Ramię Ośmiornicy

Metody Inteligencji Sztucznej i Obliczeniowej

Sprawozdanie
Marcin Mrugas 122580
Marcin Drzewiecki 122472
15.06.2018 r.

Opis

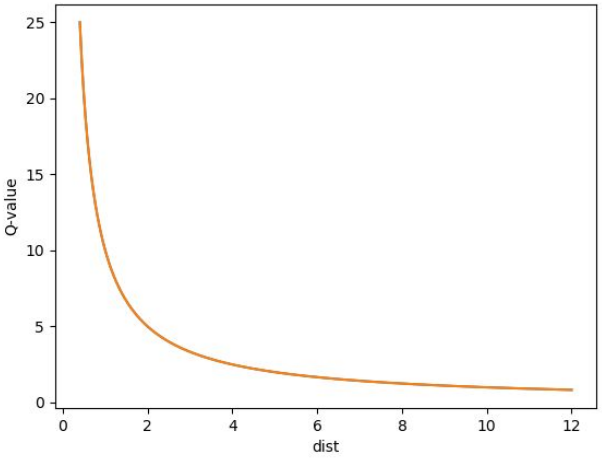
Naszym zadaniem było użycie środowiska symulującego działanie macki ośmiornicy by nauczyła się dotykać wybranego punktu. Zadanie nie było trywialne ze względu na środowisko, którego przestrzeń stanów i akcji była ogromna. Dodatkowo w środowisku zostały odwzorowane siły, które działały na ramię takie jak wyporność, grawitacja czy opory wody. Sprawiło to, że ramię zachowywało się inaczej zależnie od głębokości, ułożenia czy prędkości poruszania.
Wykorzystaliśmy uczenie ze wzmocnieniem, aby nasz agent nauczył się wykonywać poprawne ruchy. Z każdym ruchem starał się maksymalizować funkcję nagrody, która w naszym rozwiązaniu odpowiadała odwrotności odległości końcówki macki od docelowego punktu. Jako element uczący agenta użyliśmy sieci neuronowej. Sieć miała nauczyć się przewidywać wartość funkcji nagrody po wykonaniu wybranej akcji. Agent posiada także pamięć, dzięki której agent nie uczy się tylko pojedynczej akcji, ale jest w stanie doszkolić się także na bazie poprzednich doświadczeń.
Do implementacji zadania został użyty język Python w wersji 3.6 wraz z frameworkiem Keras.

Funkcja nagrody

Naszą funkcją nagrody była zmodyfikowana funkcja odległości:

$$Q(s,a) = \frac{10}{dist}$$

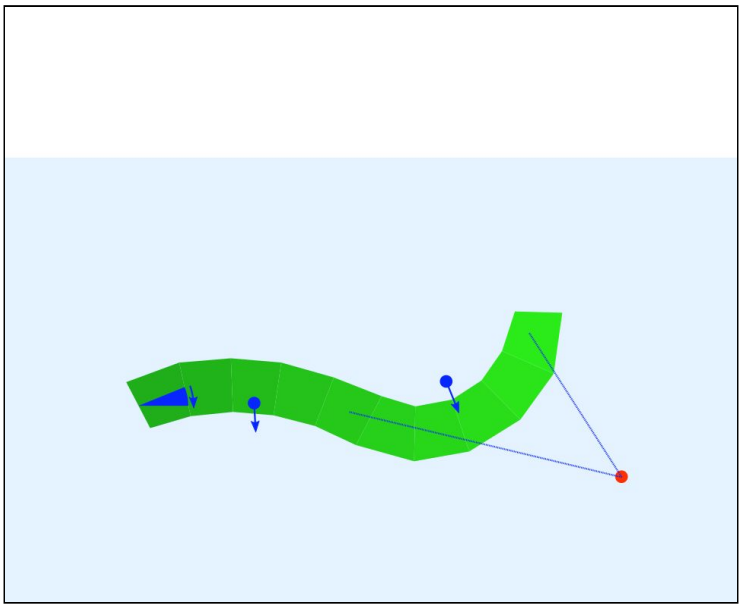
Gdzie *dist* to odległość ostatniego segmentu do punktu docelowego. Gdy akcja prowadziła do dotknięcia kropki funkcja została dodatkowo zwiększona o 10.



