Projekt ze sztucznej inteligencji w grach

Sztuczna inteligencja do TORCS

Jarosław Dzikowski, Bartłomiej Najdecki

Wrocław, 1 lutego 2017

1 Wstęp

2 Metody sztucznej inteligencji

2.1 Uniwersalny Q-Learning

Celem tej metody jest nauczenie komputera wydajnego jeżdżenia samochodem na dowolnej trasie.

2.1.1 Q-Learning oraz Decyzyjne Procesy Markova

Q-Learning jest metodą uczenia się ze wzmocnieniem służącą do wyznaczania optymalnej polityki w dowolnym skończonym Decyzyjnym Procesie Markova (Markov Decision Process). Każdy MDP składa się z pięciu części: zbioru stanów S, zbioru akcji A, które można podejmować w stanach, rozkładu prawdopodobieństwa $P_a(s, s')$ oznaczającego jaka jest szansa, że po wykonaniu akcji a w stanie s znajdziemy się w stanie s', funkcji $R_a(s, s')$ oznaczającej nagrodę za wykonanie akcji a w stanie s i znalezieniu się w stanie s', oraz współczynnika dyskontowania $\gamma \in [0, 1]$ reprezentującego ważność przyszłych nagród. Rozwiązaniem MDP jest optymalna polityka $\pi' : S \mapsto A$ wyboru akcji maksymalizująca sumę nagród częściowych (Nagród uzyskiwanych w trakcie wędrówki po przestrzeni stanów). $\pi'(s)$ oznacza optymalną akcję, którą należy wykonać w stanie s.

W wielu problemach, które można reprezentować jako MDP, rozkład prawdopodobieństwa jest nieznany i często niemożliwy do obliczenia. Właściwością metody Q-Learning jest to, że nie wymaga ona znajomości rozkładu prawdopodobieństwa $P_a(s, s')$. Q-Learning polega na wyznaczeniu funkcji $Q : S \times A \mapsto \mathbb{R}$, gdzie $Q(s, a)$ reprezentuje użyteczność wykonania akcji a w stanie s. W Q-Learningu zaczynamy z użytecznościami równymi zero, następnie rozpoczynamy wędrówkę po przestrzeni stanów ze stanu początkowego s_{init} wykonując akcje według pewnej polityki π. Często wykorzystywana w Q-Learningu jest polityka ϵ-greedy, w której z prawdopodobieństwem ϵ wybierana jest losowa akcja, a z prawdopodobieństwem $1 - \epsilon$ akcja z największą użytecznością, tj.

$\text{argmax}_{a \in A} Q(s, a)$. Z każdym przejściem między stanami wiąże się tymczasowa nagroda - ponieważ dążymy do odnalezienia polityki maksymalizującej sumę nagród, musimy uwzględnić otrzymaną nagrodę w użyteczności wykonanej akcji w stanie, z którego przyszliśmy. Funkcję Q aktualizuje się następująco:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R_a(s, s') + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)), \tag{1}$$

gdzie α jest współczynnikiem nauki, s' jest stanem, w którym się znaleźliśmy po wykonaniu akcji a w stanie s, a γ jest współczynnikiem dyskontowania znanym nam z procesów Markova. Współczynnik nauki $\alpha \in (0, 1)$ odpowiada za to jak wielkie znaczenie na naszą funkcję użyteczności ma jeden „eksperyment" wykonania akcji a w stanie s. Współczynnik dyskontowania γ opisuje jak bardzo interesują nas przyszłe nagrody: im mniejszy współczynnik, tym bardziej interesują nas tymczasowe nagrody - wędrujemy po przestrzeni stanów zachłannie.

W taki oto sposób wykonujemy wiele przebiegów uczących: każdy przebieg zaczyna w stanie s_{init}, wędruje po przestrzeni stanów S, a kiedy trafiamy na stan terminalny, to uruchamiamy kolejny przebieg uczący. Im więcej przebiegów uczących zostanie wykonanych, tym lepsza będzie wyuczona funkcja użyteczności Q.

