskich 0.8, jednak te wartości mogą się różnić dla różnych symulowanych środowisk.

W danym przypadku wartości uzyskiwane były bardzo dobre, dlatego że w danej symulacji poprzez stan uzyskiwaliśmy informację o wyglądzie (sytuacji) całej mapy, a nie jej kawałku.