UNIWERSYTET JAGIELLOŃSKI
WYDZIAŁ MATEMATYKI I INFORMATYKI
INSTYTUT INFORMATYKI ANALITYCZNEJ

Jakub Dyczek

# Analiza algorytmów Deep Q-learning oraz Proximal Policy Optimization

PRACA LICENCJACKA NAPISANA POD KIERUNKIEM
dra Michała Wrony

KRAKÓW 2020

# Spis treści

# Wstęp

Uczenie przez wzmacnianie jest jedną z gałęzi uczenia maszynowego w której celem jest znalezienie optymalnej strategii dla agenta w nieznanym środowisku. Jednym z bardziej znanych przykładów uczenia ze wzmocnieniem jest program AlphaGo [**1**], który wygrywa pojedynki z najlepszymi graczami w grze w go.

Przełomowe rozwiązania uczenia maszynowego ostatnich lat doprowadziły do powstania wielu różnych algorytmów uczenia przez wzmacnianie. W pracy wyjaśniony zostaje algorytm Deep Q-Networks (DQN) oparty na znanej już w 1992 roku idei Q-learningu [**2**] oraz nowszy algorytm Proximal Policy Optimization (PPO) [**3**] pochodzący z 2017 roku i należący do klasy nazywanej algorytmami optymalizacji strategii. Algorytm PPO powstał jako uproszczona oraz w pewnych przypadkach wydajniesza wersja algorytmu Trust Region Policy Optimization (TRPO).

W pierwszym rozdziale sformalizowany zostaje paradygmat uczenia przez wzmacnianie oraz wypisane są równania Bellmana. Drugi rozdział zawiera opis wariantów algorymu DQN oraz pokazane zostają ich wyniki w środowisku dostarczonym przez OpenAI Gym [**4**]. W ostatnim rozdziale pokazane jest rozumowanie prowadzące do powstania algorytmu PPO oraz porównane są wyniki jego czterech wersji w zależności od architektury sieci neuronowej oraz pewnego hiperparametru.

Do pracy dołączony jest kod[1] implementujący zaprezentowane warianty algorytmu DQN oraz algorytm PPO.

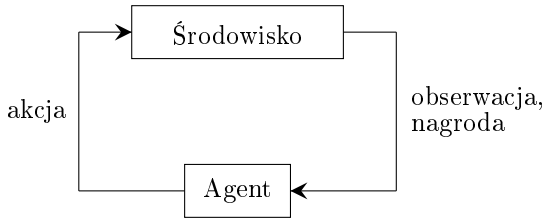
[1] https://github.com/JDkuba/dqnppo

ROZDZIAŁ 1

# Wprowadzenie do uczenia przez wzmacnienie

W pierwszym rozdziale wyjaśnione zostaje uczenie przez wzmacnienie (inaczej uczenie ze wzmocnieniem) oraz wprowadzone są pojęcia potrzebne w dalszej części pracy.

Uczenie przez wzmacnienie jest dziedziną uczenia maszynowego, w której agent oddziałowuje na środowisko, aby osiągnąć jak największą nagrodę. Akcje podejmowane przez agenta mogą spowodować zmianę stanu środowiska.




Interakcją agenta ze środowiskiem nazywamy ciąg $s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots$, gdzie $s_t, a_t, r_t$ są odpowiednio stanami, akcjami wykonanymi przez agenta oraz nagrodami otrzymanymi w czasie $t$.

Uczenie ze wzmocnieniem może być modelowane za pomocą decyzyjnych procesów Markowa.

Definicja 1.1. Decyzyjnym procesem Markowa (MDP) nazywamy czwórkę $(S, A, P, R)$, gdzie

(1) Zbiór $S$ jest zbiorem stanów.
(2) Zbiór $A$ jest zbiorem akcji.
(3) $P = \{P_a\}_{a \in A}$ i $P_a(s, s') = \mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$ jest prawdopodobieństwem przejścia ze stanu $s$ do $s'$ wykonując akcję $a$.
(4) $R = \{R_a\}_{a \in A}$ i $R_a(s, s')$ jest nagrodą otrzymaną przy przejściu ze stanu $s$ do $s'$ wykonując akcję $a$.

Definicja 1.2. Strategią nazywamy odwzorowanie

$$\pi : S \times A \to [0, 1]$$
$$\pi(s, a) = \mathbb{P}(a_t = a \mid s_t = s)$$

Postępowanie według strategii $\pi$ w czasie $t$ oznacza wybór akcji $a_t$ zgodnie z rozkładem $\pi(s_t, \cdot)$.

