Rozdział 2

Przedstawienie problemu badawczego

Uczenie ze wzmocnieniem jest jednym z trzech głównych metod uczenia maszynowego. Sposób ten opiera się na interakcji agenta z środowiskiem. Agent za pomocą metody prób i błędów dostosowuje swoje parametry aby jego akcje skutkowały jak najwyższą kumulatywną nagrodą.

2.1. Nagroda

Jedyną wskazówką dla agenta w jaki sposób się zachować jest nagroda, dlatego tak ważne jest prawidłowe jej zdefiniowanie.

Agent powinien być tak nagradzany, żeby większa nagroda wiązała się z bardziej pożądanym zachowaniem np. robot sprzątający który zgarnie więcej kurzu będzie lepiej oceniony. Czyli jest to kwestia tego jak konkretne czynności w jaki sposób są nagradzane.

Idea im większej nagrody, tym większa użyteczność działania jest zgodna z popularną hipotezą nagrody: "To wszystko, co rozumiemy przez cele, można dobrze rozumieć jako maksymalizację oczekiwanej wartości skumulowanej sumy otrzymanego sygnału skalarnego(zwanego nagrodą)."[10]

Agent dokonując decyzji, kieruje się natychmiastową nagrodą oraz szacowaną nagrodą w przyszłości, do regulacji co jest bardziej brane pod uwagę służy współczynnik dyskontujący γ o zakresie [0,1], który wymnożony z szacowaną nagrodą zmienia jej wpływ na zachowanie agenta.

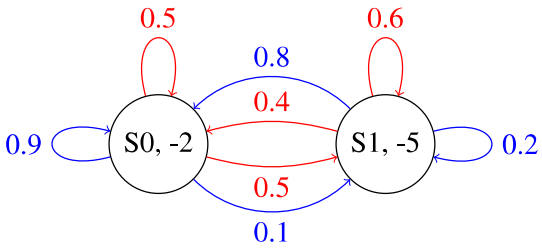
2.2. Proces decyzyjny Markowa

Proces decyzyjny Markova jest matematycznym formalizmem problemu uczenia ze wzmocnieniem. Jest on definiowany jako krotka (S, A, P, R, s_0), gdzie:

- S - zbiór stanów,
- A - zbiór akcji,
- P - oznacza funkcję przejścia,
- R - oznacza funkcję nagrody,
- s_0 - oznacza stan początkowy.
- $P(s_{t+1}|s_t, a_t)$ - dynamika, zwracająca prawdopodobieństwo przejścia do stanu $s_t + 1$ pod warunkiem wykonania akcji a_t w stanie s_t.

- $R(s_t, a_t, s_{t+1})$ - funkcja nagrody, która dla zmiany stanu spowodowanej akcją zwraca jak bardzo to zdarzenie było pożądane
- s_0 - stan początkowy epizodu

Ważną cechą procesu decyzyjnego Markowa jest własność Markowa, według której agent swoje akcje opiera wyłącznie na podstawie aktualnej obserwacji gry, tzn. nie pamięta zeszłych stanów gry.[3]

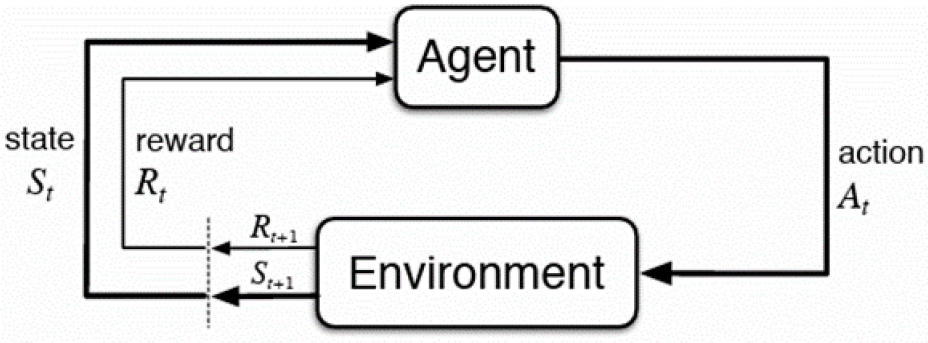


Rysunek 2.1: Prosty przykład Procesu Decyzyjnego Markowa.

