Uczenie ze wzmocnieniem

Celem tego laboratorium jest zaznajomienie się z koncepcją uczenia ze wzmocnieniem na przykładzie algorytmów Q-Learning i SARSA.

[Zadanie 0] Tak oznaczono zadania do wykonania.

Q-Learning

Poniższy wzór prezentuje podstawowy schemat działania algorytmu Q-Learning.

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Schemat ten powinien zostać omówiony przez prowadzącego w trakcie wprowadzenia do zajęć. Jednakże jedynym sposobem na utrwalenie wiedzy jest powtarzanie, więc pod spodem jeszcze raz krótkie przypomnienie najważniejszych faktów (głównie tych pomocnych w implementacji, nie należy traktować poniższego tekstu jak wpisu encyklopedycznego – w szczególności pominięto dyskusję na temat tego, w jakich sytuacjach Q-Learning jest skuteczny, a w jakich nie).

Zakładamy, że nasz byt uczący się do agent posiadający dwie poniższe zdolności:

- Postrzegania aktualnego stanu środowiska w którym egzystuje. Stan taki oznaczamy zwyczajowo literą ***s*** (state) z indeksem ***t*** (time) oznaczającym czas, w którym dany stan miałby być postrzegany. Stanem takim może być np. obraz postrzegany przez kamerę, odczyty z sensorów autonomicznego robota, położenie pionków na planszy do gry…czy nawet treść maila ocenianego przez filtr antyspamowy. Uwaga! Stan postrzegany przez agenta nie musi być pełnym opisem stanu całego środowiska – nie zawsze wiemy co czai się za rogiem.
- Wykonywania akcji, które potencjalnie mają wpływ na stan środowiska i jego miejsce w tymże. Akcje oznaczamy literą ***a*** (action). Akcją może być wciśnięcie mocniej pedału gazu, przesunięcie figury szachowej, strzał w grze FPS, nawiercenie gruntu…a nawet nie robienie kompletnie niczego (decyzja o bierności również jest decyzją).

Dodatkowo potrzebny jest nam zewnętrzny nadzorca, który potrafi oceniać skutki naszych działań.

- Za każdym razem gdy w stanie ***s*** wykonamy akcję ***a*** nadzorca ocenia skutki takiego działania i przyznaje pewną nagrodę ***r*** (reward). Nagroda może być ujemna – i zwykle nagrodę taką nazywamy po prostu karą. Nagrody i kary powinny określać pożądane i niepożądane zachowania. Przyznajemy nagrodę za zaparkowanie we właściwym miejscu, fraga, czy zbicie figury bez straty własnej. Przyznajemy karę za stratę bierki, uszkodzenie pojazdu, czy zmarnowane paliwo.

Q-learning (jak sama nazwa wskazuje) dodaje jeszcze jeden element do układanki: funkcję ***Q*** (quality). Dla danej pary „stan i akcja jaką planujemy wykonać” zwraca ona przewidywaną sumę nagród, jakie dostaniemy w przyszłości w konsekwencji wykonania tej akcji. Uwaga! Predykcja ta (szczególnie na początku procesu uczenia) nie musi być trafna i może mocno odbiegać od tego, co

wydarzy się w rzeczywistości. Funkcja ***Q*** wyraża właśnie całą zgromadzoną (na bazie wcześniejszych doświadczeń) przez agenta wiedzę.

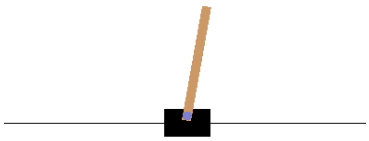
Jak ją zaimplementować? Na potrzeby tego laboratorium skorzystamy z jednego z najprostszych wariantów – słownika ***(s, a) -> r***. W praktyce wykorzystujemy bardziej złożone modele reprezentacji wiedzy – np. wielowarstwową sieć neuronową.