Definicja 1.3. Funkcją wartości strategii $\pi$ dla stanu $s$ nazywamy oczekiwaną zdyskontowaną sumę przyszłych nagród

$$V_\pi(s) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \mid s_0 = s \right],$$

gdzie akcje $a_t$ są wybierane według strategii $\pi$. Liczba $\gamma \in [0, 1]$ nazywana jest współczynnikiem dyskontowania.

Celem agenta jest znalezienie strategii bliskiej optymalnej strategii $\pi^*$ maksymalizującej funkcję wartości $V_\pi$. Oznaczamy $V^*(s) := \max_\pi V_\pi(s)$.

Definicja 1.4. Funkcją wartości akcji strategii $\pi$ nazywamy funkcję

$$Q_\pi : S \times A \to \mathbb{R}$$

spełniającą

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \mid s_0 = s, a_0 = a \right],$$

gdzie akcje $a_t$ są wybrane według strategii $\pi$ dla $t \geq 1$.

Obserwacja 1.5. Pomiędzy funkcjami $V_\pi$ oraz $Q_\pi$ zachodzi związek

$$V_\pi(s) = \sum_{a \in A} \pi(s, a) Q_\pi(s, a) = \mathop{\mathbb{E}}_{a \sim \pi} [Q_\pi(s, a)]$$

Dla optymalnej strategii $\pi^*$, jeśli znane jest $Q^* := Q_{\pi^*}$, to żeby postępować zgodnie z $\pi^*$ wystarczy w każdym kroku brać akcję maksymalizującą wartość $Q^*(s, \cdot)$. Wynika z tego, że celem agenta może być znalezienie $Q^*$. Niektóre algorytmy uczenia ze wzmocnieniem korzystają z funkcji $Q$, a niektóre z funkcji $V$.

Twierdzenie 1.6. *Zachodzą poniższe równania, zwane również równaniami Bellmana.*

$$V^*(s) = \max_{a \in A} \mathop{\mathbb{E}}_{s' \sim P_a(s, \cdot)} [R_a(s, s') + \gamma V^*(s')]$$

$$Q^*(s, a) = \mathop{\mathbb{E}}_{s' \sim P_a(s, \cdot)} \left[ R_a(s, s') + \gamma \max_{a' \in A} Q^*(s', a') \right] \tag{1}$$

Korzystając z równań Bellmana oraz teorii programowania dynamicznego można by próbować wyliczyć optymalne wartości $V^*$ lub $Q^*$. W środowiskach dla których wykorzystywane jest uczenie ze wzmocnieniem na ogół nie jest to możliwe. Programowanie dynamiczne wymaga pełnej znajomości wartości $P_a$ oraz nagród dla wszystkich kombinacji akcji i stanów. W uczeniu ze wzmocnieniem wartości te są obserwowane po wykonaniu danego ruchu.

Problemem jest również rozmiar przestrzeni stanów i akcji - w programowaniu dynamicznym stany przeszukiwane są w sposób wyczerpujący, podczas gdy uczenie ze wzmocnieniem wykorzystuje jedynie przeszłe interakcje. Powoduje to, że algorytmy uczenia ze wzmocnieniem są skierowane na znalezienie strategii bliskiej optymalnej.

ROZDZIAŁ 2

# Algorytmy Q-learning oraz Deep Q-learning

W tym rozdziale omówione zostaną algorytmy Q-learning i Deep Q-Networks wraz ze swoimi wariantami. Zaprezentowane zostaną empiryczne wyniki otrzymane przy implementacji algorytmu DQN.

## Q-learning

Algorytm Q-learning opiera się na równaniu Bellmana. W każdym kroku obliczana jest wartość

$$Q : S \times A \to \mathbb{R}$$

przybliżająca $Q^*$. Początkowo $Q$ jest inicjalizowana w losowy sposób. Po prawej stronie równania Bellmana (1) wykorzystywana jest wartość oczekiwana nagrody oraz wartość oczekiwana $Q^*$ następnego stanu, które są nieznane. Zamiast tego, można wstawić rzeczywiste wartości otrzymane przy interakcji ze środowiskiem i przesuwać obecną wartość $Q$ w tym kierunku.

W kolejnych iteracjach przy przejściu ze stanu $s_t$ do stanu $s_{t+1}$ wykonując akcję $a_t$ obliczane jest

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(R_{a_t}(s_t, s_{t+1}) + \gamma \max_{a \in A} Q(s_{t+1}, a)), \tag{2}$$

gdzie $\alpha$ jest współczynnikiem uczenia, $0 < \alpha \leq 1$.

Mając dane wartości $Q$, dla danego stanu $s_t$ kolejną akcję można wybrać na różne sposoby. Jedną z możliwości jest wybieranie akcji zgodnych z $Q$, czyli takich dla których $Q(s, \cdot)$ jest największe. Na przykład

$$a_{t+1} = \text{argmax}_{a \in A} Q(s_t, a)$$

lub według rozkładu $Q(s, \cdot)$ o ile

$$\sum_{a \in A} Q(s, a) < \infty$$

Może to jednak sprawić, że algorytm nie pozna efektów akcji o małych wartościach $Q$, które końcowo mogą okazać się lepsze.

Innym sposobem jest całkowicie losowy wybór akcji. W takim podejściu eksploracja algorytmu będzie bardzo wysoka, ale spędzi on dużo czasu na liczenie wartości dla strategii dalekich od optymalnych.

Jednym z rozwiązań tego problemu jest wybór nazywany $\epsilon$-zachłannym. W każdym kroku z prawdopodobieństwem $\epsilon$ wybierana jest akcja w sposób losowy, niezależnie od wartości $Q$. Z prawdopodobieństwem $1 - \epsilon$ wybierana jest akcja zgodnie z $Q$. Taki wybór został zastosowany w omawianej implementacji.

## Aproksymacja funkcji

Dosłowne zaimplementowanie powyższego algorytmu mogłoby być bardzo nieefektywne dla dużych przestrzeni stanów. Zapamiętanie wartości $Q$ dla każdej pary $(s, a)$ mogłoby być niemożliwe, a nawet jeśli, to liczenie wartości akcji dla wszystkich takich par byłoby zbyt wolne. Z tego powodu stosuje się aproksymacje funkcji $Q$ ([**5**]). Jedną z metod jest liniowiowa aproksymacja za pomocą funkcji $\{\psi_k\}_{k=1}^d$, które można łatwo wyliczyć dla każdej pary stan-wartość

$$Q(s,a) = \sum_{k=1}^{d} \theta_k \psi_k(s,a)$$

Wagi $\{\theta_k\}_{k=1}^d$ można obliczać za pomocą regresji liniowej, gdzie celem jest

$$R_{a_t}(s_t, s_{t+1}) + \gamma \max_{a \in A} Q(s_{t+1}, a) \tag{3}$$

Inny wariantem jest wykorzystanie sieci neuronowych (algorytm deep Q-Networks, DQN), gdzie wartość $Q$ jest aproksymowana siecią neuronową o tym samym celu. W implementacji dołączonej do pracy sieć jako wejście dostaje stan $s$ oraz zwraca rozkład $Q(s, \cdot)$ zgodnie z którym wybierana jest akcja $a$.

## Experience Replay

Algorytm DQN korzysta z techniki nazwanej *experience replay*. Zamiast aktualizowania wag sieci co każdy krok, wstawia się do pamięci wpisy $(s_t, a_t, r_t, s_{t+1})$. Następnie losuje się z pamięci *batch_size* próbek, które zostają użyte do treningu sieci neuronowej. Dzięki takiemu rozwiązaniu zredukowana zostaje korelacja pomiędzy kolejnymi próbkami na których trenowana jest sieć, a w konsekwencji optymalizacja jest bardziej wydajna ([**6**]).

Standardowo jako pamięć wykorzystuje się kolejkę, a wybór próbki następuje w sposób jednostajny. Istnieje wariant nazywany *Prioritized Experience Replay* ([**7**]). Bazuje on na założeniu, że bardziej wartościowe są próbki dla których strata względem celu jest większa. Niech w pamięci będą przechowywane próbki postaci $\{(s_{t_i}, a_{t_i}, r_{t_i}, s_{t_i+1})\}_{i=1}^{\text{memory\_size}}$. Stratę próbki $i$ określamy jako

$$\delta_i = r_{t_i} + \gamma \max_{a \in A} Q(s_{t_i+1}, a) - Q(s_{t_i}, a_{t_i})$$

Próbki losuje się z rozkładem zgodnym z $|\delta_i|$, ale żeby nawet próbki o małej stracie miały szanse na wylosowanie, do każdej straty dodaje się $\epsilon > 0$. Dla

$$p_i = |\delta_i| + \epsilon$$

prawdopodobieństwo wylosowania próbki $i$ określamy jako

$$\mathbb{P}(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha},$$

gdzie $\alpha \in [0, 1]$ jest poziomem priorytetyzacji. Jeśli $\alpha = 0$, to wariant ten jest równoważny standardowemu *experience replay*. Im $\alpha$ jest bliższe 1 tym bardziej wybór próbki jest zależny od straty.

### Pseudokod

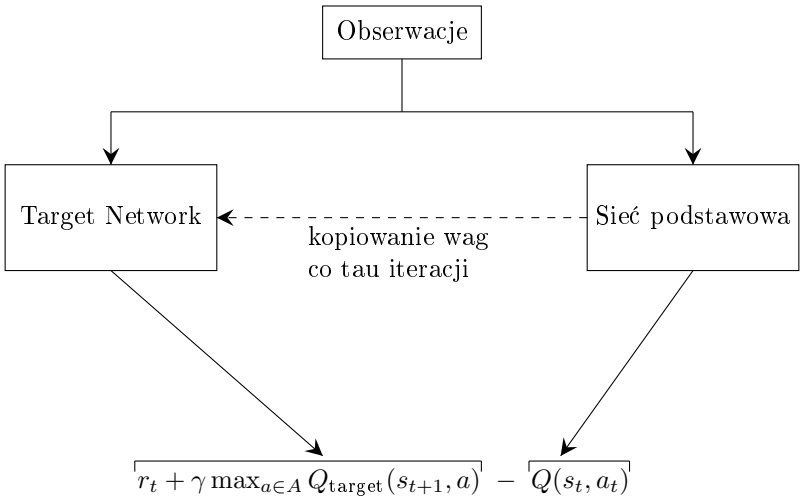
W pseudokodzie algorytmu DQN (Alg. 1) bufor $M$ może być zarówno kolejką jak i pamięcią z priorytetyzacją. Linijki 5-8 odpowiadają $\epsilon$-zachłannemu wyborowi akcji. W wierszu 13 zbiór $B$ wielkości *batch_size* wylosowany jest z rozkładem jednostajnym lub priorytetem w zależności od wybranego wariantu. W linijkach 14-17 wyliczone są aproksymowane wartości $Q$ z wylosowanego zbioru oraz wartości celu (3). Dla przejrzystości, w pseudokodzie jest pętla *for* jednak w celu optymalizacji obliczeń ten fragment został zaimplementowany za pomocą mnożenia macierzy. Wiersze 18-19 odpowiadają za wyliczenie straty wartości $Q$ oraz aktualizację wag sieci neuronowej za pomocą propagacji wstecznej.

**Algorytm 1** Deep Q-learning z Experience Replay

1: Inicjalizacja pamięci (buforu) $M$ o rozmiarze *memory_size*
2: Inicjalizacja sieci neuronowej *nn* aproksymującej wartości $Q$
3: Pobranie stanu początkowego $s$
4: **for all** $1 \leq i \leq$ episodes **do** ▷ *episodes* - maksymalny czas interakcji
5: $r \leftarrow \text{rand}(0, 1)$
6: **if** $r < \epsilon$ **then**
7: Wybierz losową akcję $a$
8: **else**
9: Wybierz akcję $a$ zgodną z nn.forward($s$) ▷ nn aproksymuje $Q$
10: **end if**
11: Wykonaj wybraną akcję $a$ oraz pobierz nowy stan $s'$ wraz z nagrodą $r$
12: $M$.push($s, a, r, s'$)
13: $B \leftarrow M$.rand(batch_size)
14: **for all** $(s, a, r, s')$ in $B$ **do**
15: $Q(s, a) \leftarrow$ nn.forward($s$)[$a$]
16: $Q_{\text{target}}(s, a) \leftarrow r + \gamma \max_{a \in A}$ (nn.forward($s'$)[$a$])
17: **end for**
18: loss $\leftarrow$ MSEloss($Q, Q_{\text{target}}$) ▷ dla obliczonych wyżej $Q, Q_{\text{target}}$
19: Zaktualizuj wagi sieci nn zgodnie z optymalizacją względem loss
20: $s \leftarrow s'$
21: **end for**

### Target Network

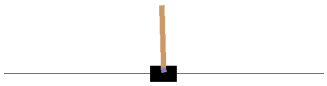
W kolejnych iteracjach cel trenowanej sieci może się różnić. Zależy on bowiem od wartości $Q$, która się zmienia. Można złagodzić ten efekt dodając drugą sieć neuronową *target network*, będącą tej samej architektury co sieć aproksymująca $Q$ ([**6**]). Wagi nowej sieci nie będą aktualizowane w każdym kroku, a co każde *tau* iteracji wagi będą kopiowane z sieci aproksymującej $Q$. Dzięki takiemu rozwiązaniu, uczenie jest stabilniejsze, gdyż przez pewną liczbę iteracji cel jest stały. Algorytm DQN z siecią *target network* nazywany jest także double deep Q-networks (DDQN).




RYSUNEK 1. Schemat sieci neuronowych w algorytmie DQN z wykorzystaniem target network.

### Implementacja

Aby porównać opisane algorytmy wykorzystane zostanie środowisko *CartPole-v0* z biblioteki *gym* od OpenAI [**4**]. Celem agenta jest utrzymanie drążka przyczepionego do ruchomego wózka w pozycji stojącej. Agent ma do dyspozycji dwie akcje: przemieszczenie wózka w lewą lub prawą stronę. Przestrzenią stanów jest wektor czterech elementów typu *float32*, którego współrzędne oznaczają kolejno pozycję wózka (z przedziału $[-2.4, 2.4]$), prędkość wózka, kąt nachylenia drążka względem wózka (z przedziału $[-41.8°, 41.8°]$) oraz prędkość końcówki drążka.

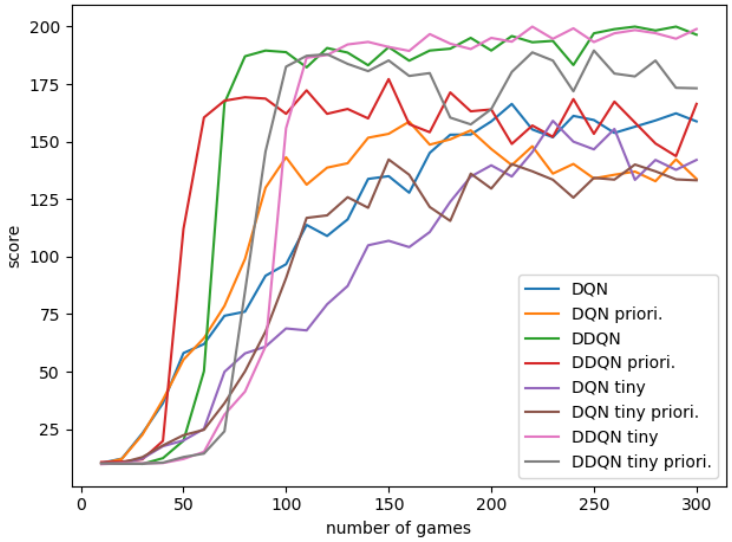


Poniższa tabelka zawiera parametry rozważanych algorytmów. Sieci neuronowe mają dwie warstwy ukryte, gdzie $h_1$ jest rozmiarem pierwszej warstwy, a $h_2$ rozmiarem drugiej warstwy.

| Nazwa | Target network | Buffer | Architektura sieci |
|---|---|---|---|
| DQN | — | Simple Buffer | $\{h_1 : 64, h_2 : 128\}$ |
| DDQN | + | Simple Buffer | $\{h_1 : 64, h_2 : 128\}$ |
| DQN tiny | — | Simple Buffer | $\{h_1 : 32, h_2 : 32\}$ |
| DDQN tiny | + | Simple Buffer | $\{h_1 : 32, h_2 : 32\}$ |
| DQN priori. | — | Prioritized Buffer | $\{h_1 : 64, h_2 : 128\}$ |
| DDQN priori. | + | Prioritized Buffer | $\{h_1 : 64, h_2 : 128\}$ |
| DQN tiny priori. | — | Prioritized Buffer | $\{h_1 : 32, h_2 : 32\}$ |
| DDQN tiny priori. | + | Prioritized Buffer | $\{h_1 : 32, h_2 : 32\}$ |

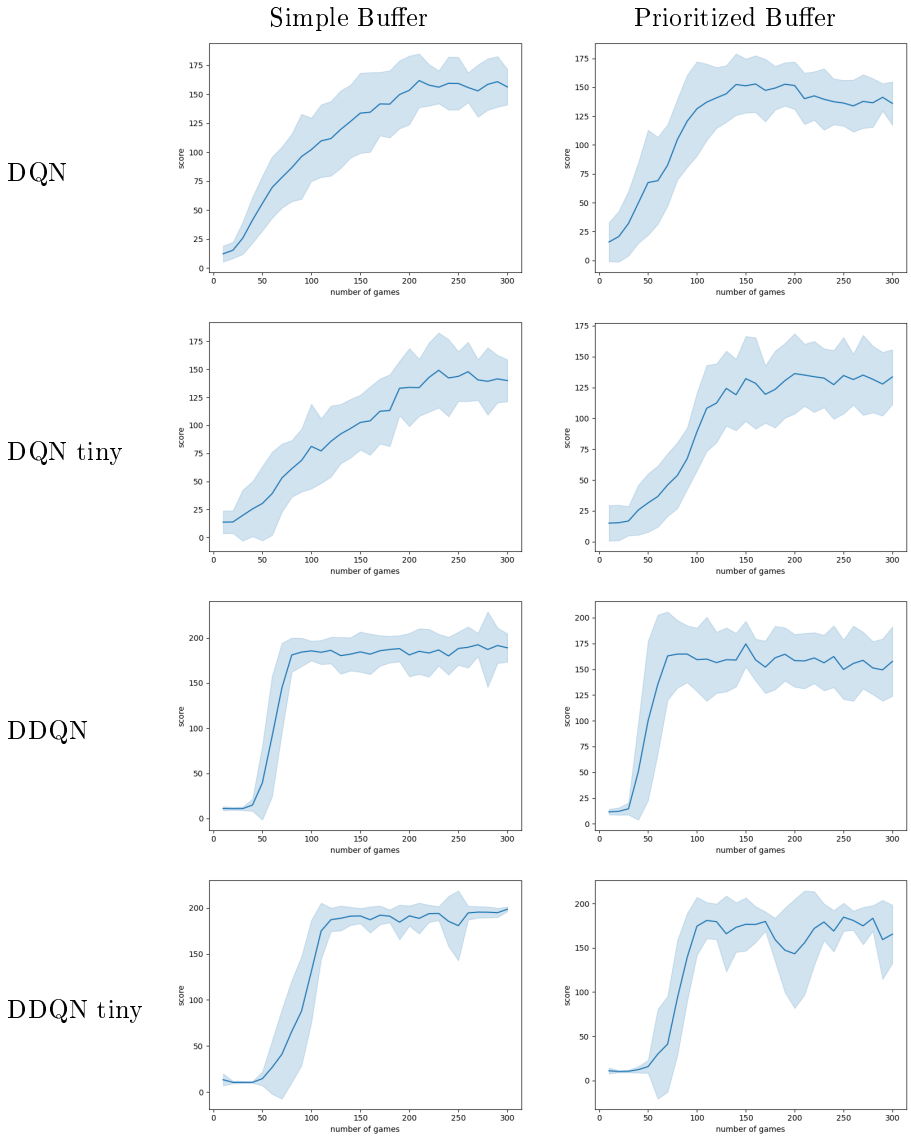
Simple Buffer jest kolejką o rozmiarze 10k z której próbki losowane są w sposób jednostajny. Prioritized Buffer jest implementacją Prioritized Experience Replay o tym samym rozmiarze co Simple Buffer. Dla zwiększenia wydajności implementacja bufferu oparta jest na drzewie binarnym w którym w liściach przechowywane są wagi elementów pamięci a wartości pozostałych wierzchołków są równe sumie wartości ich dzieci.
Pozostałe hiperparametry są wspólne dla wszystkich algorytmów.

- batch_size = 32 - wielkość losowanej próbki
- epsilon = 0.1 - prawdopodobieństwo wybrania losowej akcji
- gamma = 0.99 - współczynnik dyskontowania
- tau = 100 - czas interakcji po którym następuje aktualizacja target network dla agentów z dodatkową siecią neuronową
- alpha = 0.6 - poziom priorytezacji dla agentów z Prioritized Buffer




Rysunek 2. Mediany krzywych uczenia dla rozważanych agentów

Z rysunków (2) i (3) można wywnioskować, że agenci korzystający z target network osiągają lepsze rezultaty oraz ich krzywe uczenia mają mniejszą wariancję. Buffer z priorytetyzacją powoduje szybsze uczenie na początku jednak w późniejszym czasie interakcji nie ma większego znaczenia. Obydwie rozważane sieci neuronowe dają zbliżone wyniki z niewielką przewagą większej sieci.




