Laboratorium 6 - sprawozdanie

Wstęp do sztucznej inteligencji

Marcin Grabysz

12.01.2022

1 Treść zadania

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 (reprezentują dziury o ujemnej punktacji). 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.

2 Wstęp teoretyczny

Q-learning to model algorytmu, w którym pewien twór (agent) porusza się w pewnym środowisku (tj. wykonuje w nim pewne dostępne akcje). W wyniku swoich akcji agent znajduje się w konkretnych stanach (stanem może być na przykład pole na planszy). Za wykonanie danej akcji w danym stanie agent może otrzymać nagrodę - jego celem jest maksymalizacja otrzymywa-

nych nagród.

W tym celu agent dysponuje tabelą funkcji wartości-akcji (ang. *q-table*), w której w miarę postępu uczenia zapisuje, jak bardzo opłacalne jest podejmowanie konkretnego działania w danym stanie.

Epizodem jest pewien zbiór akcji, rozpoczynających się od pewnego stanu startowego. Na początku każdego epizodu agent dysponuje tabelą zgodną ze swoją najlepszą wiedzą uzyskaną w poprzednich epizodach - to umożliwia uczenie się w każdym kolejnym epizodzie.

3 Implementacja

3.1 Wykorzystane narzędzia

Algorytmy i funkcje służace badaniu funkcji zaimplementowane są w języku Python w wersji 3.8.10. Wykorzystane zostały także biblioteki `numpy` oraz `pandas`.

3.2 Opis implementacji

3.2.1 environment.py

W tym module zaimplementowana jest klasa `Environment`. Klasa środowiska spełnia dwie podstawowe funkcje:

- generuje labirynt, po którym porusza się agent
- zwraca informację zwrotną po działaniach podjętych przez agenta

Poniżej opisane są najważniejsze metody klasy `Environmet`:

`generate_correct_board()` - metoda wywoływana w konstruktorze, która zapewnia wygenerowanie labiryntu, w którym isnieje co najmniej jedna ścieżka między punktem startowym i końcowym. Labirynt generowany jest losowo (każde pole może zostać polem blokującym z pewnym prawdopodobieństwem podawanym przez użytkownika jako parametr). Następnie tak wygenerowany labirynt jest sprawdzany pod kątem istnienia ścieżki za pomocą algorytmu przeszukiwania wgłąb. Jeżeli ścieżka nie zostanie znaleziona,

generowany jest nowy labirynt. Po 500 nieudanych próbach funkcja podnosi wyjątek.

`available_actions()` - metoda zwraca listę dostępnych akcji agenta w danym momencie. W przyjętej implementacji akcja jest jednoznaczna z kierunkiem ruchu (góra, prawo, dół, lewo).

`step()` - na podstawie wybranej przez agenta akcji zwraca informacje o jej efekcie, tj. o nowym stanie, otrzymanej nagrodzie oraz o ewentualnym zakończeniu w wyniku wejścia na punkt końcowy lub punkt zablokowany.

`reset()` - metoda przygotowuje środowisko do nowego epizodu (w szczególności ustawia współrzędne agenta na punkt startowy).

3.2.2 main.py

W tym module zaimplementowana jest funkcja `qlearn()`, stanowiąca główną część projektu. W funkcji wykonywane są kolejne epizody, w czasie których uaktualniana jest zmienna *qtable* - tabela funkcji wartości-akcji. Każdy epizod składa się z kroków, w których agent wybiera pewną akcję i komunikuje się ze środowiskiem.

Sposób wyboru akcji zależy od rodzaju agenta. *Random-car* wybiera losową spośród dostępnych akcji. *Q-uber* najpierw decyduje się na eksplorację lub eksploatację (prawdopodobieństwo wyboru eksploracji dane jest parametrem *exploration_rate*, który maleje z czasem). Eksploracja polega na wykonaniu losowego ruchu w celu uzyskania nowych informacji. Eksploatacja polega na wybraniu najlepszej akcji według obecnego stanu *qtable*.

Po wykonaniu dowolnej akcji, odpowiednia komórka tabeli funkcji wartości-akcji jest aktualizowana - stanowi to podstawę działania algorytmu. Przyjmijmy następujące oznaczenia:

- a - wybrana akcja
- a' - akcja możliwa do wykonania w następnym ruchu
- s - stan, w którym podjęta została akcja a

- s' - stan, w którym znalazł się agent po podjęciu akcji a
- $q(s, a)$ - funkcja wartości akcji określająca jak dobre jest podjęcie akcji a w stanie s
- R - nagroda za wykonanie akcji a w stanie s

Oraz:

- α - współczynnik uczenia (ang. *learning rate*)
- γ - dyskonto (ang. *discount rate*)