Na rysunku 2.1, niebieskie strzałki mogą reprezentować akcję a_0, czerwone strzałki a_1, S_0 i S_1 stany w których można się znaleźć, a liczby obok nich to nagroda jaką się uzyskuje za każde wejście. Liczby nad strzałkami reprezentują prawdopodobieństwo przejścia do danego stanu pod warunkiem wykonania konkretnej akcji.

Optymalizacja rozwiązania prostych modeli MDP jest łatwa dla człowieka, jednak aby środowisko było przybliżone prawdziwemu światu, odpowiadający jemu model będzie znacznie bardziej skomplikowany. Uczenie ze wzmocnieniem zajmuje się ich rozwiązywaniem.

W uproszczeniu pętla zdarzeń podczas interakcji agenta ze środowiskiem wygląda następująco:



Rysunek 2.2: Schemat zdarzeń w uczeniu ze wzmocnieniem [10]

Na powyższym rysunku 2.1 widoczne jest jak agent otrzymuje aktualny stan S_t i nagrodę R_t, aby następnie wykonać akcję A_t, która spowoduje reakcję środowiska zwracając nowy stan S_{t+1} oraz nagrodę R_{t+1}.

2.3. Policy i funkcja wartości

Każdy algorytm uczenia ze wzmocnieniem zawiera policy π – funkcję która definiuje zachowanie agenta na podstawie obserwacji gry. Zadaniem algorytmu jest jej optymalizacja, żeby wszystkie wykonane akcje prowadziły do jak największej nagrody. Można to przedstawić następująco:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E[R|\pi] \text{ [3]} \tag{2.1}$$

Niektóre metody korzystają także z funkcji wartości, która przypisuje wartość do każdego stanu, lub pary stanu i akcji aby określić jak wielka jest oczekiwana wartość skumulowanej nagrody zaczynając od danego stanu i podążając do końca epizodu zgodnie z optymalną policy. Wzór funkcji wartośći definiuje się:

$$V^*(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right] \tag{2.2}$$

,gdzie γ jest współczynnikiem dyskontującym.[3]

Algorytmy mogą być on-policy, gdzie podejmowane decyzję oparte są głównie o policy, oraz off-policy, w którym to funkcja wartości ocenia wartość danych akcji, policy ogranicza się wtedy jedynie do wyboru akcji, zważając na ich ocenę przez funkcję wartości, przy czym nie zawsze musi to być najlepsza (np. w celu eksploracji).

2.4. Balans między eksploracją a eksploatacją

Jednym z głównych wyzwań uczenia ze wzmocnieniem jest równoważenie między eksploracją a eksploatacją. Z jednej strony potrzebna jest eksploracja, aby agent mógł zapoznać się ze środowiskiem, dzięki czemu może poznać lepsze strategie. Z drugiej strony aby agent osiągał wysokie wyniki, powinien korzystać ze swojej wiedzy wybierając konkretne znane mu ścieżki.

Aby zbalansować ten problem algorytmy oparte o funkcję wartości wykorzystują specjalną funkcję policy np. ϵ-greedy która określa z jakim prawdopodobieństwem zostanie wybrana losowa akcja, z czasem ϵ jest zmniejszany, ponieważ przestrzeń stanów gry jest już bardziej znana agentowi. Za to metody on-policy często oparte są o gradient, który zwraca dystrybucję prawdopodobieństwa wobec akcji. Wykorzystywany jest także często współczynnik entropii.

2.5. Temporal Diffrence a Monte Carlo

Modele uczenia ze wzmocnieniem stosują dwa podejścia do wymaganej ilości doświadczeń potrzebnych do treningu.

Monte Carlo najpierw zapisuje pełny epizod rozegrany przez agenta, żeby następnie wykorzystać informacje o końcowej nagrodzie. Jako że znana jest wartość nagrody, nie wprowadza to obciążenia wykorzystując tą metrykę do treningu. Wykorzystanie jednak jednej z dużej ilości możliwych ścieżkek interakcji między środowiskiem a agentem powoduje wysoką wariancję.