Opis stanu

Opis stanu środowiska składał się z 4 liczb opisujących położenie x, y każdego punktu ramienia oraz jego prędkości. Z początkowych 82 liczb opisujących stan środowiska stworzyliśmy 12 głównych cech, które wystarczająco opisują stan, w którym znajduje się macka:

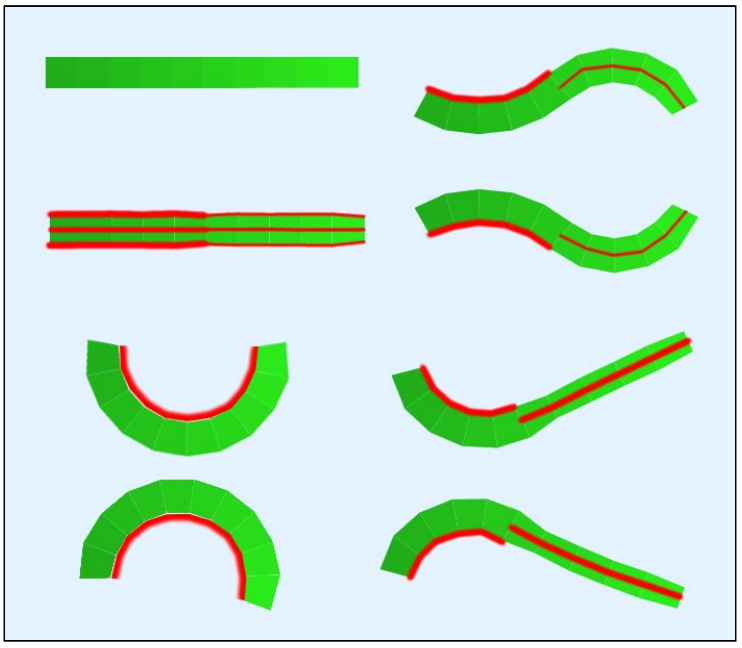
1. kąt zaczepu ramienia w radianach
2. prędkość zmiany kąta zaczepu ramienia
3. środek geometryczny pierwszej połowy segmentów (współrzędna x)
4. środek geometryczny pierwszej połowy segmentów (współrzędna y)
5. środek geometryczny drugiej połowy segmentów (współrzędna x)
6. środek geometryczny drugiej połowy segmentów (współrzędna y)
7. prędkość środka geometrycznego pierwszej połowy segmentów (współrzędna x)
8. prędkość środka geometrycznego pierwszej połowy segmentów (współrzędna y)
9. prędkość środka geometrycznego drugiej połowy segmentów (współrzędna x)
10. prędkość środka geometrycznego drugiej połowy segmentów (współrzędna y)
11. odległość środka ostatniego segmentu do punktu
12. odległość środkowego segmentu do punktu



Akcje

Dozwolone akcje pozwalały na poruszanie mięśniami podłużnymi, dzięki którym segment ramienia wyginał się w górę, bądź w dół oraz mięśni podłużnych pozwalających na wydłużenie segmentu. Każdym mięśniem w segmencie można sterować oddzielnie w zakresie od 0 (co odpowiada całkowitemu odprężeniu) do 1 (co odpowiada maksymalnemu napięciu mięśnia). Zatem przestrzeń akcji była ogromna i musieliśmy ją bardzo uprościć. Zdecydowaliśmy się także spowolnić symulację i zmniejszyć częstotliwość odpytywania agenta poprzez powtarzanie raz przewidzianej akcji 10 razy. Przyspieszyło to także czas uczenia agenta. Z ogromnej przestrzeni akcji wybraliśmy 8 podstawowych ruchów ramienia, które w symulacji manualnej wystarczyły by osiągnąć cel:

1. Nie robienie niczego
2. Napięcie wszystkich mięśni w pierwszej połowie i napięcie do połowy mięśni w drugiej połówce
3. Napięcie górnych mięśni
4. Napięcie dolnych mięśni
5. Napięcie górnych mięśni w pierwszej połowie i napięcie do połowy dolnych mięśni w drugiej połowie
6. Napięcie dolnych mięśni w pierwszej połowie i napięcie do połowy górnych mięśni w drugiej połowie
7. Napięcie górnych mięśni w pierwszej połowie i mięśni poprzecznych w drugiej połowie
8. Napięcie dolnych mięśni w pierwszej połowie i mięśni poprzecznych w drugiej połowie

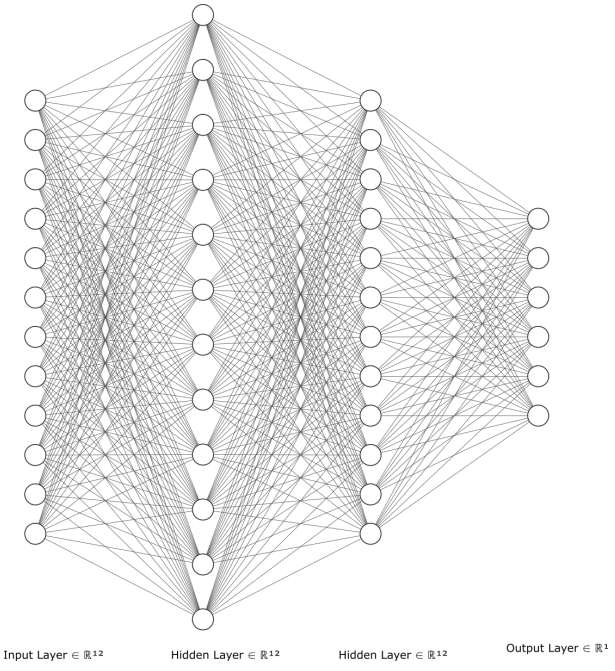


Element uczący

Sieć składa się z:

- warstwy wejściowej przyjmującej 12 liczb zmiennoprzecinkowych,
- dwóch warstw w pełni połączonych z 28 neuronami o funkcji aktywacji ReLu,
- warstwy wyjściową z 6 neuronami o liniowej funkcji aktywacji.

Na jej wejściu został podawany uproszczony opis stanu środowiska, a na każde wyjście odpowiada przewidzianą wartością funkcji nagrody po wykonaniu jednej z sześciu akcji. Przyjęliśmy współczynnik uczenia na poziomie $\eta = 0.001$.



Pamięć

Aby douczać sieć agent posiada pamięć w której pamięta 20000 ostatnich ruchów. Pojedynczy zapis zawiera opis stanu, wykonaną akcję, wartość funkcji nagrody w następnym stanie, opis następnego stanu oraz informację czy ramię dotknęło punktu w następnym stanie.
Pamięć i wagi sieci są zapisywane co 1000 kroków, aby można było przerwać naukę w dowolnym momencie.

Generator problemów

Aby zapewnić możliwość zdobywania nowych doświadczeń ruchy agenta zależały od parametru ε (exploration rate), który decydował o ilości losowych ruchów. Im większy ε, tym agent wykonywał więcej losowych ruchów.

Implementacja

Początkowo agent wykonywał całkowicie losowe ruchy co pozwoliło mu wstępnie nauczyć się poprawnych akcji. Odpowiadał temu parametr eksploracji, który na początku był równy 1 (co odpowiadało całkowicie losowemu zachowaniu) i po każdej nauce był zmniejszany poprzez przemnożenie razy 0.995.
Co 300 kroków agent wybierał losowo 512 próbek z pamięci zawierających:

- s - stan ramienia
- a - wykonaną akcję w stanie s
- s' - stan następujący po stanie s
- R - wartość funkcji nagrody w stanie s'
- d - informację czy w stanie s' został osiągnięty cel

Dla każdej z nich sieć obliczała wektor $\widehat{Q}(s,a)$, czyli przewidywane wartości funkcji nagrody po wykonaniu akcji a w stanie s oraz $max_a\{\widehat{Q}(s',a_t)\}$, czyli przewidywaną wartość funkcji nagrody po wykonaniu najbardziej obiecującej akcji a w stanie s'. Następnie obliczano wartość $Q(s,a)$ wg wzoru Bellmana:

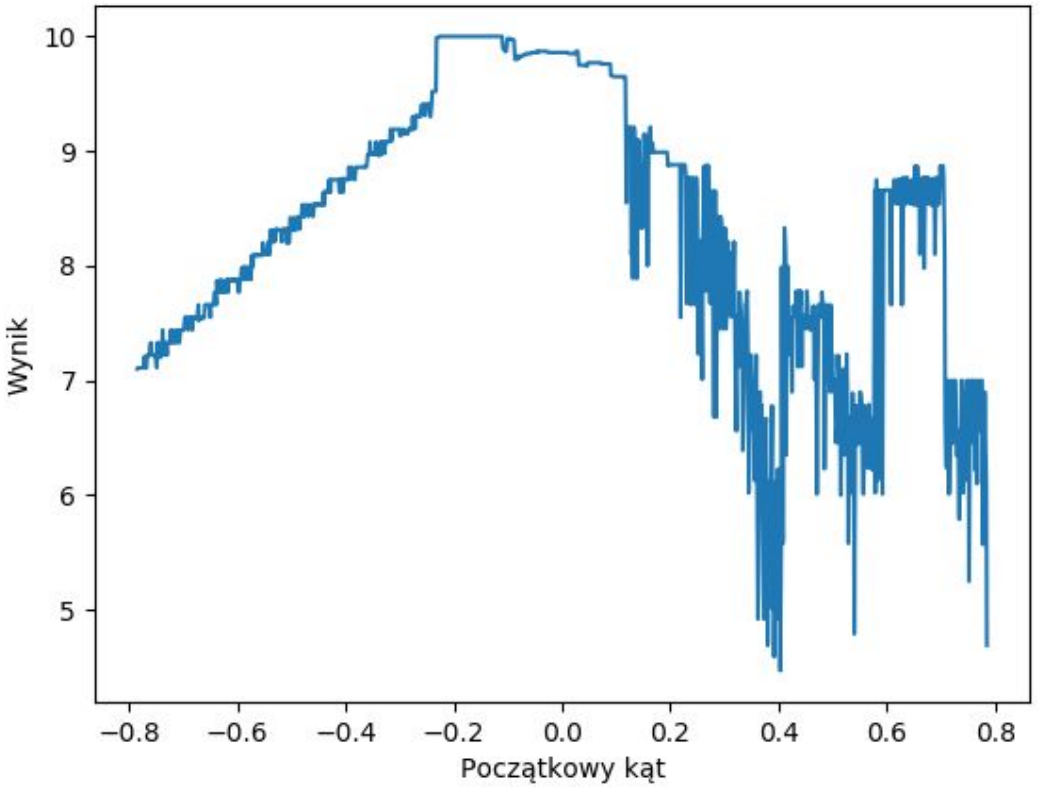
$$Q(s,a) = R + \gamma\, max_a\{\widehat{Q}(s',\ a)\}$$, gdzie współczynnik dyskontowy $\gamma = 0.9$

Co odpowiadało poprawnej wartości funkcji nagrody po wykonaniu akcji a w stanie s. Następnie w wektorze przewidzianych wartości funkcji dla ruchów zmieniono wartość na pozycji odpowiadającym wykonanej akcji a na obliczoną poprawioną wartość $Q(s,a)$. Jeśli w stanie s' został osiągnięty punkt $Q(s,a)$ zostawało zwiększone dodatkowo o 10. Zmieniony wektor używano do nauki sieci, co prowadziło do poprawienia predykcji tej akcji w przyszłości.
Aby poprawnie nauczyć agenta przepisaliśmy skrypt testujący agenta z basha do pythona, aby agent nie wczytywał za każdym uruchomieniem wag sieci oraz historii. Pozwoliło to także na wybór testu w sposób losowy.

Do nauki zostały usunięte testy w których w początkowym stanie ramię już dotykała kropki. Były to testy gdzie ramię była wychylona początkowo od -0.095° do -0.232°.
Na początku agent był uczony na łatwych planszach, które nie wymagały skomplikowanych ruchów. Były to plansze w których początkowy kąt wynosił od -0.076° do -0.254°.Następnie zwiększyliśmy zakres testów do początkowego wychylenia ramienia w zakresie od 0.000° do -0.350°. W kolejnym kroku agent uczył się na wszystkich planszach do pozwoliło jej nauczyć się najtrudniejszych przypadków jak i nie zapomnieć najłatwiejszych.

Wyniki

Poniższy wykres przedstawia wyniki w zależności od początkowego kąta w teście. Można z niego odczytać, że agent dobrze nauczył się poruszać w środowisku i we wszystkich testach udawało mu się osiągnąć cel. Widać także, że gdy ramę było początkowo ustawione od dołu agent umiał poczekać, aż znajdzie się na odpowiedniej głębokości, by na końcu "zaatakować" docelowy punkt. Zaczynając od góry agent wykonuje więcej zamachów i szybciej wydaje mu się, że jest wystarczająco blisko by zakończyć symulację, jednak myli się co wymaga dodatkowych ruchów, w których traci punkty. Widać to w okolicy 0.4°. Wykres z tych testów, gdzie ramię zaczyna od góry ma także charakter bardziej schodkowy, co świadczy o tym, że ramię wykonując odpowiednie wygięcie może nabrać prędkości i w efekcie szybciej dotknąć punktu docelowego. Średni wynik na wszystkich planszach to 8.35935.



Próby

Pierwsza próbą było uruchomienie agenta z niezmienionym stanem wyjściowym, który zawierał 82 zmienne. Agent był w stanie nauczyc sie na najłatwiejszych planszach, ale miał problem z trochę trudniejszymi testami. Wtedy dodatkowo nie ograniczyliśmy zbioru akcji zezwalając na wszystkie $2^6 = 64$ kombinacje ruchów mięśni. Zwiększyliśmy także rozmiar sieci dodając warstwy ukryte, jednak to nie polepszało wyników. Próbowaliśmy także wykorzystać takie zmienne opisujące stan ramienia jak jego wydłużenie, kąty pomiędzy pierwszą i drugą połową segmentów, obrotem fragmentów ramienia względem poziomu, położeniem pierwszej połowy segmentów, średnią głębokością zanurzenia ramienia.

Wnioski

Przez długi czas jednak agent uczył się zbliżać do punktu jednak nie dotykał punktu tylko zastygał w jej okolicy. Było to spowodowane błędem w kodzie przez który gdy ramię dotknęło punktu otrzymywało nagrodę równą odległości od punktu, lecz bez części estymowanej nagrody. Pokazuje to jak dobrze agent stara się zmaksymalizować funkcję nagrody i próbuje znaleźć najlepszy sposób by osiągnąć najlepsze wyniki.
Wykonując to zadanie nauczyliśmy się jak dzięki uczeniu ze wzmocnieniem można rozwiązywać trudne problemy, w których nie jest znana najlepsza strategia i zachowanie.

Podział pracy

Marcin Drzewiecki wydobył odpowiednie cechy i dopasował opis stanu. Marcin Mrugas zajął się implementacją agenta i jego sposobu jego uczenia. Samo nauczanie i dobór parametrów było prowadzone także równolegle, gdyż wymagało kilku prób. Końcowe sprawozdanie powstało wspólnymi siłami autorów.

Bibliografia

- *Octopus Arm Control with RVM-RL*, https://www.cs.colostate.edu/~lemin/octopus.php
- Keon, *Deep Q-Learning with Keras and Gym*, https://keon.io/deep-q-learning/
- Arthur Juliani,*Simple Reinforcement Learning with Tensorflow Part 0: Q-Learning with Tables and Neural Networks*, https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0
- Tambet Matiisen, *Guest Post (Part I): Demystifying Deep Reinforcement Learning*, https://ai.intel.com/demystifying-deep-reinforcement-learning/
- Alex Lenail, http://alexlenail.me/NN-SVG/index.html