Wówczas pozycja w tabeli aktualizowana jest zgodnie ze wzorem:

$$q^{new}(s, a) = (1 - \alpha)\, q(s, a) + \alpha(R + \gamma \max q(s', a'))$$

Kolejnym parametrem, który zmienia wartość po wykonaniu każdego kroku jest *exploration rate* ϵ. Wspomniany w jednym z poprzednich akapitów ϵ jest prawdopodobieństwem zdarzenia, że agent wybierze eksplorację zamiast eksploatacji. Ponieważ na początku procesu uczenia celem agenta jest poznanie środowiska, chętnie będzie wybierał eksplorację (dlatego początkowo $\epsilon = 1$), jednak w miarę postępów w uczeniu, zależy nam na tym, aby wykorzystywał nabytą wiedzę do posuwania się wgłąb labiryntu - dlatego w miarę kolejnych kroków i zmniejszania się *exploration rate* agent coraz chętniej decyduje się na eksploatację. Po każdym kroku nowy ϵ wyznaczany jest według następującego wzoru:

- $\epsilon_{min}, \epsilon_{max}$ - minimalna i maksymalna wartość *exploration rate*, zdefiniowane przez użytkownika
- d_ϵ - *exploration decay rate* - wskaźnik określający szybkość z jaką maleje ϵ
- t - numer epizodu

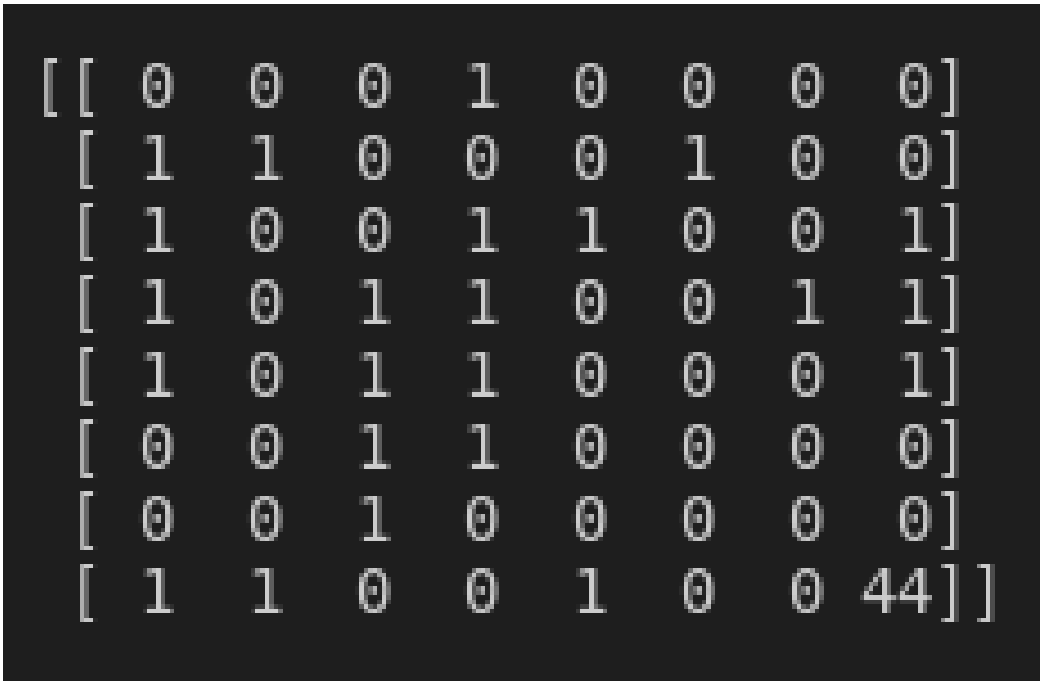
$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-d_\epsilon t}$$

Funkcja zbiera także dane o konkretnym wykonaniu i zapisuje je w plikach json oraz csv. Najważniejsze zachowywane dane to: średnia nagroda w kolejnych n epizodach (gdzie $n = \frac{\text{liczba epizodów}}{100}$), średnia liczba kroków w kolejnych n epizodach, epizod, w którym wystąpił pierwszy sukces, łączna liczba epizodów zakończonych sukcesem.

Pomocna przy analizowaniu wyników jest także funkcja `generate_path()`, która na podstawie dostarczonej tabeli funkcji wartości-akcji generuje listę najlepszych kroków - lista omożliwia własnoręczne przeanalizowanie, czy algorytm znalazł najkrótszą ścieżkę.

4 Środowisko

Badanym środowiskiem jest plansza 8x8, w której punktem startowym jest (0, 0) a punktem końcowym - (7, 7) (kolejno: lewy górny róg, prawy dolny róg). Prawdopodobieństwo, że dane pole zostanie zablokowane wynosi 0,4. Po załadowaniu generatora liczb losowych ziarnem równym 1, generowana jest plansza przedstawiona na rysunku poniżej. Cyfry 0 oznaczają wolne pola, a cyfry 1 - pola zablokowane. Liczba 44 (w hołdzie narodowemu wieszczowi) oznacza punkt końcowy. Nie należy mylić tych wartości z nagrodami przypisanymi danym polom - na schemacie znajdują się jedynie poglądowo.



Rysunek 1: Środowisko

Korzystając z najkrótszej ścieżki, agent może przedostać sie z punktu startowego do punktu końcowego w 18 krokach. Co więcej, w labiryncie istnieje dość głęboka fałszywa odnoga, dodatkowo utrudniająca agentowi zadanie.

