Implementacja algorytmu Q-Learning dla gry NIM

Wybrane zagadnienia sztucznej inteligencji. Zadanie 2.

Norbert Ropiak, Maksym Telepchuk

Maj 2020

1 NIM

NIM jest klasycznym przykładem dobrze zanalizowanej od strony matematycznej gry. W tym zadaniu zostały zaimplementowane dwa warianty gry.

1. W przypadku trywialnego wariantu zasady są takie:
   - w szeregu ustawione jest N pionów
   - gracz w swojej turze usuwa liczbę pionów od 1 do K (domyślnie $K = 3$)
   - wygrywa gracz, który usunie ostatni pion z planszy
2. W NIMie z wieloma rzędami zasady gry są następujące:
   - w k szeregach ustawiono po N pionów
   - gracz w swojej turze usuwa dowolną ilość pionów z wybranego rzędu
   - wygrywa gracz, który usunie ostatni pion z planszy

2 Q-Learning

Uczenie ze wzmocnieniem (ang. Reinforcement learning) jest dziedziną algorytmów uczenia maszynowego, celem których jest maksymalizacja nagrody poprzez wykonanie szeregu działań w zmieniającym się środowisku na podstawie reakcji środowiska na te działania.

Q-learning jest przykładem algorytmem uczenia ze wzmocnieniu polegającym się na strategii, który stara się znaleźć najlepszą akcję do podjęcia w obecnym stanie.

Funkcja oceny takich akcji jest uczona na podstawie nagród, które są przypisywane każdemu działaniu. Przy uczeniu najlepszych akcji algorytm poprzez podejmowanie losowych działań również bada te akcje, które są mniej nagradzane.

Q-learning ma na celu poznanie polityki, która maksymalizuje całkowitą nagrodę. „Q” w q-learningu oznacza jakość (ang. quality). Jakość w tym przypadku reprezentuje użyteczność danego działania w zdobywaniu przyszłej nagrody.

2.1 Wzór Bellmana

Żeby ocenić jakość akcji a w stanie s, algorytm Q-Learning liczy q wartości według wzoru Bellmana :

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

gdzie r - jest funkcją nagrody dla akcji a w stanie s, $\max_a Q(s',a)$ jest najwyższą q wartością, którą można uzyskać w stanie s'(który jest stanem, w którym znajdzie się agent po podjęciu działania ze stanu s). Dodatkowo wprowadzono parametr γ, zwykle nazywany współczynnikiem dyskontowym (ang. discount factor), który kontroluje znaczenie nagród długoterminowych w porównaniu do natychmiastowego.

Rekurencyjny charakter wzoru Bellmana umożliwia wsteczną propagację nagród z przyszłych stanów do stanów z przeszłości. Dodatkowo nie jest potrzebne dokładne określenie funkcji Q, ponieważ po długiej interakcji ze środowiskiem te wartości zbiegają do takich, które dają najlepsze wyniki.

2.2 Tabela q-wartości

Q-Learning korzysta z zachłannej strategii - w obecnym stanie wykonaj akcję, która przyniesie najwyższą łączną nagrodę. W przypadku stanu końcowego, q-wartość równa się nagrodzie tego stanu:

$$Q(s_{terminal}, a) = r(s_{terminal}, a)$$

Jednak takie zachłanne podejście jest wykorzystywane w fazie testowej. W procesie uczenia w celu szerszego zbadania innych akcji, wprowadza się pojęcie ϵ-zachłanności : dla zdefiniowanego $\epsilon \in [0,1]$ stosowane jest zachłanne podejście z prawdopodobieństwem $p = 1 - \epsilon$ oraz z prawdopodobieństwem $p = \epsilon$ jest podejmowana losowa decyzja.

