Uczenie się ze wzmocnieniem– wsi lab 6

Autor: Patryk Zdziech

Nr.a 311028

Środowisko: Python

Użyte biblioteki: numpy, gym, random, pyplot

Implementacja

Klasa q_learn zawierającą:

Zmienne:

`env`- wybrane środowisko z biblioteki „gym” (taxi, ew. frozen lake)
`q_table`- tablica przewidywanych nagród za akcje zależnie od stanu. Ma wymiary „m” X „n”, gdzie „m” odpowiada wielkości przestrzeni stanów a „n” wielkości przestrzeń akcji.
`alpha` *(learning rate)*, `gamma` *(discount)* – parametry aktualiacji wartości w tablicy `q_table`
`espsilon` - wartość używana do strategii „epsilon zachłannej”, określająca szanse na losowe posunięcie oraz do strategii „Boltzmanna” jako temperatura
`self.state` - obecny stan środowiska

Funkcje:

- `__init__` – na wejściu przyjmuje parametry `alpha, gamma, epsilon` oraz opcjonalnie nazwę środowiska z biblioteki „gym”, jeśli chcemy użyć innego niż „Taxi-v3”.
- `epsilon_greedy, boltzmann` – polityki wyboru kolejnej akcji
- `train` – trenowanie dla zadanej liczby epizodów. Opcjonalna flaga określa czy będzie użyta polityka Boltzmanna, czy epsilon zachłanna
- `test` – oblicza średnią ilość ruchów jaką potrzebuje agent by przejść przez zadaną liczbę epizodów. Ustawienie flagi `render` na `True` powoduje że z opóźnieniem 0.1 s będą wyświetlane kolejne klatki działania agenta, co pozwala śledzić go w czasie rzeczywistym

Pozostałe funkcje:

- `meta_test` – testuje skuteczność uczenia dla strategii epsilon zachłannej dla parametrów `alpha,gamma,epsilon` o wartościach 0.1, 0.5, 0.9
- `plot_test_eps, plot_test_boltzmann` – testuje jak szybko polityka zbiegnie do optymalnej wartości i rysuje wykres

Określenie meta-parametrów

Przy użyciu funkcji `meta_test` ustaliłem jak parametry wpływają na wydajność dla 2 000 epizodów uczenia.

Alfa = 0.1

Gamma \ Epsilon	0.1	0.5	0.9
0.1	200.0	184.316	128.39
0.5	200.0	173.331	116.321
0.9	195.764	172.314	97.378

Alfa – 0.5

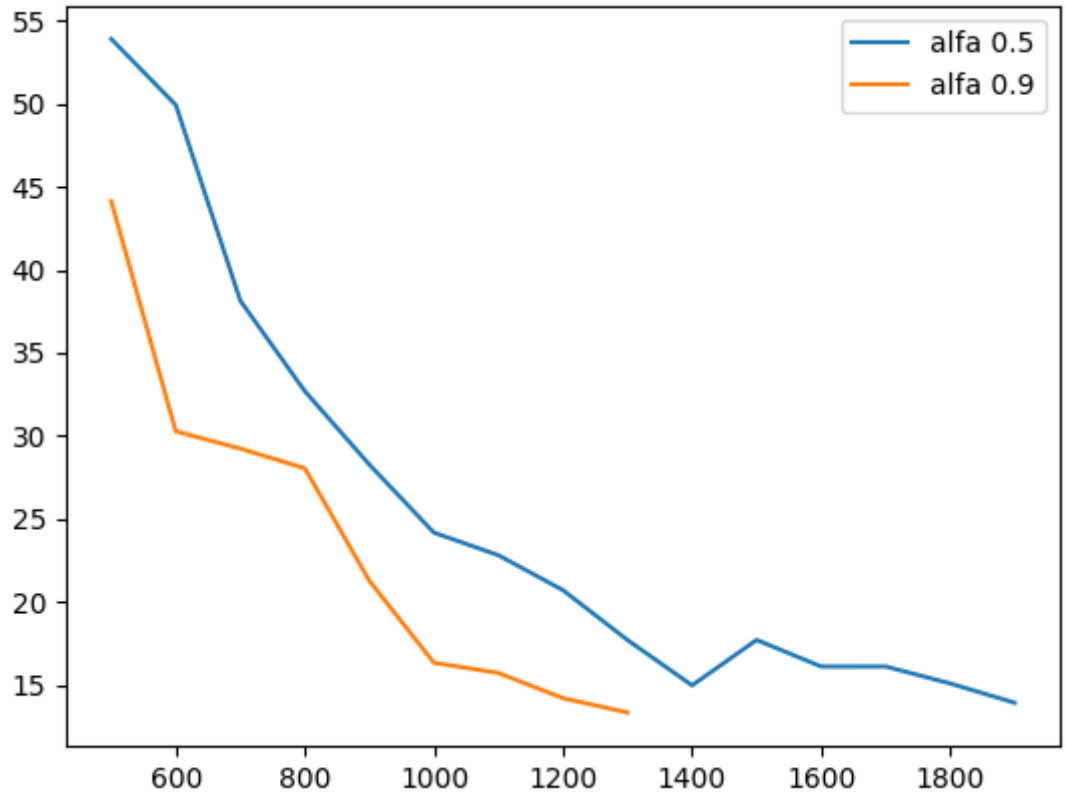
Gamma \ Epsilon	0.1	0.5	0.9
0.1	119.103	62.395	16.57
0.5	84.343	52.641	14.716
0.9	23.897	13.068	13.072