Labirynt zawiera trzy typy pól, nazwane obrazowo *good quare* (pola, po których może poruszać się agent), *bad square* (pola, po których nie może się poruszać) oraz *end square* (punkt końcowy). Punkt startowy jest zwykłym "dobrym" polem. Podczas tworzenia środowiska, każdemu typowi pól można przypisać konkretną wartość nagrody, którą otrzyma agent za wejście na pole danego typu.

5 Badanie implementacji

5.1 Agent "Random car"

Na początku dajmy szansę na odnalezienie drogi przez labirynt agentowi "Random car", który wykonuje ruchy losowe. Labirynt zachowuje taki kształt, jak na rysunku 1. Wartości nagrody przypisane każdemu z pól nie są istotne - "Random car" nie bierze ich pod uwagę.

Pomimo 100 000 prób agentowi ani razu nie udaje się odnieść sukcesu. Trudno się temu dziwić, bo dojście do punktu końcowego wymaga wykonania co najmniej osiemnastu kroków, czyli podjęcia co najmniej osiemnaście razy słusznej (a przynajmniej bezpiecznej) decyzji.

Co ciekawe, maksymalna liczba kroków, jaką udało się wykonać agentowi w czasie jednego epizodu, wynosi 24. To oznacza, że właśnie tyle razy pod rząd "Random car" podjął bezpieczną decyzję i nie wjechał na "złe" pole - nie jest to jednak jednoznaczne z tym, że choćby zbliżył się do pola końcowego, gdyż nie został zaimplementowany żaden mechanizm zabezpieczający agenta przed cofaniem się po własnych śladach (te 24 kroki mogą być równie dobrze efektem wykonania 12 razy sekwencji prawo-lewo). Oznacza to jednak, że punkt końcowy *może* zostać osiągnięty (do tego wystarczy - dla przypomnienia - 18 kroków), ale jest to niezwykle mało prawdopodobne. Największa średnia liczba kroków na 1000 kolejnych epizodów wynosi 1,527, zatem epizod z liczbą 24 kroków należy traktować jako ewenement - zdecydowanie częściej "Random car" rozbijał się już po 1-2 krokach.

O agencie poruszającym się losowo nie można już powiedzieć nic ciekawego - dalsza część sprawozdania jest poświęcona agentowi uczącemu się, który jest prawdziwym przedmiotem omawianego badania.

5.2 Przykład uczenia się

Wykonajmy proces uczenia się agenta "Q-uber" z następującymi parametrami:

Parametry środowiska:

- *bad square* $= -30$
- *good square* $= -1$
- *end square* $= 30$

Pozostałe parametry środowiska nie zmieniły się (i nie zmienią się podczas całego badania), zatem kształt labiryntu pozostaje taki jak na rysunku 1.

Parametry uczenia się (najważniejsze):

- liczba epizodów = 100000
- *learning rate* $= 0,3$
- *discount rate* $= 0,99$
- $\epsilon_{min} = 0,01$
- $\epsilon_{max} = 1$
- $d_\epsilon = 0,0001$

Omówmy wszystkie dane, jakie zgromadziła funkcja `qlearn()` (zapisane są w plikach quber_1.csv oraz quber_1.json, które zostaną dołączone do niniejszego sprawozdania). Ze względu na ich ilość, nie będziemy tego robić dla każdego wykonania funkcji podczas późniejszego badania.

5.2.1 Najkrótsza ścieżka

Najważniejszym pytaniem, na które należy sobie odpowiedzieć jest: czy agent nauczył się najkrótszej ścieżki? Aby się o tym przekonać, zachowano ostateczną tabelę funkcji wartości-akcji. Na jej podstawie, za pomocą funkcji `generate_path()` można wygenerować listę najlepszych akcji:

['right', 'right', 'down', 'right', 'right', 'up', 'right', 'right', 'down', 'down', 'left', 'down', 'down', 'right', 'down', 'right', 'down', 'down']

Wygenerowana sekwencja osiemnastu kroków jest poprawnym sposobem najszybszego przedostania się z punktu startowego do końcowego, poruszając się po "dobrych" kwadratach, o czym czytelnik może się przekonać osobiście, porównując ją z labiryntem z rys. 1 (nie jest jedynym poprawnym sposobem, ale wynika to ze specyfiki danego labiryntu - ostatnie kilka kroków może przyjąć parę równie dobrych wariantów). Stanowi to dowód na to, że algorytm z podanymi paramatrami, optymalny lub nie - jest skuteczny.