Podstawowa pętla ucząca w Q-Learning wygląda następująco:

1. Podejmij decyzję w jaki sposób wybrać następną akcję. Pojawia się pytanie, czy bazować na wcześniejszej wiedzy, czy też wykonać eksperyment. Szansę na ten drugi scenariusz opisuje parametr sterujący $\boldsymbol{\varepsilon}$ (experiment rate).
    a. Dla danego stanu $\boldsymbol{s_t}$ rozważ wszystkie możliwe akcje $\boldsymbol{a}$. Wybierz tą, dla której $\boldsymbol{Q(s_t,a)}$ będzie największe. To z punktu widzenia posiadanej wiedzy decyzja racjonalna.
    b. Ze zbioru dostępnych akcji wybierz losowo akcję $\boldsymbol{a}$. To zaś eksperyment pozwalający na eksplorację nieprzetestowanych jeszcze możliwości.
2. Wykonaj wybraną akcję $\boldsymbol{a_t}$.
3. Zaobserwuj nowy stan środowiska $\boldsymbol{s_{t+1}}$.
4. Otrzymaj nagrodę o wartości $\mathbf{r_t}$.
5. Zaktualizuj wartość przypisaną w słowniku do $\boldsymbol{Q(s_t,a_t)}$ w oparciu o wzór podany na początku sekcji. Zwróć uwagę na poniższe kwestie.
    a. Nowa wartość predykcji jest średnią ważoną uprzednio znajdującego się pod tym kluczem wpisu (dotychczasowa wiedza) oraz wniosków z nowego doświadczenia (człon w nawiasie). Stosunek między jednym a drugim (wpływający na to jak „konserwatywny i stabilny w poglądach" jest nasz algorytm) wyraża współczynnik $\boldsymbol{\alpha}$.
    b. Predykcja składa się z dwóch członów: nagrody już otrzymanej ($\boldsymbol{r_t}$) oraz przewidywanych nagród w kolejnych krokach (definicja jest jak widać rekurencyjna z natury). Ten drugi człon generuje jednak dwa dodatkowe problemy. Po pierwsze jest z definicji mniej wiarygodny (rzeczywista nagroda już „w rękach", a luźne przewidywania na temat tego jakie nagrody jeszcze nas spotkają). Po drugie zaś (poprzez tą właśnie rekurencyjną zależność) powodowałby on, że wszystkie ścieżki do nagrody są równie dobre, tak długo jak „kiedyś tam" ją dostaniemy. W związku z tym osłabia się wpływ tego składnika wykorzystując współczynnik $\boldsymbol{\gamma}$.
6. Wykonaj ponownie punkt pierwszy.

Warto zwrócić uwagę, że parametry algorytmu nie muszą być stałe w trakcie całego procesu uczącego. W szczególności $\boldsymbol{\varepsilon}$ i $\boldsymbol{\alpha}$ powinny startować z dużych wartości (otwartość na nową wiedzę, skłonność do poszukiwać), a następnie spadać (mniej nieodpowiedzialnych eksperymentów, bazowanie na wcześniejszych doświadczeniach).

Balansowanie kijkiem na wózku



Pierwszy problemem, który rozwiążemy z użyciem Q-Learningu, jest próba utrzymania w pozycji pionowej kijka umieszczonego na poruszającym się wózku (patrz obrazek powyżej). Problem ten

pochodzi z biblioteki Gym gromadzącej zadania testowe dla algorytmów uczących się. W tym celu skorzystamy z umieszczonego w pliku balance.py schematu rozwiązania. Omówmy w zarysie jego zawartość.