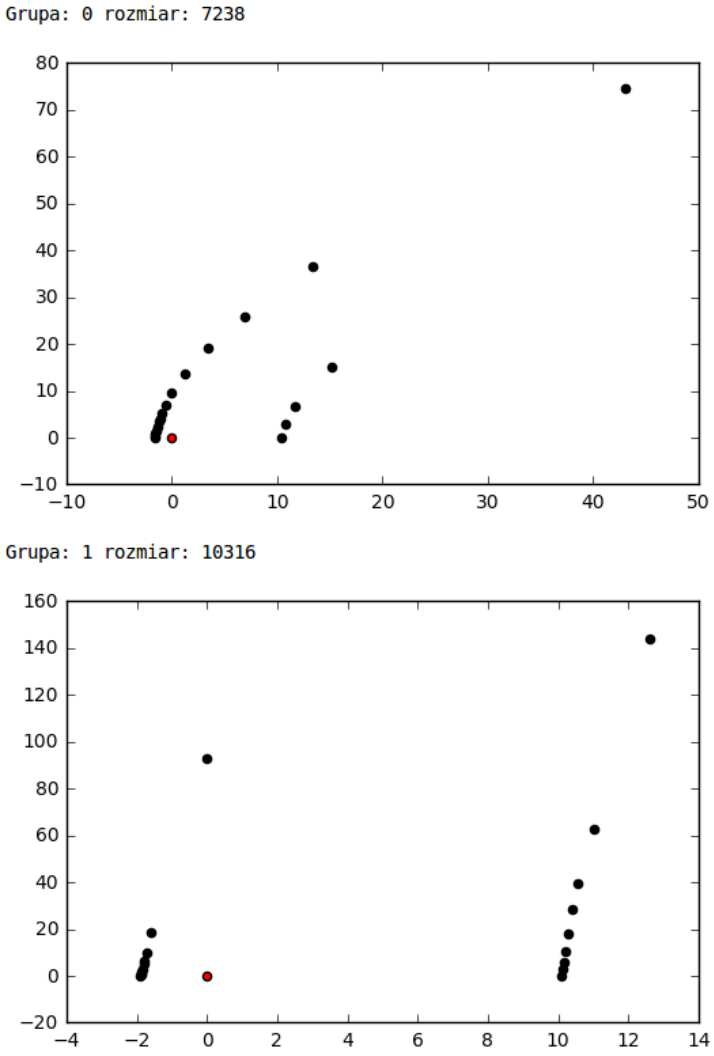
2.1.2 Model Q-Learningu w TORCS

Przestrzeń stanów: Aby móc przeprowadzić uczenie za pomocą Q-Learningu potrzebne jest umodelowanie przestrzeni stanów oraz akcji. Intuicyjnie, podjęta przez komputer akcja powinna zależeć od czynników takich jak kształt odcinka toru przed samochodem oraz prędkość z jaką porusza się samochód: gdy jedziemy po prostej, możemy wcisnąć gaz do dechy, a kiedy widzimy, że zbliża się zakręt, zwalniamy (Pod warunkiem, że jedziemy za szybko). W takim wypadku możemy reprezentować stan jako para (kształt odcinka toru, prędkość). O ile prędkość jest skalarem, to należy zastanowić się jak reprezentować kształt odcinka toru bezpośrednio przed nami. Można tutaj wykorzystać informacje, które zapewnia interfejs botów: zestaw sensorów toru. Mamy do dyspozycji 19 sensorów umieszczonych symetrycznie pod kątami z przedziału [-90 deg, 90 deg]. Każdy sensor daje nam informację jak daleko znajduje się krawędź toru pod danym kątem, lecz maksymalny zasięg sensora wynosi 200m. Mamy zatem wektor dziewiętnastu liczb rzeczywistych, które mogą posłużyć nam za reprezentacje kształtu odcinka toru.

Pojawia się problem nieskończoności oraz wymiarowości przestrzeni stanów: $s \in \mathbb{R}^{19} \times \mathbb{R}^1 \sim \mathbb{R}^{20}$. Przestrzeń stanów należy zamienić w przestrzeń skończoną oraz zmniejszyć wymiar stanu. Jeśli chodzi o prędkość, to łatwo ją zdyskretyzować: przestrzeń $\mathbb{R}$ możemy podzielić na przedziały, np. $\{(-\infty), 0), [0, 30), [30, 80), (80, +\infty)]\}$, a następnie myśleć tylko o numerze przedziału prędkości, w którym się znajdujemy.

Co do redukcji wymiarowości oraz dyskretyzacji przestrzeni kształtów odcinków toru, to proponujemy nieco bardziej skomplikowane podejście. Zebrawszy uprzednio wielki zbiór danych różnych odczytów dziewiętnastu sensorów, grupujemy odczyty (wektory w $\mathbb{R}^{19}$) za pomocą algorytmu K-Means, który wyznacza

centroidy grup - każdy centroid jest wektorem reprezentującym swoją grupę. Zestaw danych możemy pogrupować na dowolną liczbę grup. Aby wyznaczyć do której grupy należy aktualny stan sensorów samochodu, znajdujemy najbliższy w sensie metryki euklidesowej centroid, a następnie zapamiętujemy jego identyfikator, czyli de facto identyfikator grupy, do której przynależy aktualny stan sensorów.



Rysunek 1: Dwa przykładowe centroidy reprezentujące dwie grupy kształtów odcinków toru przy grupowaniu algorytmem K-Means (K = 120). Czarne kropki reprezentują odczyty z dziewiętnastu sensorów, czerwona kropka przedstawia pozycję samochodu, który ustawiony jest zawsze wzdłuż osi Y.

Po zastosowaniu powyższych technik przestrzeń stanów jest reprezentowana jako para liczb naturalnych ($\mathbb{N} \times \mathbb{N}$), gdzie pierwszą współrzędną jest identyfikator grupy, do której należy kształt odcinka toru przed samochodem, a drugą numer przedziału prędkości, do którego należy aktualna prędkość samochodu.

Przestrzeń akcji: Dla uproszczenia problemu, odpuścimy sobie manipulację sprzęgłem oraz będziemy używać automatycznej skrzyni biegów. Sterujemy zatem samochodem używając tylko kierownicy (Wartość skrętu z przedziału $[-1, 1]$), pedałem gazu (Wartość z $[0, 1]$) oraz hamulcem (Wartość z $[0, 1]$). Dodatkowo założymy, że nigdy nie gazujemy oraz hamujemy jednocześnie. Jeśli chodzi o akcje skrętu, to w naszym modelu wystarczy wybrać sobie kilka konkretnych skrętów (Np. 0, -0.1, +0.1, -0.5, +0.5). Zbiór akcji gazowania/hamowania możemy ograniczyć do wartości z przedziału $[-1, 1]$, gdzie -1 oznacza pełne wciśnięcie hamulca, a +1 oznacza pełne wciśnięcie gazu. W naszym modelu wyróżniamy trzy akcje gazowania/hamowania: -1, 0, 1.

Funkcja nagrody: Ostatnią rzeczą potrzebną do umodelowania środowiska do Q-Learningu jest funkcja nagrody, na podstawie której nasz bot będzie się uczył, które akcje są opłacalne, a które nie. Żeby wyprowadzić taką funkcję, należy uprzednio ustalić jakie efekty chcemy osiągnąć.

Przede wszystkim, nie chcemy by samochód wyjechał poza tor - funkcja nagrody powinna surowo karać takie sytuacje zwracając dużą ujemną nagrodę. Stan, w którym samochód wyjeżdża poza tor jest terminalny - wracamy wtedy na linię startu i rozpoczynamy kolejny przebieg uczący. Żeby uniknąć negatywnej nagrody, uczący się bot mógłby po prostu stać miejscu i zadowolić się zerową nagrodą. Aby tego uniknąć wprowadzamy kolejną surową ujemną nagrodę w przypadku, kiedy między przejściem ze stanu s do stanu s' przejechany dystans nie przekroczył progu akceptowalnej przebytej drogi. Chcemy, by nasz bot jeździł wydajnie - im szybciej, tym lepiej, dlatego wprowadzamy dla zachęty dodatnią nagrodę, którą jest przebyta odległość między stanem s a staniem s'. Im większa przebyta odległość, tym większa prędkość samochodu, tym większa nagroda.