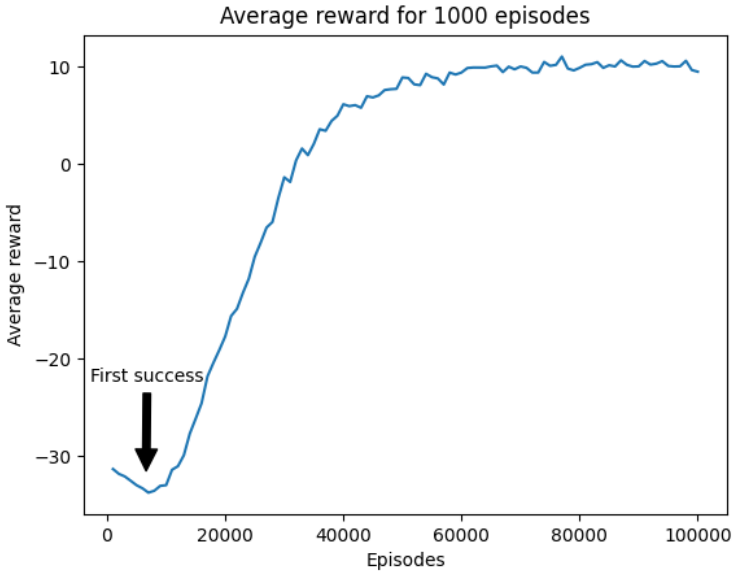
5.2.2 Osiągi w kolejnych epizodach

Podczas wykonania funkcji zgromadzone zostały dane następujących typów:

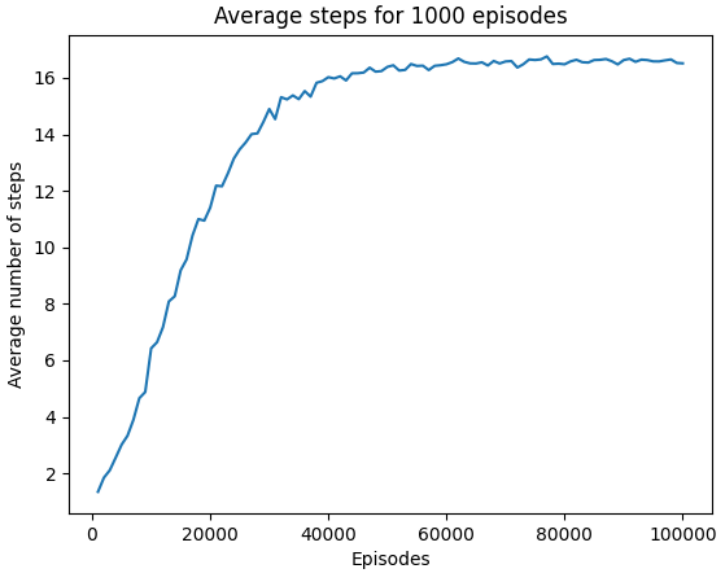
1. Średnia nagroda otrzymana w czasie kolejnych 1000 epizodów - wykres ten jest czytelniejszy i mówiący więcej niż wykres nagrody w *każdym* epizodzie. Wartość na osi y w punkcie x to w rzeczywistości średnia nagroda z epizodów od $x - 999$ do x.

2. Średnia liczba kroków w czasie kolejnych 1000 epizodów - analogicznie.

3. Wartość *exploration rate* w czasie kolejnych epizodów.



(a) Średnia nagroda



(b) Średnia liczba kroków

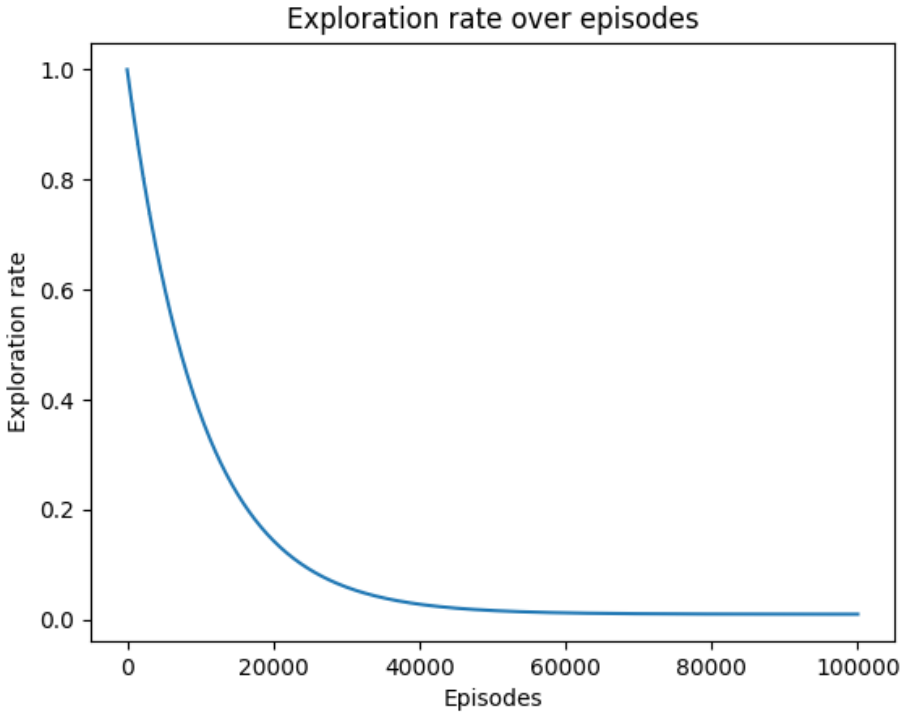
Rysunek 2: Średnia nagroda i liczba kroków

Wykres średniej nagrody (rys. 2a) zachowuje się w przewidywalny sposób. Pierwsze próby to głównie ruchy losowe (prawdopodobieństwo eksploracji jest bardzo duże), które szybko kończą się wejściem na "złe" pole i nagrodą o wartości -30.

