WSI zad. 6@14c504a6ec54d5d66f7b6d7f9d6cefeb8ef8cf52 **Grzegorz Rypeść**

Tym razem odchodzimy od idei klonowania behawioralnego. Nadal jednak chcemy stworzyć agenta, który będzie dobrze grał w *Snake'a*. Należy wykorzystać uczenie przez wzmacnianie, a konkretnie algorytm uczenia się Q-funkcji ze strategią ε-zachłanną (ang. ε-greedy *Q*-Learning), by agent sam nauczył grać się w grę.

Plik *main.py* do zadania został zmieniony, gra w każdej chwili czasowej zwraca teraz agentowi nagrodę. Celem agenta jest maksymalizacja tej nagrody. Proszę uzupełnić klasę QLearningAgent (*main_rl.py*). Aby uzyskać 7 pkt za zadanie, agent musi uzyskiwać średnio co najmniej 10 punktów w grze, teraz uzyskuje około 3 punkty.

Dla 3 różnych wartości ε oraz wartości γ (dyskonta): 0.8, 0.90, 0.99, proszę przygotować wygładzony wykres średniej nagrody za epizod. Epizod to jedna rozgrywka węża. Proszę zapisać wnioski. Czym jest probem balansu między eksploracją, a eksploatacją (ang. exploration-exploatation trade-off)?

Proszę najlepszą uzyskaną Q-funkcję przetestować dla 100 rozgrywek w wężu pamiętając o przestawieniu parametru ε na 0 (w czasie testu zawsze korzystamy ze strategii zachłannej, a nie ε-zachłannej). Proszę porównać wyniki z zadaniem 4. oraz 5 i zapisać spostrzeżenia.

Raport oraz pliki proszę spakować do pliku o nazwie WSI-6-NAZWISKO-IMIE.zip i przesłać na adres grzegorz.rypesc.dokt@pw.edu.pl.

Uwagi

Gra w węża jest tutaj środowiskiem, zwanym też generatorem danych. Stanem gry jest macierz obejmująca wszystkie bloki gry, gdzie każdy blok zawiera informacje czym jest, np. ścianą, głową węża skierowaną w danym kierunku, jest pusty itp. Proszę napisać w raporcie ile stanów może mieć środowisko. W celu zmniejszenia złożoności obliczeniowej nasz agent nie dysponuje całkowitym stanem gry, lecz jego obserwacją reprezentowaną przez wartości atrybutów. Mamy zatem do czynienia z procesem Markowa w środowisku częściowo obserwowalnym. Algorytm Q-Learning zaprezentowany na wykładzie jest jednak w stanie uzyskać dobre wyniki, choć nie ma gwarancji, że zawsze wyindukuje politykę optymalną.

Q-funkcją w zadaniu może być macierz o wymiarach *liczba obserwacji* x *liczba akcji*. Każdy element tej macierzy mówi jaka jest estymowana Q-wartość danej akcji (w wężu są 4 akcje) dla danej obserwacji. Obserwacja to zaobserwowana wartość atrybutów. Atrybutów należy ustalić co najmniej 9 (dyskretne), 5 jest już zaimplementowanych. Zakładając, że mamy 10 binarnych atrybutów (np. czy jedzenie jest na lewo od głowy węża, czy jest na prawo itd.), to mamy 1024 możliwe obserwacje. Indeksowanie macierzy 1024x4 może być nieeleganckie. Implementacja będzie tutaj dużo prostsza, jeżeli Q-funkcja będzie reprezentowana przez tensor rangi 11, gdzie pierwsze 10 osi to atrybuty przyjmujące binarne wartości, a oś ostatnia to akcje. Tensor taki zawiera estymowane Q-wartości, czyli liczby rzeczywiste. Indeksowanie takiego tensora jest czytelniejsze.

Może się zdarzyć, że na skutek zbyt negatywnej nagrody za śmierć, polityka agenta wpadnie w lokalne maksimum, w którym agent będzie kręcił wężem kółeczka i nie zbierał jedzenia. Należy wtedy odpowiednio przestawić wartości nagród.