Dobranie wartości kary za wyjechanie poza tor, wartości kary za „stanie w miejscu" oraz ustalenie progu minimalnej drogi, którą należy przebyć między dwoma stanami jest zadaniem użytkownika.

2.1.3 Jazda z wyuczoną polityką

Kiedy już zdecydujemy, że komputer nauczył się jeździć wystarczająco dobrze, musimy wykorzystać utworzoną funkcję użyteczności Q do konstrukcji polityki optymalnej π'. Z tego powodu zaimplementowany został bot, który wczytuje z pliku wyuczoną funkcję Q oraz będąć w danym stanie wykonuję akcję, która ma największą użyteczność, tj. $\text{argmax}_{a \in A} Q(s, a)$.

Niestety, jak się okazuje, polityka π' nie zawsze chroni nas od wyjechania poza tor. W celu powrotu na trasę zostałą wykorzystana procedura po-

wrotu zaimplementowana w prostym, naiwnym bocie domyślnie dołączanym do TORCS'a.

Ciekawym podejściem do ostatecznej polityki jazdy jest skonstruowanie rozmytej polityki jazdy π_{rozm}. W odróżnieniu od zachłannej polityki π', polityka π_{rozm} bierze K najbliższych stanów oraz konstruuje finalną akcję jako średnią ważoną z najlepszych akcji w K najbliższych stanach. Im bliższy stan, tym większą wagę ma jego akcja. Najbliższe stany można wybierać względem grup kształtów odcinków trasy, do których przynależy nasz aktualny odczyt z sensorów samochodu, lub względem przedziałów prędkości. Polityka, która bierze średnią ważoną najlepszych akcji wśrod K stanów o najbliższym kształcie odcinka toru oraz uwzględnia najlepsze akcje nie tylko w przedziale prędkości, w którym znajduje się samochód, ale także w sąsiadujących przedziałach, nazywana będzie polityką $\pi_{rozm+speed}$.

2.1.4 Testy

Ponieważ ticki serwera gry zdarzają się 100 razy na sekundę, akcje wybierane w learningu są stosowane przez 10 kolejnych ticków, a aktualizacja funkcji użyteczności następuje podobnie tylko raz na 10 ticków. Powodem ku temu jest fakt, że stan, w którym znajduje się samochód nie zmieni się znacznie w przeciągu jednej setnej sekundy, przez co będziemy często aktualizować użyteczność jakiejś akcji w stanie s korzystając z użyteczności wszystkich akcji w s.

Pierwsze podejście Przestrzeń stanów zawiera 120 grup kształtów odcinków toru oraz 7 przedziałów prędkości: 0-10, 10-30, 30-80, 80-110, 110-150, 150-200, 200-∞. Przestrzeń akcji skrętów zawiera 5 możliwości: skręt -0.5, -0.1, 0, +0.1, +0.5, a akcje gazowania / hamulców są takie same jak opisano wcześniej. Dodatkowo współczynnik nauki $\alpha = 0.1$, współczynnik dyskontowania $\gamma = 0.99$, a szansa na podjęcie losowego ruchu w polityce ϵ-greedy wynosi $\epsilon = 0.05$. Kara za opuszczenie toru wynosi -10, kara za bezruch wynosi -20, a próg bezruchu wynosi 0.1m.

Przy tej konfiguracji komputer uczył się przez około dwa tygodnie. Wybrana została trasa Wheel 2 zapewniająca wiele zakrętów różnych trudności w celu wyuczenia się odpowiedniego zachowania w różnych sytuacjach na drodzę. Początkowo uczący się komputer stoi w miejscu chaotycznie wykonując losowe akcje w celu odkrycia, które akcje dają jakieś ciekawe efekty (Pozytywną nagrodę). Kolejna faza uczenia, w którą wchodzi komputer jest przejeżdżanie bardzo wolno coraz to większej części toru wyścigowego. Już kilka godzin od rozpoczęcia nauki, komputer potrafi przejechać okrążenie bez wypadania poza tor w czasie około 500 sekund, czyli bardzo wolno. Dla porównania człowiek potrafi przejechać tę samą trasę w około 140 sekund bez uprzedniego treningu. W miarę upływu czasu komputer uczył się jeździć co raz szybciej, o czym świadczyła liczba ticków serwera między początkiem każdego przebiegu uczącego a wypadnięciem samochodu poza tor. W efekcie bot przejeżdżał całe okrążenie coraz rzadziej, lecz ze zmniejszającym się czasem. Uzyskany rekord po dwóch tygodniach nauki wynosił około 215 sekund.