Ponieważ pola tabeli funkcji wartości-akcji inicjowane są wartościami 0, zaś nagroda za wejście na "dobry" kwadrat wynosi -1 ("dobry" kwadrat nie jest zatem taki dobry), agentowi zawsze bardziej opłaca się wejść na pole niezbadane, niż na pole, o którym już wie, że oznacza ujemną nagrodę. Dlaczego to bardzo ważne, zostanie wyjaśnione w kolejnych badaniach.

Zmotywowany w ten sposób agent w kolejnych epizodach kontynuuje zagłębianie się w labirynt, dlatego też początkowo średnia liczba kroków rośnie, a wartość nagrody maleje. Maleje, bo agent przechodzi przez więcej pól "dobrych", a kończy i tak na jednym z pól "złych", na każdym z odwiedzonych pól otrzymując pewną ujemną nagrodę.

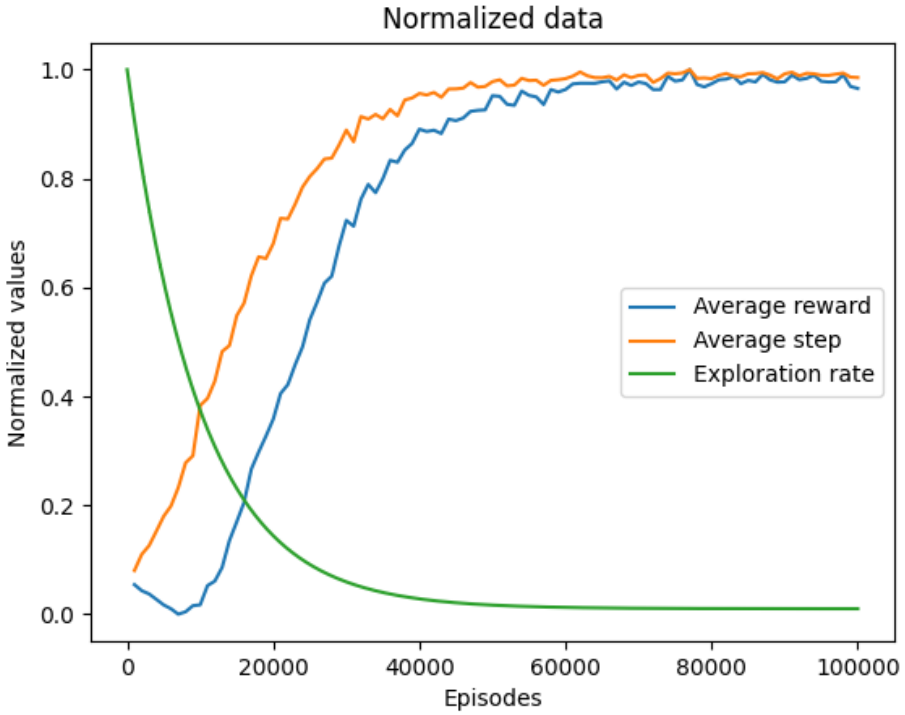
Sytuacja odwraca się po tym, jak "Q-uber" odnosi pierwszy sukces (ma to miejsce się to w epizodzie 6684.) i zasmakowuje dodatniej nagrody $+30$. Od tej pory, za każdym razem gdy wybierze eksploatację zgodnie z wartościami tabeli funkcji wartości-akcji, podejmuje akcję, która przybliża go pola końcowego. Co więcej, w miarę wykładniczego spadku *exploration rate* (wykres poniżej), eksploatacja jest wybierana coraz częściej.



Rysunek 3: Wartość ϵ w kolejnych epizodach

Maksymalna nagroda, jaką może otrzymać agent, pokonując labirynt, wynosi 13 (+30 za wejście na pole końcowe i -17 za siedemnastokrotne wejście na pole "dobre"). Dlaczego więc średnia nagroda zatrzymuje się na poziomie ok. 10? Należy pamiętać, że zgodnie z parametrem ϵ_{min}, *exploration rate* nie spada poniżej 0,01 - dlatego jedna na sto podjętych akcji jest eksploracją (jest losowa), co często musi się kończyć wejściem na "złe" pole i nagrodą -30. Z tego samego powodu średnia liczba kroków w epizodzie nie osiąga wartości 18, mimo że agent "zna" najlepszą ścieżkę.

