Implementacja algorytmów uczenia maszynowego do samodzielnego przechodzenia gier wideo

Kacper Majkowski

29 października 2023

1 Motywacja

1.1 Historia powstania

Uczenie maszynowe jest ostatnio bardzo popularnym tematem. Z pojawieniem się takich przydatnych technologi jak Chat GPT, lub szkodliwych jak Deepfake, widać że sztuczna inteligencja będzie grała orgomną rolę już w niedalekiej przyszłości. Ta praca skupi się na jednej dziedzinie uczenia maszynowego, mianowicie "uczenie przez wzmacnianie" (ang. "reinforcement learning / RL"). Jest to technika uczenia metodą "prób i błędów", lub dokładniej "nagród i kar". Można to lepiej przedstawić jako porównanie do tego jak uczą się ludzie i zwierzęta.

1.2 Inspiracja naturą

Wyobraźmy sobię mysz grasującą na małym osiedlu. Pewnego dnia wkrada się ona do domu A. Znajduje spiżarnię i dużo pysznego sera. Następnego dnia wkrada się do domu B. Tam zamiast sera znajduje groźnego kota. Mysz wie teraz że wkradanie się do domu A przynosi większą "nagrodę" niż do domu B, i powinna do niego chodzić częściej. Pojawia się jednak pewien problem. A co jeżeli w domu B kot strzeże bardziej wartościowej spiżarni niż tej w domu A? Może warto to sprawdzić, pomimo ryzyka? A może wybrać łatwiejszą opcję i chodzić cały czas do domu A? To wszystko ma swoje odzwierciedlenie w nauczaniu przez wzmacnianie.

2 Elementy działania uczenia przez wzmacnianie

2.1 Nazewnictwo

Wytłumaczenie pojęć:

Agent - Jednostka która wchodzi w interakcję ze środowiskiem, podejmuje decyzje oraz uczy się które decyzje są lepsze lub gorsze na podstawie otrzymanych nagród

Środowisko - Przestrzeń w której porusza się agent

Stan - Pewne unikalne warunki środowiska

Akcja - Decyzja podjęta przez agenta w danym stanie środowiska, prowadząca do zmiany stanu na nowy

Nagroda - Wartość otrzymywana przez agenta po każdej akcji, mówiąca czy wykonana akcja była właściwa czy nie

Epizod - Zbiór ruchów po których środowisko jest przywracane do stanu początkowego

Obserwacja - Zbadanie stanu przez agenta oraz ustalenie przewidywanych nagród za każdą możliwą akcję

Dzięki wytłumaczeniu tych pojęć, można lepiej przedstawić na czym polega algorytm uczenia przez wzmacnianie. Na początku tworzone jest środowisko z agentem w stanie wyjściowym. Następnie agent obserwuje obecny stan i wykonuje akcję na jej podstawie. Potem za wykonaną akcję przyznawana jest nagroda, po czym agent wyciąga z niej wnioski. Następnie proces zaczyna się od początku: Obserwacja -> Akcja -> Nagroda -> Nauka -> Nowy stan. Jeśli agent zakończy epizod, stan przywracany jest do początkowego, a agent rozpoczyna kolejny epizod, ale ze zdobytą już wiedzą.

2.2 Q-Tabela

Ale jak agent powinien zapisywać swoją wiedzę? Robi to za pomocą Tabeli, konkretnie Q-Tabeli. Przechowuje ona wszystkie kombinacje (Stan, Akcja) i zapisuje jak "opłacalna" jest to akcja w danym stanie. Nie jest to do końca nagroda za tę akcję, to trochę bardziej skomplikowane. Jak obliczamy konkretną wartość

wyjaśnione będzie w kolejnym punkcie. Przykładowa Q-Tabela może wyglądać tak:

Stan / Akcja	Akcja 1	Akcja 2	Akcja 3	Akcja 4
Stan 1	1	6	-3	5
Stan 2	4	1	0	-10
Stan 3	-8	3	6	0
Stan 4	2	0	-1	7
Stan 5	0	-5	2	4