- Moduł ten zawiera minimalną realizację algorytmu, która zachowuje się w sposób zupełnie losowy, ale dokonuje prawidłowych interakcji z bibliotecznym API. Możesz go uruchomić, by zobaczyć kolejne nieudolne próby balansowania kijkiem.
- Uwaga! Linijka 33 zawiera polecenie wyrenderowania wizualizacji środowiska. Jeżeli kod po uruchomieniu „nic nie robi" to oznacza to, iż wykorzystywana maszyna wirtualna nie pozwala na rendering z użyciem tej biblioteki. Wykomentuj tą linię – w praktyce i tak znacząco spowalnia proces uczenia (lepiej robić je „bez podglądu").
- Funkcja main (linia 53) tworzy nowy byt uczący i uruchamia go na 10000 podejść do problemu.
- Linia 7 zawiera konstruktor klasy.
    - Tworzymy środowisko na bazie przykładu z biblioteki o nazwie „CartPole-v1" (linia 8).
    - Ustawiamy licznik prób na 1 (linia 9).
    - Definiujemy górne i dolne granice poszczególnych zmiennych opisujących stan (linie 11-21). Będą przydatne podczas jego dyskretyzacji.
        - Pierwsza z nich to pozycja wagonika w osi lewo-prawo. Ma ona zakres od -2.4 do 2.4, definiowany przez bibliotekę (i z tego zakresu skorzystamy).
        - Druga to prędkość wagonika w tej osi. W teorii może być ona dowolna (stąd zakres od -inf do inf deklarowany przez bibliotekę). W praktyce znacznie wygodniej będzie założyć, że mieści się on między -0.5 a 0.5.
        - Trzecia to wychylenie kątowe drążka od pozycji równowagi. Tu również skorzystamy z wartości deklarowanych przez bibliotekę.
        - Ostatnią jest prędkość kątowa drążka. Ponownie koniecznie było zawężenie rozważanych wartości do rozsądnego zakresu.
- Metoda learn (linia 23) zawiera główną pętlę uczącą. Wykonujemy zadaną liczbę podejść, po każdym z nich wypisując otrzymaną sumę nagród.
- Metoda attempt (linia 28) definiuje przebieg jednego podejścia.
    - Najpierw resetuje stan środowiska (linia 29) i zeruje zmienną sumującą nagrody (linia 31).
    - Następnie wybiera akcję do podjęcia (linia 34).
    - Deklaruje jej wykonanie i poznaje rezultaty (linia 35).
    - Aktualizuje wiedzę (linia 37).
    - Oraz aktualizuje stan zgromadzonych nagród (linia 39).
        - W tym problemie dostajemy 1 jednostkę nagrody za każdą iterację w trakcie której kijek nie przewrócił się.
    - Kroki te powtarza aż spełniony zostanie jeden z warunków stopu (zmienna done).
        - Pręt przechyli się poniżej granicznego wychylenia kilkunastu stopni.
        - Wagonik wyjedzie poza dozwolony obszar.
        - Uda się utrzymać pręt w pionie przez 500 jednostek czasu. Sukces!
- Metoda discretise (linia 43) jest pierwszą z tych, które wymagają właściwej implementacji. Aby skorzystać ze słownika jako reprezentacji wiedzy musimy dokonać dyskretyzacji otrzymanych obserwacji stanu poprzez podział na kubełki. W przeciwnym wypadku nie będzie szans na uczenie się (algorytm musi trafić wielokrotnie na dany klucz w słowniku, by ustalić jego finalną wartość). W końcu pozycja -1.353 to coś zupełnie innego niż -1.357. Możemy jednak reprezentować obie jako np. kubełek numer 3 (od -1.5 do -1.0).

    - Ta metoda ma zwracać dla każdej zmiennej opisującej zaobserwowany stan numer kubełka do którego przynależy.
    - Obecnie zwraca cztery razy 1 – zakłada istnienie tylko jednego kubełka na każdą zmienną i do niego wrzuca wszystkie obserwacje.
    - Musimy ją zaimplementować, by zwracała numery kubełków do których zostały przyporządkowane obserwacje – np. 2, 4, 7, 1.
    - Uwaga! Im więcej kubełków tym bardziej „rozrzedzona" jest wiedza (i to wykładniczo!) i tym trudniej cokolwiek nauczyć Q-Learning oparty o słownik. Starajmy się zawsze minimalizować ich liczbę. Dla tego problemu polecam wartości między 2 a 8 kubełków na daną zmienną (sam dobierz ich właściwą liczbę – to jeden z parametrów algorytmu!).
- Metoda pick_action (linia 46) ma za zadanie wybrać kolejną akcję do wykonania zgodnie z punktem 1. opisanego wcześniej schematu algorytmu.
    - Na razie jednakże zamiast tego zawsze wybiera losową akcję ze zbioru dostępnych.
    - Dostępne są dwie akcje: 0 – popchnij wózek w lewo i 1 – popchnij wózek w prawo.
    - Ją też musimy zaimplementować!
- Ostanią do realizacji jest metoda update_knowledge (linia 49).
    - Powinna aktualizować naszą wiedzę (słownik) zgodnie z omówionym schematem.
    - Zamiast tego…chwilowo nie robi zupełnie nic.