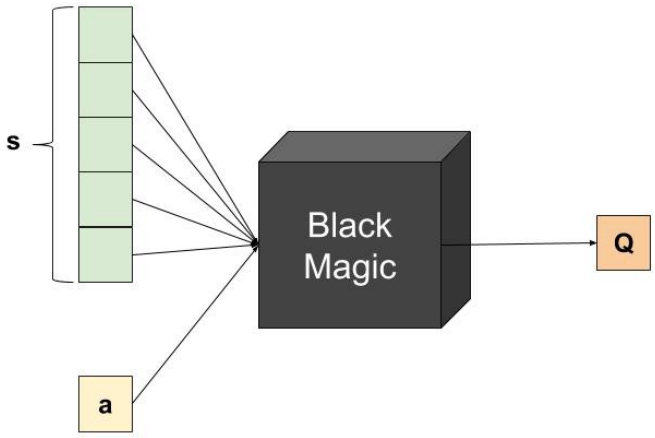
W przypadku gry dwuosobowej, takiej jak NIM, każdy ruch jest zapisywany do lokalnej historii gracza w postaci obiektu [stan gry, podjęta akcja]. W podejściu tabelkowym, wartości q są przechowywane w postaci tabeli. Na końcu gry użytkownik, w zależności od wyniku gry, dostaje nagrodę:

1. $reward = 1$ w przypadku, jeśli gracz wygrał grę
2. $reward = -1$ w przypadku przegranej gry

Zatem po zakończeniu gry, q-wartości są aktualizowane w s następujący sposób:

$$Q(s,a) = (1-\alpha) * Q(s,a) + \alpha * (r(s,a) + \max_a Q(s', a_p)$$

gdzie $r(s,a) = reward$, α - współczynnik uczenia się. Uczenie się odbywa poprzez odbycie wielu gier z innym agentem (agent losowy, zachłanny i inne).



Rysunek 1: Model nadzorowany w Q-Learning

2.3 Q-Learning z użyciem modelu uczenia nadzorowanego

Przy dużej ilości możliwych stanów i akcji, podejście tabelkowe będzie wymagać dużych zasobów pamięciowych. Rozwiązaniem może być model, który jest uczony na podstawie historii ruchów wraz z przypisanym do nich nagrody otrzymanej na końcu gry (rys. 1). Taki model będzie tylko aproksymował q-wartości używając parametrów uczenia się θ i zapisuje się jako $Q(s, a; \theta)$. Uczenie modelu polega na zmniejszeniu kosztu funkcji straty, np. średniej różnicy kwadratowej pomiędzy predykcją tego modelu a nagrodą, przypisaną do wykonanego ruchu

$$\underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} (Q(s, a; \theta) - r(s, a))^2$$

.

3 Użyty model uczenia nadzorowanego - sieć neuronowa

Do przybliżania wyniku meczu NIM została użyta sieć neuronowa nauczona przy użyciu zbioru historii gier. Zdecydowano się przetestować 2 warianty gry NIM:

1. 1 wiersz z 20 pionami, w której gracz w swojej turze może usunąć od 1 do 3 pionów
2. 3 wiersze z 10 pionami, w której gracz w swojej turze może dowolną ilość pionów z jednego wybranego rzędu

Wejściem do sieci jest 5 lub 3 elementowy wektor, w zależności od wariantu NIM. Pierwszymi elementami (dla 1 wariantu - 1 element, dla drugiego - 3 elementy) jest stan gry, który jest opisywany ilością pionów w danym wierszu. Dla przykładu stan początkowy dla pierwszego i drugiego wariantu może być opisany odpowiednio wektorem [20] lub $[10, 10, 10]$. Kolejnymi 2 elemantami jest ruch gracza opisany jednakowo dla obu wariantów przez wiersz z którego usuwane są piony oraz liczba pionów do usunięcia. Przykładowym ruchem dla stanu początkowego może być [1, 3].

Wyjściem tego modelu jest liczba z przedziału $[-1, 1]$. Wartość 1 oznacza wygraną gracza, który aktualnie podejmuje ruch, a wartość -1 oznacza wygraną przeciwnika.