Temporal Diffrence do aktualizacji parametrów potrzebuje jedynie jednej interakcji ze środowiskiem. Funkcja wartości przybliża spodziewaną nagrodę, dzięki czemu możliwe jest zestawienie przybliżeń z dwóch następujących po sobie stanów, rozpatrując otrzymaną nagrodę. Dla funkcji wartości stanu wzór wygląda następująco:

$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \text{ [3]} \tag{2.3}$$

2.6. Optymalizacja hiperparametrów

Algorytmy uczenia ze wzmocnieniem charakteryzują się wieloma zmiennymi, które wpływają drastycznie na ich działanie. Dlatego optymalizacja hiperparametrów jest kluczowa przed ewaluacją modeli na środowisku. Najprostrzym sposobami są grid search - przeszukiwanie wszystkich możliwych

wartości i random search - losowe dobieranie, są jednak bardzo czasochłonne dla dużych zakresów. Optymalizację można także przeprowadzić automatycznie za pomocą specjalnych algorytmów które decydują które wartości parametrów są warte eksplorowania. Jest to bardziej skomplikowane rozwiązanie, jednak znacznie szybciej znajduje się w ten sposób najlepsze zestawy hiperparametrów.

W tej pracy zostanie wykorzystany algorytm TPE zaimplementowany w Optuna.[1] TPE tworzy dwa modele prawdopodobieństwa, $l(x)$ dla wartości uznanych za dobrych kandydatów i $g(x)$ dla pozostałych. Dobranie hiperparametrów polega na maksymalizowaniu stosunku $l(x)/g(x)$.

Do optymalizacji zostanie wykorzystane RL Baselines3 Zoo[7] opakowujący Optuna i wyznaczający zakresy wartości na których dobieranie powinno się opierać.

2.7. Porównywane algorytmy

Każdy z wybranych algorytmów charakteryzuje się użyciem głębokich sieci neuronowych do reprezentacji wiedzy oraz wykorzystują metodę temporal diffrence. Różnią się one typem implementowanych architektur, polityk oraz stosują różne techniki aby przeciwdziałać znanym problemom trenowania modeli uczenia ze wzmocnieniem. Wszystkie zostały zaimplementowane w Stable baselines3.[8]

2.7.1. DQN

Deep Q-learning(DQN) jest metodą off-policy, opartą o przypisywanie każdej możliwej akcji poprzez funkcję wartości w każdym możliwym stanie wartości Q. W porównaniu do poprzednika Q-learning różni się sposobem reprezentacji wiedzy, ponieważ zamiast tablicy dla każdej wartości Q, wykorzystywana jest sieć neuronowa. Pozwala to na uczenie modeli w środowiskach o znacznie większej dyskretnej przestrzeni akcji i stanów. DQN wykorzystuje Experience Replay – określonej wielkości pamięć, w której przechowuje się ostatnie doświadczenie agenta przez krotki składające się z stanu, akcji, otrzymanej nagrody oraz następnego stanu. Model wykorzystuje go w treningu, ucząc się z losowo wybranych krotek. Pozytywnymi skutkami takiego rozwiązania jest na przykład lepsze zapamiętywanie zeszłych zdarzeń oraz zapobieganie korelacji między doświadczeniami.[4]

Do balansu między eksploracją a eksploatacją DQN używa policy ϵ-greedy.

Dodatkowo do zaimplementowanego w eksperymentach DQN wprowadzono fixed Q-target. Wykorzystuje ona dodatkową sieć neuronową Q-target która służy stabilizacji treningu poprzez ograniczenie częstotliwości zmiany celu do którego dąży algorytm.

Do wyznaczeniu celu, do którego algorytm się zbliża, wykorzystuje się Temporal Diffrence względem Q-target:

$$y_t = r_t + \gamma \max_a Q_{\text{target}}(s_{t+1}, a) \tag{2.4}$$

Następnie porównuje się predykcje TD Q-target i Q-network:

$$L(\theta_{\text{network}}) = \mathbb{E}\left[(y_t - Q_{\text{network}}(s_t, a_t))^2\right] \tag{2.5}$$

Na koniec obliczany jest gradient descent z tej funkcji, który wskaże kierunek zmian θ_{network} w kontekście minimalizacji $L(\theta)$[6]

Dla M kroków, algorytm który aktualizuje swoje wagi co akcję wygląda następująco [6]: