Arkadiusz Kojtek 310746

WSI

Q-Learning

Uruchamianie programu

Przed uruchomieniem programu konieczne zainstalowanie następujących modułów.
Numpy - do tworzenia macierzy zawierającej ocenę jakości stanów Q-Table:
pip install numpy
tqdm umożliwiający wizualizację przewidywanego trenowania agenta
pip install tqdm
gym - umożliwia stworzenie środowiska, dla którego pracować będzie nasz agent. W projekcie korzystam ze środowiska **Taxi-v3**
pip install gym

Implementacja algorytmu

Implementacja składa się z klasy **Algorithm** będącej implementacją algorytmu Q-Learn. Jako argumenty konstruktor przyjmuje środowisko dla którego będzie pracował algorytm. W naszym wypadku środowiskiem tym jest **Taxi-v3**.
Klasa algorytm składa się z następujących metod:
greedy_epsilon_strategy() - metoda przyjmuje jako argumenty aktualną wartość epsilona oraz stan w którym aktualnie znajduje się agent. Wybiera on numer następnej akcji na podstawie polityki greedy-epsilon.
boltzmann_strategy() - przyjmuje jako argumenty tau oraz stan, w którym aktualnie znajduje się agent. Korzystając z polityki boltzmanna tworzy tablice prawdopodobieństwa z której następnie wybierany jest numer następnej akcji.
training() - metoda przyjmuje jako argumenty hiper parametry alpha, gamma, epsilon, liczbę epok, stopień spadku epsilona oraz numer wybranej strategii.
Przeprowadza ona trening agenta zmieniając wartości w tablicy Q-Table gdzie wybór akcji podczas treningu będzie określany na podstawie wybranej polityki.
evaluate_trening() - metoda ta przyjmuje jako argumenty ilość testów oraz maksymalną ilość kroków agenta podczas testu. Metoda ta dla każdego epizodu zbiera ilość kroków oraz błędów i na ich podstawie określa średnie wartości na przestrzeni wszystkich epizodów.

Testy numeryczne

Podstawowe założenia:
1. Testować będę wpływ alphy, gammy, epsilona oraz ilości epizodów podczas treningu na skuteczność działania każdej ze strategii.
2. Maksymalną ilość kroków podczas ewaluacji treningu ustawiłem na 10000. Jest to bardzo duża wartość jednak moim celem będzie uzyskanie jak największej skuteczności działania treningu przy jak najmniejszej ilości epizodów. Nie będę więc zwracał uwagi na ilość kroków potrzebną agentowi na dotarcie do celu.
3. Na każdym uzyskanym Q-Table przeprowadzone zostanie 100 losowych testów sprawdzających skuteczność treningu.

Tabela wyników

Wartości w tabeli nie odzwierciedlają dokładnego działania algorytmu. Testy przeprowadzałem przez wielokrotne uruchomienie treningu na tych samych parametrach przez co miałem dokładniejszy obraz że wynik treningu nie zawsze był skuteczny. Dlatego też zero błędów nie oznacza, że agent zawsze był w stanie odnaleźć ścieżkę. Ilość wierszy tabeli, która by musiała znaleźć się przy faktycznym zaprezentowaniu moich testów byłaby dwudziestokrotnie większa niż aktualnie. Zdecydowałem się więc tylko na umieszczenie pojedynczych testów w sprawozdaniu. Jednak faktyczne wyniki omówię we wnioskach.

Numer w kolumnie Polityka określa jaka strategia była zastosowana do określenia następnej akcji. Liczba 1 odpowiada polityce greedy-epsilon natomiast 2 strategii Boltzmanna. Kolumna *liczba niepowodzeń* określa ilu agentów nie dotarło do celu podczas przeprowadzania testów.

Testy rozpocząłem od strategii zachłannej-epsilon jako parametry podając losowe wartości, na podstawie których planowałem dążyć do uzyskania jak najmniejszej liczby epizodów.

Jako początkowe wartości ustaliłem
Alpha - 0.1
Gamma - 0.6
Epsilon/Tau - 0.1
Ilość epizodów 5000
Rozpad Tau/Epsilon - 0.001

Niestety dla zadanych parametrów wyniki były bardzo niedokładne, zacząłem więc zwiększać ilość epizodów aż do 10000 dla których uzyskiwałem dokładniejsze ale wciąż nieprecyzyjne wyniki. Dla 10000 epizodów 7 agentów na 100 prób nie było w stanie dotrzeć do celu.

Alpha	Gamma	Epsilon/Tau	Ilość epizodów	Rozpad Tau/Epsilon	Polityka	Liczba niepowodzeń	Liczba kroków na test	Liczba błędów na test
0.1	0.6	0.1	5000	0.001	1	36	3607.8	0.0
0.1	0.6	0.1	8000	0.001	1	15	1511.09	0.0
0.1	0.6	0.1	10000	0.001	1	7	711.68	0.0
0.2	0.6	0.1	8000	0.001	1	0	13.4	0.0
0.3	0.6	0.1	8000	0.001	1	0	13.37	0.0
0.5	0.6	0.1	7000	0.001	1	0	12.85	0.0
0.5	0.4	0.1	5000	0.001	1	0	13.1	0.0
0.5	1	0.1	5000	0.001	1	0	12.97	0.0
0.5	1	0.1	3000	0.001	1	0	12.73	0.0
0.5	1	0.5	3000	0.001	1	0	13.13	0.0
0.5	1	0.5	1000	0.001	1	2	212.7	0.0
0.5	1	0.3	1000	0.001	1	0	13.05	0.0
0.5	1	0.1	1000	0.001	1	1	112.61	0.0
0.1	0.6	0.1	10000	0.001	2	0	13.7	0.0
0.1	0.6	0.1	8000	0.001	2	0	12.93	0.0
0.1	0.6	0.1	5000	0.001	2	0	12.97	0.0
0.1	0.6	0.1	3000	0.001	2	1	112.59	0.0
0.1	0.6	0.1	5000	0.001	2	1	112.43	0.0
0.5	0.6	0.1	2000	0.001	2	0	12.89	0.0
0.5	0.6	0.1	1500	0.001	2	0	13.51	0.0
0.5	0.6	0.1	1000	0.001	2	0	12.58	0.0
0.5	1	0.1	1500	0.001	2	0	13.28	0.0
0.5	0.2	0.1	1500	0.001	2	0	13.14	0.0
0.5	0.8	0.1	1500	0.001	2	1	113.11	0.0
0.5	0.6	1	1500	0.001	2	0	13.24	0.0
0.5	0.6	1	1000	0.001	2	0	13.33	0.0
0.5	0.6	1	800	0.001	2	0	13.05	0.0
0.5	0.6	1	500	0.001	2	0	12.99	0.0
0.5	0.6	3	500	0.001	2	0	12.71	0.0
0.5	0.6	3	400	0.001	2	0	13.22	0.0
0.5	0.6	3	300	0.001	2	0	13.41	0.0

Jednak wyniki był dostatecznie zadowalający by zacząć próbować wprowadzać zmiany w wartościach pozostałych hiper parametrów. Zacząłem więc od zmian współczynnika uczenia - alpha. Jego zwiększenie wpłynęło pozytywnie na skuteczność treningu. Zwiększenie alphy do 0.5 z startowego 0.1 pozwoliło na zmniejszenie ilości epizodów do tylko 7000 przy jednoczesnej 100% skuteczności.

Następnie zacząłem edytować wartość gammy. Jej zmniejszenie i nastawienie się na krótkoterminowe zyski nie spowodowało żadnych pozytywnych zmian.
Zwiększenie gammy oraz nastawienie się na długoterminowe zyski natomiast wyraźnie poprawiło skuteczność programu, który zachowywał 100% skuteczność już tylko przy 3000 epizodów.
Jako ostatnią zacząłem edytować wartość parametru epsilon, niestety tutaj nie udało się uzyskać dodatkowej poprawy działania kodu. Testy dla mniejszej ilości epizodów napotykały na błędy w niektórych testach.

Wychodząc z podobnych startowych parametrów zacząłem następnie pracę ze strategią Boltzmannowską. Od razu mogę zauważyć że program korzystając z tej strategii działał wyraźnie wolniej niż poprzednio, dodatkowe obliczenia wymagane przy tworzeniu tabeli prawdopodobieństw wyraźnie wpłynęły na szybkość programu.

Jednak tutaj warto zwrócić uwagę na skuteczność działania programu, dla tych samych parametrów algorytm zachowywał 100% skuteczność w odnajdywaniu najlepszej ścieżki dla tylko 5000 epizodów co było wyraźnie lepszym wynikiem niż przy strategii greedy-epsilon.

Jednak i tutaj zacząłem szukać lepszych parametrów. Zacząłem ponownie od edycji alphy od razu ustawiając ją na wartość 0.5, która dała wyraźną poprawę przy poprzednim teście. Teraz było podobnie, dla tej wartości udało się uzyskać 100% skuteczność treningu już przy tylko 1500 epizodach!

Jako kolejny zacząłem edytować parametr gamma, jednak tutaj zarówno zmniejszanie jak i zwiększanie jej wartości nie dawało żadnej poprawy, często wręcz przeciwnie - psując skuteczność treningu. Zachowałem więc jej wartość jako 0.6 i zacząłem edytować wartość Tau wykorzystywaną w strategii Boltzmanna do określania stopnia jak bardzo prawdopodobna jest każda akcja. Przez jej zwiększanie wyrównywałem prawdopodobieństwo wystąpienia każdej z akcji. Jej zwiększanie wpłynęło korzystnie na skuteczność programu. Zwiększenie tau do 3 umożliwiło uzyskanie 100% skuteczności przy tylko 800 epizodach.

Kolejne testy przeprowadziłem dla tych samych parametrów co wcześniej jednak z dalej zmniejszoną ilością epizodów. Program wciąż zachowywał bardzo wysoką skuteczność nawet dla 500, 400 czy 300 epizodów co jest bardzo dobrym wynikiem zważając na ogólną ilość stanów w środowisku Taxi-v3 których jest 500.

Wnioski

Istnieje bardzo wyraźna różnica w skuteczności obu strategii. Strategia Boltzmanna traci na szybkości znajdowania samej akcji przez dużą ilość obliczeń jakie trzeba przeprowadzić. Jednak jest o wiele bardziej skuteczna niż strategia greedy-epsilon, przez co braki w szybkości obliczeń nadrabia bardzo małą potrzebną ilość epizodów treningu, których udało mi się uzyskać ponad razy mniej niż w wypadku greedy-epsilon.