3.1 Zebranie historii

Dla wyuczenia modelu nadzorowanego potrzebne dane w postaci historii ruchów wraz z przypisanymi do nich nagrodą końcową. W przypadku NIMa z 3 rzędami, jest to wektor $\{s_1, s_2, s_3, row, count\}$, gdzie s_i - liczba pionków w i-tym rzędzie, row - numer rzędu z którego są zabierane pionki, $count$ - liczba zabieranych pionków. Wartością docelową y w tym przypadku jest nagroda uzyskana na końcu gry graczem, który wykonał dany ruch.

W celu zebrania takich danych, 2 agenty Q-Learning z podejściem tabelkowym zagrały 5000 gier przeciwko siebie, żeby wytrenować swoje q-wartości. Później, każdy z tych agentów zagrał z agentem losowym po 225000 gier, startując z tej pozycji gracza, na której występował w procesie uczenia się. Po każdej grze historia agentów Q-Learning została zapisana do jednego pliku, z którego później został nauczony model nadzorowany.

3.2 Architektura sieci neuronowej

Ze względu na reprezentację wejścia i małe skomplikowanie problemu zdecydowano się na sieć neuronową z warstwami gęstymi (w pełni połączonymi).

Wypróbowane architektury:

1. 3 warstwy gęste po 64 neuronów
   - warstwa gęsta 5(3) $\longrightarrow$ 64 - funkcja aktywacji ReLU
   - warstwa gęsta 64 $\longrightarrow$ 64 - funkcja aktywacji ReLU
   - warstwa gęsta 64 $\longrightarrow$ 1 - funkcja aktywacji $2\tanh$
2. 4 warstwy gęste z rosnącą ilością neuronów
   - warstwa gęsta 5(3) $\longrightarrow$ 64 - funkcja aktywacji ReLU
   - warstwa gęsta 64 $\longrightarrow$ 128 - funkcja aktywacji ReLU
   - Dropout(0.25)
   - warstwa gęsta 128 $\longrightarrow$ 128 - funkcja aktywacji ReLU
   - Dropout(0.25)
   - warstwa gęsta 128 $\longrightarrow$ 1 - funkcja aktywacji $2\tanh$
3. 3 warstwy z dropout
   - warstwa gęsta 5(3) $\longrightarrow$ 64 - funkcja aktywacji ReLU
   - warstwa gęsta 64 $\longrightarrow$ 64 - funkcja aktywacji ReLU
   - Dropout(0.15)
   - warstwa gęsta 64 $\longrightarrow$ 1 - funkcja aktywacji $2\tanh$

3.2.1 Funkcja straty

Funkcja straty użyta w procesie uczenia to błąd średnio kwadratowy

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - h_\Theta(x_i))^2$$

3.3 Porównanie parametrów uczenia

W procesie uczenia został wykorzystany standardowy optymalizator SGD (Stochastic Gradient Descent). W zależności od parametrów uzyskano wyniki:

architektura	epochs	α	momentum	MSE na zbiorze walidacyjnym	inference time
1	2	3×10^{-3}	0.9	0.1514	0.00033
2	10	3×10^{-3}	0.9	0.1164	0.000384
3	10	3×10^{-3}	0.9	0.1419	0.00035

3.4 MCTS z użyciem modelu

Zakładając, że model, który został użyty w Q-Learning przewiduje skuteczność danego ruchu, można go również użyć w algorytmie MCTS. Na etapie playoutu, zamiast tego, żeby losowo przeprowadzać grę do końca i patrzyć na wyniki, MCTS będzie wyliczał aproksymację końcowego wyniku za pomocą nauczonego modelu. Wtedy w węzłach zamiast liczby odwiedzeń będzie zapisywana suma nagród.

4 Porównanie wyników

4.1 Agent zachłanny

Zachłanny agent ma następującą politykę wyboru akcji:

1. Jeśli jest tylko 2 nie puste rzędy $\rightarrow$ weź jeden pion z rzędu, w którym jest najwięcej pionów
2. Inaczej $\rightarrow$ weź wszystkie piony z jednego z niepustych rzędów.