Jeżeli zaimplementujemy wszystko prawidłowo, to nagrody zdobywane przez agenta w kolejnych podejściach powinny systematycznie rosnąć. Optymalne rozwiązanie powinno powtarzalnie uzyskiwać nagrodę 500 jednostek

[Zadanie 1] Uzupełnij dziury w kodzie i wykonaj pełną implementację algorytmu Q-Learning zgodnie z omawianym schematem.

Analiza statystyczna wyników

By uzyskać to optymalne rozwiązanie musimy dobrać odpowiedni zestaw parametrów (***γ***, ***ε*** i **α**), tempo i sposób ich zmiany w czasie, oraz granulację dyskretyzacji obserwacji. Niestety uczenie jest procesem mocno stochastycznym i ocena wyników poszczególnych kroków „na oko" może być silnie myląca. Na pewno pomoże w tym wykres zależności nagrody od numeru iteracji. Niestety, jak szybko się okaże, będzie on mocno zaszumiony. Jak sobie z tym poradzić? Stosując następujące dwie techniki.

- **Średnia krocząca.** Zamiast prezentować drgające w górę i w dół wyniki otrzymanych nagród uśrednij je poruszając się po wynikach okienkiem o wybranej szerokości i zwizualizuj informację o tych średnich. Na przykładzie – najpierw średnia z podejść 1 do 10, potem 2 do 11, potem 3 do 12, potem 4 do 13, etc. W ten sposób wyeliminujesz z wizualizacji lokalny szum.
- **Odchylenie standardowe.** Zamiast wykonywać eksperyment uczący raz, wykonaj go kilka razy. Dzięki temu będziesz wiedzieć, czy dany trend (np. nagły wzrost wyników po 15 iteracji) jest czymś powtarzalnym, czy był skutkiem przypadku (algorytm wykonał korzystając z losowej eksploracji kilka bardzo dobrych kroków). Uśrednij wyniki uzyskane przez wszystkie podejście w danym oknie czasowym, policz również ich odchylenie standardowe. Zwizualizuj w następujący sposób: średnie tworzą główną linię trendu (ciągła, pogrubiona, wyraźny

kolor). Punkty średnia+odchylenie oraz średnia-odchylenie tworzą poboczne linie (przerywane, jasne, cienkie) nad i pod linią trendu, tzw. korytarz. Alternatywnym sposobem wizualizacji odchyleń standardowych są tzw. errorbary – byłyby one jednak niepraktyczne na tak gęstym wykresie.

PS. Zamiast wypisywać wyniki na konsolę warto zacząć zapisywać je do pliku tekstowego – ułatwi to późniejszą obróbkę. Do rysowania wykresów można użyć dowolnego lubianego narzędzia.

[Zadanie 2] Wybierz kilka zestawów parametrów (przynajmniej 4) i przetestuj wpływ ich zmian na zachowanie algorytmu. Zwizualizuj i porównaj uzyskane wyniki korzystając z wymienionych wyżej technik.

SARSA

Algorytm SARSA (nie mylić z piosenkarką, skrót od state–action–reward–state–action) jest wariantem Q-Learning korzystającym z delikatnie zmodyfikowanego schematu uczenia, opisane poniższym wzorem.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Zwróć uwagę, że człon opisujący przyszłe zyski nie zawiera teraz maksimum po możliwych akcjach, a rzeczywistą akcję a_{t+1} podjętą w iteracji kolejnej po iteracji t. Ma to dwie konsekwencje.