Zamiast omawiać trzy powyższe wykresy osobno, spróbujmy spojrzeć na nie całościowo i dostrzec, w jaki sposób prawdopodobieństwo eksploracji, średnia nagroda i średnia liczba kroków są ze sobą powiązane. W tym celu, wartości tych trzech parametrów zostały znormalizowane w przedziale (0, 1) i umieszczone na jednym wykresie:



Rysunek 4: Wykres znormalizowanych danych

Z powyższego wykresu widać, że w okolicach 60 000. epizodu, średnia nagroda i liczba kroków ustalają się na pewnym poziomie, tak samo jak wartość ϵ (wiemy, że funkcja wykładnicza w istocie nigdy nie jest stała, ale w przedziale (60 000, 100 000) jej pochodna jest tak bliska zeru, że w badanym zagadnieniu możemy uznać funkcję za stałą). Innymi słowy - agent uczy się, dopóki relatwynie często decyduje się na eksplorację. Czy to oznacza, że jesteśmy w stanie wymusić szybszą naukę agenta przyśpieszając spadek *exploration rate*? Czy też wręcz przeciwnie - jeżeli ϵ zbyt szybko osiągnie wartość minimalną, "Q-uber" nie zdąży nauczyć się najkrótszej trasy? Wpływ *exploration decay rate* jest przedmiotem kolejnych badań.

5.3 Wpływ zaniku *exploration rate*

Przeprowadzono badanie dla następujących wartości parametru *exploration decay rate*: 0,00005, 0,0001, 0,0005, 0,001, 0,005, 0,01, 0,05. Pozostałe parametry funkcji się nie zmieniły. Za każdym razem algorytm zadziałał skutecznie, czyli znalazł najkrótszą sekwencję osiemnastu kroków do punktu końcowego.

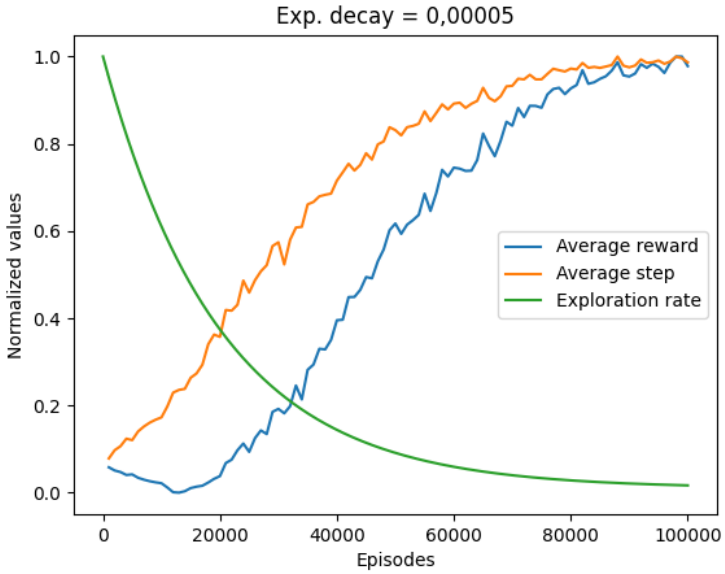
Szczegółowe dane są zilustrowane w poniższej tabeli i na wykresach:

L.p.	L. epizodów	Wsp. uczenia	Zanik ϵ	L. sukcesów	Pierwszy sukces
0	100 000	0,3	0,00005	49 989	11 929
1	100 000	0,3	0,0001	72 097	6 684
2	100 000	0,3	0,0005	90 338	2 013
3	100 000	0,3	0,001	92 537	1 151
4	100 000	0,3	0,005	94 505	371
5	100 000	0,3	0,01	94 437	204
6	100 000	0,3	0,05	94 839	75

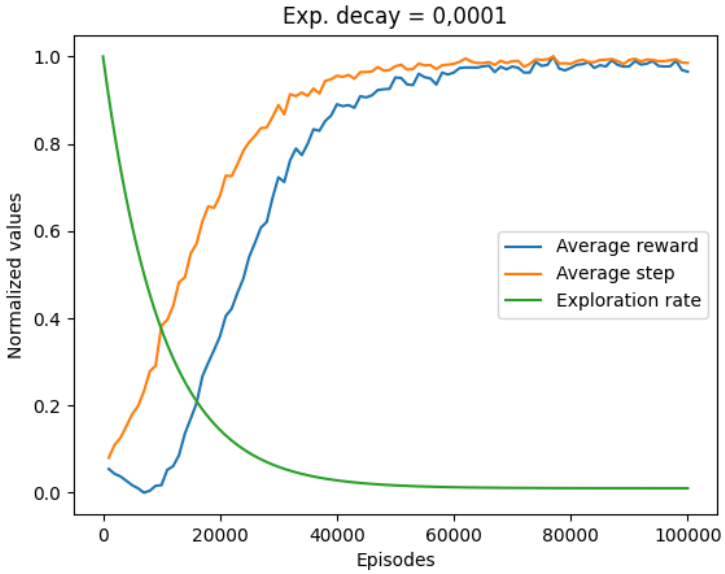
Tabela 1: Wpływ *exploration decay rate*

Okazuje się, że początkowo przyjęte parametry (ustalone mniej lub bardziej przypadkowo) są dalekie od optymalnych. Dla większych wartości *exploration decay rate* agent uczy się o wiele szybciej i nie ma potrzeby wykonywać tak wielkiej liczby epizodów. Przy tym jakość nauki pozostaje taka sama, bo agent nie może nauczyć się bardziej, niż znaleźć sekwencję 18 najlepszych kroków (i uznawać ją za najlepszą).