Z takim podejściem algorytm gra szybko sprowadzi się do gry na dwóch rzędach. Agent zatem tym będzie zabierał po jednym pionie, żeby doczekać się sytuacji, gdy zostanie się jeden rząd, żeby zabrać wszystkie piony i wygrać grę.

4.2 Standardowy MCTS vs. Random Agent

Liczba rzędów : 1
Liczba pionów : 30
Czas myślenia dla MCTS : 50 ms
Liczba wygranych dla agenta losowego : 1
Liczba wygranych dla MCTS : 99

4.3 Q-MCTS vs. Random Agent

Liczba rzędów : 1
Liczba pionów : 30
Czas myślenia dla Q-MCTS : 50 ms
Liczba wygranych dla agenta losowego : 0
Liczba wygranych dla Q-MCTS : 100

4.4 Q-MCTS vs. MCTS

4.4.1

Liczba rzędów : 1
Liczba pionów : 30
Czas myślenia dla obu algorytmów : 50 ms
Liczba wygranych dla MCTS : 3
Liczba wygranych dla Q-MCTS : 97

4.4.2

Liczba rzędów : 1
Liczba pionów : 30
Czas myślenia dla MCTS : 50 ms
Czas myślenia dla Q-MCTS : 10 ms
Liczba wygranych dla MCTS : 21
Liczba wygranych dla Q-MCTS : 79

4.4.3

Liczba rzędów : 1
Liczba pionów : 30
Czas myślenia dla MCTS : 50 ms
Czas myślenia dla Q-MCTS : 10 ms
Liczba wygranych dla MCTS : 21
Liczba wygranych dla Q-MCTS : 79

4.4.4

Liczba rzędów : 1
Liczba pionów : 30
Czas myślenia dla MCTS : 100 ms
Czas myślenia dla Q-MCTS : 10 ms
Liczba wygranych dla MCTS : 41
Liczba wygranych dla Q-MCTS : 59

4.4.5

Liczba rzędów : 3
Liczba pionów : po 10
Czas myślenia dla MCTS : 50 ms
Czas myślenia dla Q-MCTS : 50 ms
Liczba wygranych dla MCTS : 36
Liczba wygranych dla Q-MCTS : 64

4.5 MCTS vs. Q-MCTS

4.5.1

Liczba rzędów : 3
Liczba pionów : po 10
Czas myślenia dla MCTS : 50 ms
Czas myślenia dla Q-MCTS : 50 ms
Liczba wygranych dla MCTS : 23
Liczba wygranych dla Q-MCTS : 77

4.6 Q-MCTS vs. Greedy

4.6.1

Liczba rzędów : 3
Liczba pionów : po 10
Czas myślenia dla Q-MCTS : 50 ms
Liczba wygranych dla Greedy : 0
Liczba wygranych dla Q-MCTS : 100

4.7 Supervised model vs Q-MCTS

4.7.1

Liczba rzędów : 3
Liczba pionów : po 10
Czas myślenia dla Q-MCTS : 500 ms
Liczba wygranych dla Supervised model : 14
Liczba wygranych dla Q-MCTS : 86

4.7.2

Liczba rzędów : 3
Liczba pionów : po 10
Czas myślenia dla Q-MCTS : 50 ms
Liczba wygranych dla Supervised model : 93
Liczba wygranych dla Q-MCTS : 7

4.8 Q-MCTS vs Supervised model

4.8.1

Liczba rzędów : 3
Liczba pionów : po 10
Czas myślenia dla Q-MCTS : 500 ms
Liczba wygranych dla Supervised model : 22
Liczba wygranych dla Q-MCTS : 78

4.8.2

Liczba rzędów : 3
Liczba pionów : po 10
Czas myślenia dla Q-MCTS : 50 ms
Liczba wygranych dla Supervised model : 97
Liczba wygranych dla Q-MCTS : 3

4.9 Q-Table vs Q-MCTS

Q-Table uczony przez 1000 iteracji grając przeciwko sobie.