Testy odbyły się na dwóch torach: na torze Wheel 2, na którym bot uczył się jeździć, oraz na torze Forza (Całkiem prosty tor umożliwiający szybką jazdę). Na torze Forza bot jeżdżący z użyciem polityki optymalnej π' uzyskał najlepszy czas 3:07, czyli 187 sekund. Bot jeżdżący z użyciem rozmytej polityki biorącej pod uwagę 3 najbliższe stany uzyskał dużo lepszy czas - 2:30, czyli 150 sekund. Bot z rozmytą polityką uwzględniającą także najlepsze akcje ze stanów z sąsiadującymi przedziałami prędkości osiągnął czas 2:33, czyli podobny do zwykłej rozmytej polityki. Dla porównania autorom projektu udało się osiągnąć czas 1:57 za pierwszą próbą.

Na torze Wheel 2, na którym bot uczył się jeździć, czasy prezentują się następująco: π': 3:08, π_{rozm}: 3:03, $\pi_{rozm+speed}$: N/A (Bot utyka podczas zakrętu o 180 stopni i nie kończy wyścigu), a człowiek przejeżdża trasę za pierwszym razem w 2:13. Warto wspomnieć, że bot posługujący się rozmytą polityką w pewnym momencie, na zakręcie o 180 stopni, wypada z trasy i uderza w bandę, co sprawia, że bot traci sporo czasu na powrót na trasę.

Ciekawą obserwacją jest to, że w polityce π' często można dostrzec sytuację, gdzie bot jedzie z równą prędkością, która jest na granicy dwóch przedziałów prędkości. Takie zachowanie wynika z faktu, iż bot nauczył się, że w przedziale prędkości k najbardziej opłaca się wciskać pedał gazu, a w przedziale prędkości $k+1$ najbardziej opłaca się wciskać hamulec. W efekcie bot jedzie ze stałą, niską prędkością, a mógłby spokojnie jechać szybciej. Polityki z rozmyciami nie mają takiego problemu, ponieważ liczą średnie ważone akcji, co sprawia, że we wspomnianych sytuacjach bot cały czas przyspiesza, podczas gdy bot z polityką π' jechałby ze stałą prędkością.

	π'	π_{rozm}	$\pi_{rozm+speed}$	**Człowiek**
Forza	3:07	2:30	2:33	1:57
Wheel 2	3:08	3:03	—	2:13

Tablica 1: Wyniki Q-Learningu w porównaniu z wynikami ludzi.

Drugie podejście W drugim podejściu poszerzona została zarówno przestrzeń stanów jak i akcji - mamy 13 przedziałów prędkości: 0-10, 10-30, 30-50 itd., oraz 11 możliwości sterowania kierownicą: skręty -0.4, -0.3, -0.2, -0.1, -0.05, 0, 0.05, 0.1, 0.2, 0.3, 0.4, itd. Kara za wyjechanie poza trasę wynosi -10, kara za bezruch -20, a próg bezruchu wynosi 2m. Taki wysoki próg bezruchu powinien zmusić komputer do jazdy z całkiem wysoką prędkością minimalną.

Po ponad tygodniu nauki najlepszy czas, jaki osiągnął komputer w jakimś przebiegu uczącym wynosił 247 sekund, jest to wynik dalece oddalony od wyniku z poprzedniego podejścia. Aby móc przedstawić sensowne wyniki, wymagana byłaby dalsza nauka. Ponieważ mamy dużo szersze przestrzenie stanów i akcji, to czas rzeczywisty nauki wynosiłby przynajmniej 3 tygodnie. Ze względów czasowych nie udało się zbadać wyników po tym czasie.

Literatura

[1] *Q-Learning*, https://en.wikipedia.org/wiki/Q-learning