Tutaj warto zaznaczyć że tworząc tabelę inicjalizujemy wszystkie wartości na 0, tak że zawsze mamy wartość do odczytania, nawet jeśli natrafiamy na nią po raz pierwszy. Odczytywanie tabeli jest bardzo proste. Dla przykładu jeżeli agent znajduje się w stanie 3, to z dotychczasowych doświadczeń wynika że najkorzystniejszyą akcją byłaby akcja 3. Jak jednak otrzymywać te wartości? Przy tym pomoże nam Równanie Bellmana.

2.3 Obliczanie wartości ruchu oraz równanie Bellmana

Równanie Bellmana pozwala nam uwzględznić wiele rzeczy które pomogą ocenić wartość danej akcji. Równanie jest dość złożone, więc zbudujmy je od podstaw. Załóżmy że właśnie wykonaliśmy akcję a w stanie s i otrzymaliśmy pewną nagrodę $r(s, a)$:

$$Q(s, a) = r(s, a)$$

$Q(s, a)$ - nowa wartość akcji w Q-Tabeli którą chcemy wyznaczyć
$r(s, a)$ - nagroda (ang. reward) za wykonanie akcji a w stanie s

Tak wyglądałoby równanie, gdybyśmy brali pod uwagę jedynie nagrodę za właśnie wykonaną akcję a w stanie s i nic poza tym.

Co jednak jeśli gorsza akcja teraz prowadzi do o wiele lepszej nagrody później? Jak uwzględnić to w równaniu? Robimy to, dodając do otrzymanej nagrody najlepszą wartość następnego ruchu po wykonaniu rozpatrywanej akcji. Innymi słowy, idziemy krok dalej i rozglądamy się, jakie akcje możemy wykonać po zrobieniu tego ruchu.

$$Q(s, a) = r(s, a) + \max_{a'} Q(s', a')$$

s' - Stan po wykonaniu akcji a w stanie s
$\max_{a'} Q(s', a')$ - Wartość najlepszej akcji w następnym stanie s' (maks po wszystkich możliwych akcjach)

Tutaj dochodzimy do jednego z kluczowych momentów. Skąd jednak mamy znać wartości ruchów w stanie s'? Tutaj z pomocą przychodzi Q-Tabela. Naiwnym podejściem byłoby rekursywne wykorzytywanie tej fromuły i zagłębiania się coraz to głębiej w drzewo możliwości. Zamiast tego odczytujemy już wcześniej wyliczoną wartość z Q-Tabeli.

Aby to lepiej zrozumieć rozważmy teraz sytuację gdy wszystkie wartości w Q-Tabeli są optymalne. Skoro wiemy że wartość następnego stanu i akcji $Q(s', a')$ również została wyliczona wcześniej za pomocą równania Bellmana, to możemy zamienić (s', a') na $(r(s', a') + \max_{a''} Q(s'', a''))$. Otrzymujemy wtedy:

$$Q(s, a) = r(s, a) + \max_{a'} r(s', a') + \max_{a''} Q(s'', a'')$$

Tutaj otrzymujemy kolejny element (s'', a''), dla którego możemy zastosować te podstawienie. Jeżeli będziemy tak kontynuować to zobaczymy że faktycznie to równanie mówi że wartość teraz to suma wszystkich kolejnych nagród, jeżeli kierujemy się zawsze najlepszymi akcjami:

$$Q(s, a) = r(s, a) + \max_{a'} r(s', a') + \max_{a''} r(s'', a'') + \max_{a'''} r(s''', a''') + ...$$

Oznacza to jednak, że nagrody którą dostaniemy w przyszłości mają dla nas takie same znaczenie jak nagrodę którą dostaliśmy teraz. Nie jest to zawsze najlepsze podejście. Nasza mysz pewnie wolałaby dostać ser teraz niż na miesiąc. Dlatego tutaj wprowadzimy kolejny parametr - "współczynnik dyskontowania" (and. "discount rate"). Mówi on jak spada dla nas wartość nagród, im dalej w przyszłości je dostaniemy. Parametr ten oznaczamy grecką literą γ (gamma). Przyjmuje on wartości od 0 do 1. Dla $\gamma = 0$ nie bierzemy przyszłych nagród pod uwagę i kierujemy się tylko natychmiastową nagrodą za ruch. Dla $\gamma = 1$, mamy sytuację jak w równaniu powyżej, gdzie przyszłe nagrody są dla nas dokładnie tak samo warte jak nagroda natychmiastowa. Wprowadzamy ten parametr do równania, dodając go jako współczynnik przy $\max_{a'} r(s', a')$

$$Q(s, a) = r(s, a) + \gamma * \max_{a'} Q(s', a')$$

Teraz jeżeli wykonamy przedstawione wcześniej podstawianie otrzymujemy:

$$Q(s, a) = r(s, a) + \gamma * \max_{a'} r(s', a') + \gamma^2 * \max_{a''} r(s'', a'') + \gamma^3 * \max_{a'''} r(s''', a''') + ...$$

Oznacza to, że dla $\gamma < 1$, każda kolejna nagroda będzie dla nas mniej wartościowa. Dla przykładu dla $\gamma = \frac{1}{2}$ nagroda którą dostalibyśmy za 2 ruchy jest dla nas

warta połowę swojej wartości, za 3 ruchy $\frac{1}{4}$ swojej wartości itd.

To co otrzymaliśmy jest właśnie znane jako równanie Bellmana. Przedstawia ono zależności między wartościami w optymalnej Q-Tabeli. My jednak nie mamy tej optymalnej Q-Tabeli, my chcemy do niej dążyć. Robimy to, powtarzając ten proces wiele razy: Obserwacja -> Akcja -> Nagroda -> Nowy stan.

Powinniśmy uwzględnić również wcześniejszą wiedzę o obecnej wartości $Q(s, a)$. Gdybyśmy tego nie robili, szybkie i znaczące zmainy w Q-Tabeli mogłyby zaburzyć działanie algorytmu. Spowalniając szybkość uczenia sprawiamy, że kosztem dłuższego procesu uczenia jest ono stabilniejsze i nie dochodzi do dużych wahań. Tutaj w grę wchodzi parametr zwany "współczynnik nauczania" (ang. "learning rate"). Definiuje on jaką wagę przykładamy do nowych informacji, a jaką do już znanych. Jest on oznaczany grecką literą α (alfa). Współczynnik nauczania przyjmuje wartość od 0 do 1. Dla wartości 0 program nic by się nie uczył, uważał by zdobytą nagrodę za bezwartościową. Dla współczynnika 1, program zapomiałby wszystko co wcześniej wiedział od danej parze (stan, akcja), i przepisywał nową nauczoną wartość do tabeli. Ważne jest więc aby odpowiednio dobrać współczynnik nauczania pomiędzy 0 a 1, odpowiednio do naszego problemu. Dzięki niemu możemy zmodyfikować nasze równanie aby wyglądało tak:

$$Q(s, a) = (1 - \alpha) * Q(s, a) + \alpha * (r(s, a) + \gamma * \max_{a'} Q(s', a'))$$

$Q(s, a)$ - obecna wartośc dla pary (stan, akcja) w Q-Tabeli, którą chcemy poprawić

Po krótkim spojrzeniu można zauważyć że jest to po prostu średnia ważona obecnej wartości oraz nowej, wyliczonej za pomocą równania Bellmana, gdzie α jest wagą nowej wartości a $1 - \alpha$ wagą tej z tabeli.

Gdy już wiemy jak uzupełniać Q-Tabelę na podstawie stanów, akcji i nagród, zobaczmy to na przykładzie wcześniej przedstawionej tabeli

Stan / Akcja	Akcja 1	Akcja 2	Akcja 3	Akcja 4
Stan 1	1	6	-3	5
Stan 2	4	1	0	-10
Stan 3	-8	3	6	0
Stan 4	2	0	-1	7
Stan 5	0	-5	2	4

Przyjmijmy również dane parametry:
Współczynnik nauczania: $\alpha = 0.3$
Współczynnik dyskontowania: $\gamma = 0.8$

Załóżmy że agent znajdował się w stanie 4 oraz podjął akcję 3. Za tę akcję otrzymał nagrodę w wysokości 5 punktów. Po wykonaniu tej akcji znalazł się w stanie 1. Oznacza to że:
Stan $s = 4$
Akcja $a = 3$
Nagroda $r(4, 3) = 5$
Nowy stan $s' = 1$

Wstawiając te wartości do formuły otrzymujemy:
$Q(s, a) = (1 - \alpha) * Q(s, a) + \alpha * (r(s, a) + \gamma * \max_{a'}(s', a')) = (1 - 0.3) * (-1) + (0.3) * (5 + 0.8 * max([1, 6, -3, 5])) = -0.7 + (0.3) * (5 + 0.8 * 6) = -0.7 + 2.94 = 2.24$
Tę wartość wstawiamy w pole (stan 4, akcja 3), po czym wykonujemy koleją akcję, będąc w stanie 1. Tylko jak wybrać którą akcję mamy wykonać? Najlepszą - akcję 2, aby dalej polepszać najlepsze dotąd rozwiązanie. Czy może jakąś inną, w nadzieji że znajdziemy później coś lepszego?

2.4 Eksploracja vs Eksploitacja

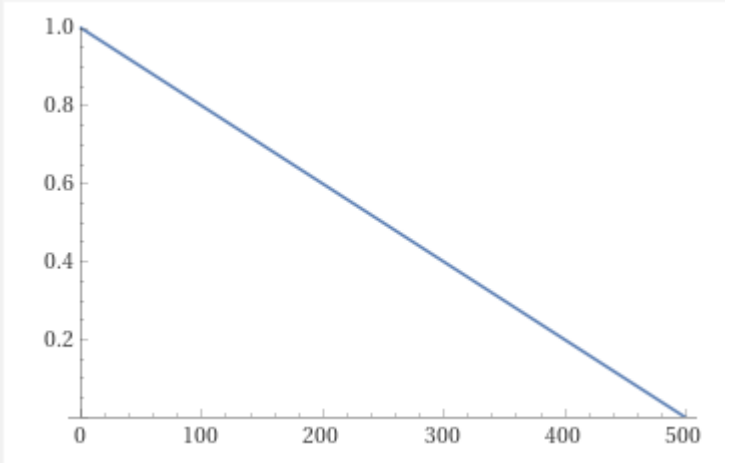
Najlepiej byłoby gdybyśmy na początku uczenia wybierali akcje losowo (nazywamy to exploracją, ang. exploration) . Pozwoli to na przejrzenie jak najwięcej liczby stanów, aby jak najwięcej dowiedzieć się o środowisku. Jednocześnie dobrze by było gdybyśmy w późniejszej fazie częściej wybierali najlepszą odkrytą ścieżkę, aby ją dalej polepszać i otrzymywać coraz lepsze rozwiązania na jej podstawie (nazywamy to exploitacją, ang. exploitation). Tylko jak to zaimplementować?

Zastosujemy kolejny parametr, który nazwiemy "współczynnikim eksploracji" (ang. "exploration rate"). Oznaczymy go grecką literą ϵ (epsilon). Podobnie jak pozostałe wcześniej użyte parametry, przyjmuje on wartość od 0 do 1. Jego zastosowanie jest bardzo proste - przed wyborem akcji losujemy liczbę x od 0 do 1. Jeżeli $x < \epsilon$ wtedy wybieramy losową akcję za wszystkich możliwych, w przeciwnym wypadku wybieramy akcję o największej wartości. Można zauważyć że dla $\epsilon = 1$ zawsze będziemy wybierać losowe akcje, dla $\epsilon = 0$ zawsze będziemy wybierać najlepsze akcje, a dla np. $\epsilon = 0.5$ będziemy w połowie przypadków wybierać losową akcję, a w połowie najlepszą akcję.

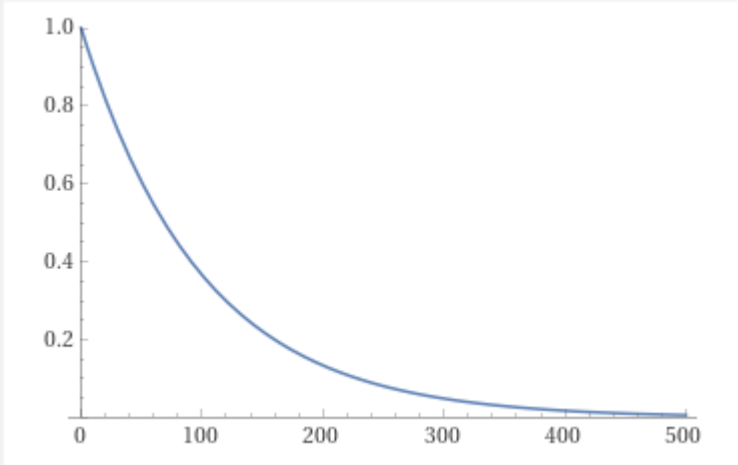
Ideą jest więc aby zacząć od wysokiego współczynnika eksploracji ϵ, i z czasem uczenia stopniowo go zmniejszać. Jest na to wiele sposobów ale omówimy sobie dwa - liniowy oraz geometryczny. W sposobie liniowym z każdą iteracją odejmujemy pewną wartość od współczynnika exploracji.

Tak może wyglądać przykładowy wykres przedstawiający wartość współczynnika

eksploracji zależnie od postępu nauczania algorytmu używając metody liniowej:



A tak wygląda przykładowy wykres dla metody geometrycznej:



Można zauważyć że w metodzie geometrycznej dużo więcej czasu algorytm spędza z niższym współczynnikiem eksploracji. Oznacza to że szybciej porzuca strategię odkrywania środowiska i skupia się na doskonaleniu odnalezionego już rozwiązania. W przedstawionych przykładach wybrałem startowy $\epsilon = 1$ i końcowy $\epsilon = 0$, ale można dobrać dowolny przedział, zależnie od potrzeb i warunków środowiska.

3 Przedstwanienie działania uczenia przez wzmacnianie w praktyce

3.1 "Catch the coins"

Opis zaimplementowanej gry "Catch the coins" oraz zauważonych problemów z losowością. Opis celu gry i dlaczego akurat taką. Parę klatek z gry, wykorzystane

technologie.

3.2 Reprezentacja stanu gry

Przedstawienie sposobu reprezentacji stanu gry jako liczba naturalna i dlaczego pomoże to w implementacji Q-Tabeli

3.3 Implementacja Q-Tabeli

Przedstawienie implementacji Q-Tabeli która nie wymaga tablicy długości milionów komórek, zamiast tego użycie hashmapy

3.4 Nagrody

Opis za co i dlaczego agent (gracz) dostaje nagrody oraz kary

3.5 Program w całości

Przedstwaienie flow chart działania algorytmu, opisanie każdego kroku.

4 Wyniki

Przestawienie wyników działania algorytmu, wykresy pokazujące działanie i postępy w wynikach na różnych mapach startowych. Przedstawienie wpływu parametrów takich jak współczynnik uczenia i współczynnik eksploracji na skutecznośc uczenia.

5 Podsumowanie i zastosowania

Uczenie przez ma wiele zastosowań, poza nauką w gry przedstawioną w tej pracy. Stosuje się ją to tworzenia algorytmów.

Jakieś adekwatne podsumowanie, dlaczego wybrałem ten temat, jak można dalej ulepszać zaprezentowany algorytm (Deep Q-Learning, Double Deep Q-Learning).