Na poniższych wykresach zostało przedstawione zestawienie średniej nagrody, średniej liczby kroków i wartości *exploration rate*, przy czym: Dla badań 0. i 1. wykresy przedstawiają wszystkie epizody, dla badań 2. i 3. wykresy przedstawiają tylko 25 000 pierwszych epizodów, a dla pozostałych badań wykresy zostały pominięte.

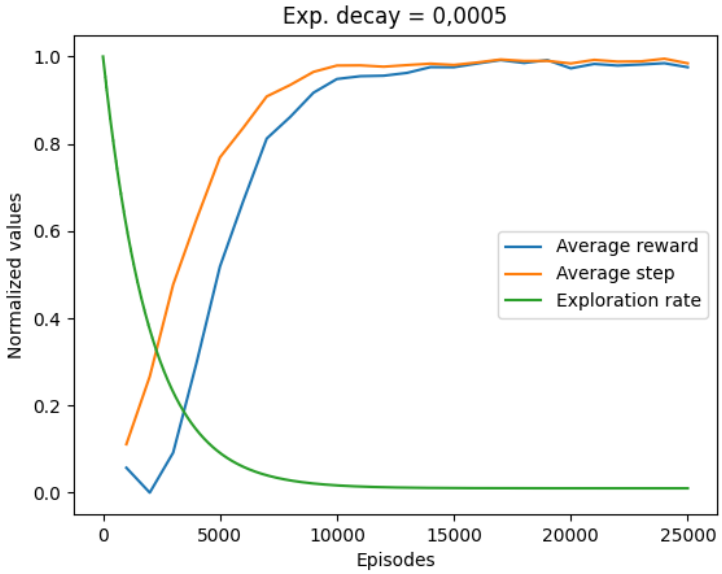


(a) Wykres znormalizowanych danych dla $d_\epsilon = 0,00005$

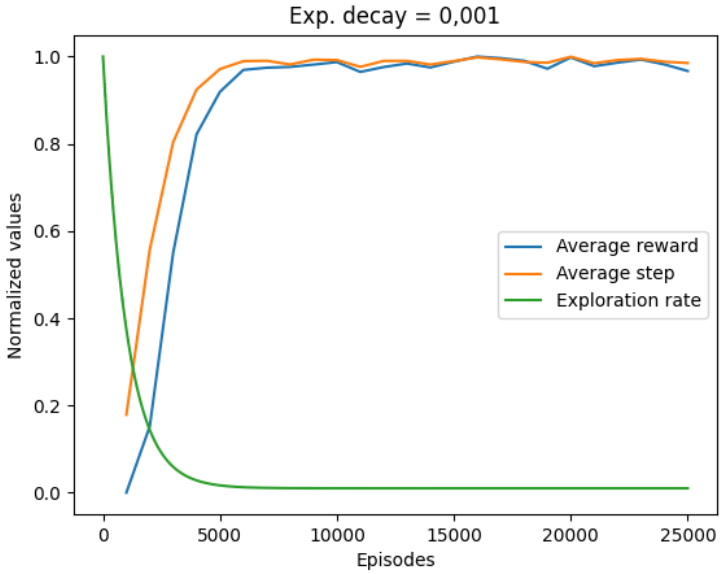


(b) Wykres znormalizowanych danych dla $d_\epsilon = 0,0001$

Rysunek 5: Znormalizowane dane dla różnych wartości d_ϵ



(a) Wykres znormalizowanych danych dla $d_\epsilon = 0,0005$



(b) Wykres znormalizowanych danych dla $d_\epsilon = 0,001$

Rysunek 6: Znormalizowane dane dla różnych wartości d_ϵ

5.4 Wpływ współczynnika uczenia się

Wartości uzyskane dla różnych współczynników uczenia się są przedstawione w poniższej tabeli. W każdym z przypadków została znaleziona najkrótsza ścieżka.

L.p.	L. epizodów	Wsp. uczenia	Zanik ϵ	L. sukcesów	Pierwszy sukces
0	5 000	0,1	0,05	4 666	44
1	5 000	0,3	0,05	4 695	37
2	5 000	0,5	0,05	4 726	39
3	5 000	0,7	0,05	4 728	37
4	5 000	0,9	0,05	4 729	40

Tabela 2: Wpływ współczynnika uczenia się

Daje się zauważyć, że w przyjętej implementacji wartość współczynnika uczenia się nie ma za dużego wpływu na jakość uczenia. *Learning rate* określa, jak bardzo agent jest skłonny do porzucania dotychczasowych wartości tabeli na rzecz nowych danych. Być może badany problem jest za mało złożony - w środowisku występują tylko trzy wartości nagrody, w dodatku dość od siebie odległe (-30, -1 oraz $+30$). Dla agenta, który początkowe epizody kończy z nagrodami na poziomie -30, znalezienie pola końcowego i otrzymanie dodatniej nagrody powoduje wielką różnicę, niezależnie od tego, czy do tabeli funkcji wartości-akcji zapisze wartość nagrody przemnożoną przez mały współczynnik.