4.9.1

Liczba rzędów : 3
Liczba pionów : po 10
Czas myślenia dla Q-MCTS : 50 ms
Liczba wygranych dla Q-Table : 0
Liczba wygranych dla Q-MCTS : 100

4.10 Q-MCTS vs Q-Table

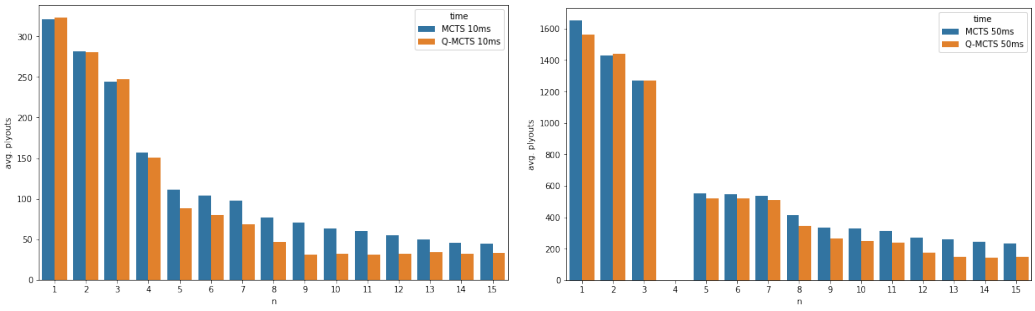
4.10.1

Liczba rzędów : 3
Liczba pionów : po 10
Czas myślenia dla Q-MCTS : 50 ms
Liczba wygranych dla Q-MCTS : 100
Liczba wygranych dla Q-Table : 0

4.11 Porównanie wydajności czasowej

W tabeli 1 porównano średnią ilość wykonanych playoutów dla algorytmu MCTS oraz Q-MCTS. W tabeli jest wypisana średnia ilość playoutów, które są wykonywane w NIMie z 1 rzędem przy n pionkach i czasie wykonywania 50 milisekund.

Na rys. 2 porównano wydajność czasową algorytmów przy ograniczonym czasie myślenia.



(a) Czas myślenia - 10ms

(b) Czas myślenia - 50ms

Rysunek 2: Porównanie wydajności czasowej MCTS i Q-MCTS

n	Q-MCTS avg. playouts	MCTS avg. playouts
30	149	122
28	150	132
27	152	135
26	147	143
25	154	145
24	149	152
23	149	155
22	148	164
21	149	171
20	149	184
19	149	184
18	145	196
17	151	206
16	150	218
15	148	231
14	144	244
13	147	258
12	173	271
11	240	313
10	248	329
9	267	334
8	344	412
7	511	535
6	519	544
5	521	551
3	1271	1271
2	1437	1428
1	1559	1651

Tablica 1: Wydajność czasowa dla Q-MCTS i MCTS z czasem myślenia 50ms

5 Wnioski

Wszystkie zaimplementowane algorytmy bardzo dobrze sobie radzą z agentem losowym i prostym agentem zachłannym.

Prosta architektura z 3 warstwami gęstymi wystarczyła, aby osiągnąć satysfakcjonujące wyniki.

Połączenie algorytmu MCTS i nadzorowanego modelu dało imponujące wyniki. Przy równym czasie myślenia zmodyfikowany MCTS (Q-MCTS) wygrywa w 97% przypadków. Żeby wyrównać szansy potrzebne jest zwiększenie czasu myślenia w 10 razy dla MCTS.

Średnia ilość playoutów dla MCTS była nieco wyższa niż dla Q-MCTS przy czasie myślenia 10 i 50 milisekund. To może być związane z tym, że NIM jest bardzo prostą grą i rozgrywanie playoutów jest nawet szybsze od użycia modelu sieci neuronowej.

Q-MCTS wygrał wszystkie gry przeciwko Q-Table. Może być to związane z tym, że w przypadku gry NIM q-wartości są niskie na początku i wysokie na końcu gry, lecz Q-MCTS uczy się od razu na wynikach końcowych gry, czyli „widzi" całą grę do przodu.