Alfa = 0.9

Gamma \ Epsilon	0.1	0.5	0.9
0.1	40.567	16.362	18.791
0.5	30.739	21.565	15.008
0.9	13.061	13.109	13.082

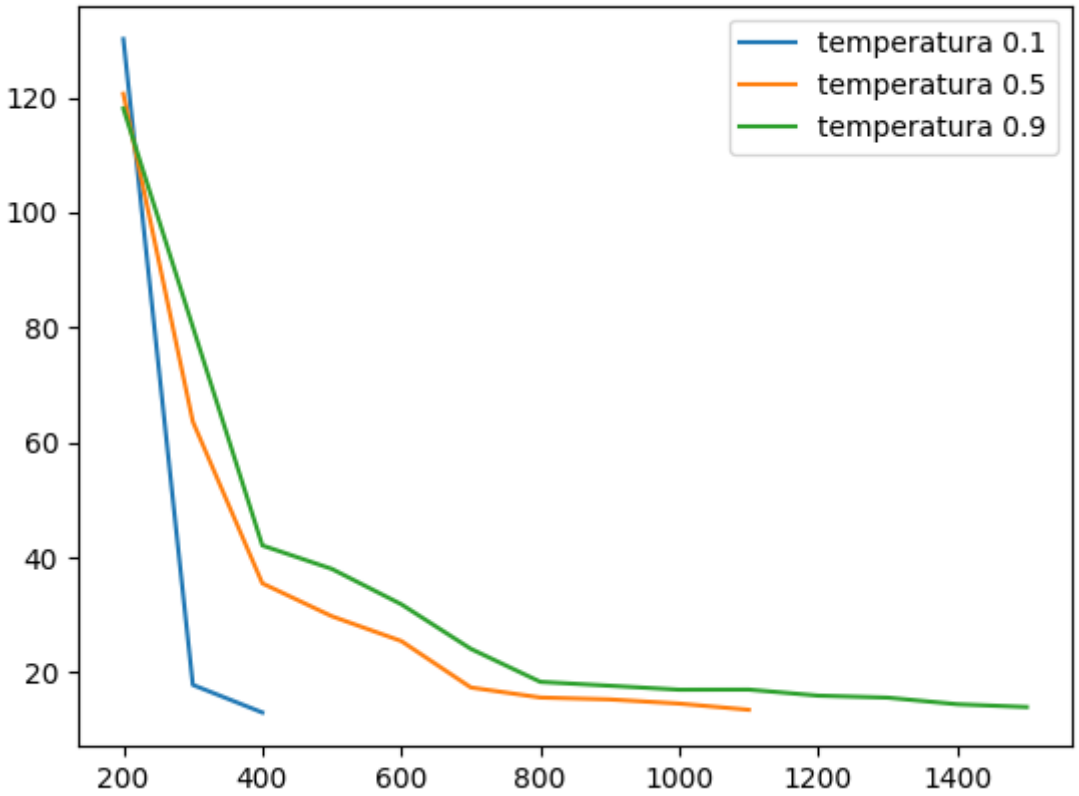
Skuteczność algorytmu jest największa gdy wartości alfa i gamma są wysokie. Jeśli chodzi natomiast o wartość epsilon, to skuteczność algorytmu jest większa gdy przyjmuje dużą wartość, jednak jest to spowodowane tym że wybierając losową ścieżkę agent więcej czasu spędza w każdym epizodzie. Podczas testowania widać było, że dla dużych wartości epsilon każdy epizod zajmuje więcej czasu. Z tego względu przyjmę wartości gamma= 0.9, epsilon = 0.1 jako najbardziej optymalne wydajnościowo.

Ponieważ średnia wartość kroku równa 13 wydaje się być wartością optymalną, sprawdzę jak szybko algorytm do niej zbiega dla wartości alfa = 0.5 i 0.9 używając funkcji `plot_test_eps`



Parametry alfa = 0.9 gamma= 0.9, epsilon = 0.1 okazały się optymalne i pozwoliły osiągnąć optymalną wartość po około 1300 epizodach uczenia.

W ostatni teście sprawdzę czy zastosowanie polityki Boltzmanna poprawi wynik oraz dla jakiej wartości temperatury polityka będzie najszybciej zbiegać do wartości optymalnej `plot_test_boltzmann`



Polityka Boltzmanna radzi sobie najlepiej dla niskiej wartości temperatury-epsilon, ponadto poprawia znacząco tempo uczenia w porównaniu do epsilon zachłannej.

Podsumowanie

Dla prostego problemu algorytm q learning w dość krótkim czasie osiąga zadowalające efekty dla obu przetestowanych polityk. Środowisko Taxi v3 jest jednak w gruncie rzeczy deterministyczne i posiada jedynie 400 stanów osiągalnych i 6 możliwych akcji, co sprawia że problem jest stosunkowo mało skomplikowany.