5.5 Wpływ nagród

W poniższej tabeli przedstawione są dane uzyskane dla różnych wartości nagród przypisanych *good_square, bad_square* oraz *end_square*. Wszystkie badania zostały przeprowadzone przy następujących parametrach:

- liczba epizodów = 1000
- *learning rate* = $0,3$
- $d_\epsilon = 0,05$

L.p.	*bad sq.*	*good sq.*	*end sq.*	L. sukces.	Pierw. sukces	Popr. ścieżka
0	-30	-1	30	862	80	tak
1	-5	-1	5	0	-	nie
2	-10	-1	20	5	92	nie
3	-10	-1	10	5	92	nie
4	-30	0	30	0	-	nie

Tabela 3: Wpływ nagród

Powyższe wyniki są dowodem na to, że odpowiednie wartości nagród są kluczowe w procesie uczenia. Tylko w jednym z pięciu przypadków agentowi udało się znaleźć najkrótszą ścieżkę do celu. Dlaczego tak się stało?

W niemal każdym przypadku wyjaśnienie będzie nieco inne. Zauważmy, że w próbie 1. agent tak naprawdę znalazł ścieżkę najlepszą, tj. najlepiej punktowaną. Wykonanie 17 kroków na pole "dobre" po to, aby raz wejść na pole końcowe, dawałoby mu łączną nagrodę o wysokości -12; zatem "Q-uber" nauczył się ścieżki optymalnej, którą jest... natychmiastowe wjechanie w stażystę, czyli pole "złe" i zakończenie epizodu z nagrodą -5. Tak właśnie wygląda jego *q-table* - z punktu startowego najlepszą akcją jest "down".

Przypadki 2. i 3. są do siebie podobne. Wagi nagród są dobrane w taki sposób, żeby dojście do pola końcowego najkrótszą drogą w istocie dawało najwyższą wypłatę ($+3$ w przypadku 2. i -7 w przypadku 3.). Okazało się to niewystarczające dla agenta, nawet pomimo tego że w ciągu 1000 epizodów dotarł 5 razy do pola końcowego (przypomnę, że przy każdym teście

generator liczb losowych jest inicjowany tym samym ziarnem - stąd dokładne podobieństwo przypadków 2. i 3.). Być może jednak "Q-uber" przemieścił się na pole końcowe po raz pierwszy nie najkrótszą drogą, przez co nie "uświadomił sobie", jak korzystne jest odebranie nagrody z pola końcowego. Być może w momencie badania obszaru labiryntu leżącego wokół pola końcowego *exploration rate* był już zbyt mały, aby pozwolić agentowi na wykonanie paru losowych ruchów i w wyniku tej przygody - na znalezienie lepszej ścieżki. Być może "Q-uber" miał wyjątkowego pecha i w wyniku zainicjowania generatora liczb losowych innym ziarnem znalazł by poprawną ścieżkę. Znalezienie najlepszej sekwencji ruchów powinno być możliwe przy tych wartościach nagród, ale najwyraźniej pozostałe parametry nie zostały odpowiednio dobrane. Po raz kolejny agent uznał, że optymalną strategią jest popełnienie samobójstwa przez akcję "down" w pierwszym ruchu. Dużo więcej pewności daje wybranie nagród, które w wyraźniejszy sposób karają za wejście na pole "złe" i premiują za wejście na pole końcowe.

Badanie 4. różni się od pozostałych przypadków tym, że agent nie jest karany za przebywanie na polu "dobrym". Oznacza to jednak, że nie ma też żadnego powodu, dla którego miałby eksplorować labirynt - każdy ruch "do tyłu", na znane pole "dobre" jest równie opłacalny, co wejście na pole nieznane. Przykład "Random cara" pokazał również, że nie ma co liczyć na to, że ruchy losowe doprowadzą agenta do pola końcowego. Agent nie premiowany za poznawanie nieodkrytych stanów pozostaje na terenie sobie znanym. I taka w istocie jest propozycja "Q-ubera" z przykładu 4. po 1000 epizodach nauki - nieskończona sekwencja postaci: "right", right", "down", "up", "down", "up", "down"... itd.

6 Wnioski

Zagadnień związanych z *Q-learningiem* jest zbyt wiele, aby omówić je dokładnie na kartach niniejszego sprawozdana, jednak przedstawione badania dowodzą, że dana implementacja algorytmu *Q-learning* jest skuteczna. Liczne parametry funkcji oraz środowiska wpływają na zachowanie agenta i proces uczenia w wyraźny i przewidywalny sposób. Daje się również łatwo określić, dlaczego pewne ustawienia prowadzą do znalezienia poprawnego rozwiązania, a inne je uniemożliwiają.