- Algorytm uwzględnia zagrożenia związane z tym, że wykonywana akcja nie zawsze jest tą optymalną (bo przecież jest szansa na eksperyment). Stany w których losowy wybór akcji wiąże się z potencjalnie dużą karą (potocznie „kroczenie nad brzegiem przepaści” – jedna-dwie nieoptymalne akcje mogą skończyć się tragedią) będą niepożądane.
- Aby zaktualizować wiedzę konieczna jest znajomość zarówno akcji a_t, jak i kolejnej a_{t+1}. W kroku uczenia należy pamiętać je obie (a nie tylko ostatnią).

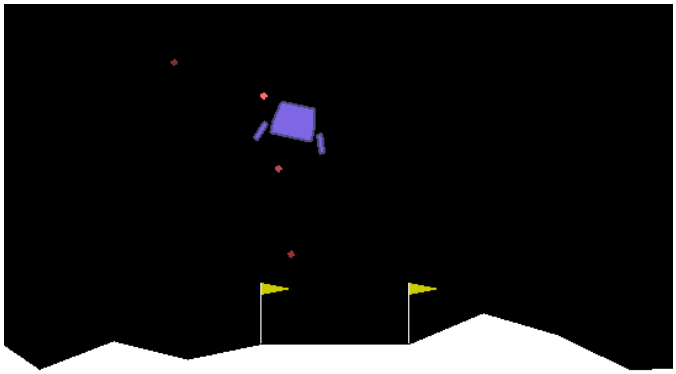
[Zadanie 3] Zmodyfikuj kod, by realizował algorytm SARSA. Porównaj jego działanie (dla wybranej grupy parametrów) z działaniem Q-Learning, używając omówionych wcześniej technik wizualizacji.

Zadanie domowe

Zadania do wykonania na zajęciach zostały rozpisane trochę „na wyrost” – po to by najaktywniejsi (i będący danego dnia w dobrej kondycji umysłowej) mieli co robić w trakcie trwania laboratorium. Pierwszą kwestią do zrobienia w domu jest więc po prostu dokończenie tego, czego nie udało się zrealizować „na żywo” i wykonanie z tego raportu.

[Zadanie 4] Podsumuj pracę nad zadaniami 1-3 i jej rezultaty w formie raportu w pliku PDF. Pamiętaj o tym, by umieścić go jako rozwiązanie na UPEL. Jest on nawet ważniejszy niż kod!

Lądowanie na księżycu



Ostatnim krokiem jest wykorzystanie poznanego algorytmu do próby rozwiązania trudniejszego problemu – sterowania lądownikiem księżycowym. W bibliotece Gym jest on nazwany „LunarLander-v2”. Jego kod można znaleźć na platformie github: https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py (bardzo pomocne w analizie działania symulacji i próbach zrozumienia zwracanego stanu). Zaadaptuj istniejący kod do walki z tym zadaniem i wyląduj (umiarkowanie) bezpiecznie na księżycu!

- Możliwe akcje to: uruchom lewą dyszę sterującą, uruchom prawą dyszę, uruchom dolną dyszę, nie rób niczego.
- Dobierz przedział rozważanych wartości i ziarnistość dyskretyzacji do nowych obserwacji środowiska.
    - W tym celu warto wykonać szybki histogram wartości otrzymywanych w trakcie losowego lotu.
    - Przydaje się też przejrzenie kodu źródłowego problemu.
- Uczenie musi przebiegać znacząco wolniej i dłużej – to nie jest łatwy problem dla tak uproszczonego algorytmu.
- Powinno się jednak udać uzyskać dodatnie nagrody (rzędu 50 jednostek) – a jeżeli nie, to przynajmniej mniejsze kary niż przy losowym zachowaniu.
- Skuteczna procedura ucząca z wykorzystaniem komputera osobistego może być czasochłonna (kilka h).

[Zadanie 5] Jak tym razem spisał się algorytm? Uzupełnij raport o wykresy i wnioski z pracy nad lądownikiem księżycowym.

Maksymalna ocena z zadania domowego wymaga (przynajmniej pobieżnego) wykonania wszystkich pięciu zadań.