Rysunek 3. Uśredniona krzywa uczenia wraz z odchyleniem standardowym

ROZDZIAŁ 3

# Algorytmy TRPO oraz PPO

Rozdział zawiera wyjaśnienie podstaw algorytmu *Trust Region Policy Optimization* [8] oraz *Proximal Policy Optimization*. Pokazana zostanie implementacja algorytmu PPO.

Strategie $\pi$ będą odtąd indeksowane za pomocą parametrów $\{\theta \in \Theta\}$. Mogą być nimi na przykład wagi sieci neuronowej. W dalszej części pracy będziemy rozważali strategie, które są różniczkowalne względem $\theta$. Funkcją korzyści strategi $\pi_\theta$ nazwijmy funkcję

$$A_{\pi_\theta} : \ S \times A \to \mathbb{R}$$
$$A_{\pi_\theta}(s,a) = Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s)$$

Wartość funkcji korzyści oznacza jak bardzo opłaca się wybrać akcję $a$ w stanie $s$ niż wybrać akcję zgodnie ze strategią $\pi_\theta$. Niech $\theta_t$ będzie parametrem strategii w czasie $t$.

### Trust Region Policy Optimization

Celem agenta jest znalezienie parametru $\theta$ maksymalizującego wartość przyszłych nagród

$$J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right]$$

gdzie $\tau$ jest interakcją $s_0, a_0, r_0, s_1, \ldots$ otrzymaną w wyniku stosowania strategii $\pi_\theta$. Rozważmy strategie zależne od parametrów $\theta$ oraz $\theta'$. Zauważmy, że

$$\begin{aligned}
\mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi_{\theta'}}(s_t,a_t)\right] &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t\left(Q_{\pi_{\theta'}}(s_t,a_t) - V_{\pi_{\theta'}}(s_t)\right)\right] \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t\left(r_t + V_{\pi_{\theta'}}(s_{t+1},a_t) - V_{\pi_{\theta'}}(s_t)\right)\right] \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right] - \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[V_{\pi_{\theta'}}(s_0)\right] \\
&= J(\theta) - J(\theta')
\end{aligned}$$

Niech $d_\pi$ będzie rozkładem na przestrzeni stanów

$$d_\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\mathbb{P}(s_t = s|\pi)$$

Wtedy zachodzi

$$\begin{aligned} J(\theta) - J(\theta') &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi_{\theta'}}(s_t,a_t)\right] \\ &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_\theta}\\ a\sim\pi_\theta}}\left[A_{\pi_{\theta'}}(s,a)\right] \\ &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_\theta}\\ a\sim\pi_{\theta'}}}\left[\frac{\pi_\theta(s,a)}{\pi_{\theta'}(s,a)}A_{\pi_{\theta'}}(s,a)\right] \end{aligned} \tag{4}$$

W pracy [**9**] wykazano, że (4) można przybliżyć licząc wartość oczekiwaną na rozkładzie $d_{\pi_{\theta'}}$

$$J(\theta) - J(\theta') \approx \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_{\theta'}}\\ a\sim\pi_{\theta'}}}\left[\frac{\pi_\theta(s,a)}{\pi_{\theta'}(s,a)}A_{\pi_{\theta'}}(s,a)\right]$$

Dla parametrów $\theta', \theta$ zdefiniujmy funkcję

$$\mathcal{L}(\theta',\theta) = \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_{\theta'}}\\ a\sim\pi_{\theta'}}}\left[\frac{\pi_\theta(s,a)}{\pi_{\theta'}(s,a)}A_{\pi_{\theta'}}(s,a)\right]$$

Ideą algorytmu *Trust Region Policy Optimization* jest takie przekształcanie parametru $\theta_t$, żeby $\theta_{t+1}$ z jednej strony maksymalizowało wartość funkcji $\mathcal{L}(\theta_t,\cdot)$ jednocześnie nie różniąc się zbyt mocno od $\theta_t$. Żeby to osiągnąć stosowana jest miara odległości rozkładów prawdopodobieństwa - dywergencja Kullbacka-Leiblera.

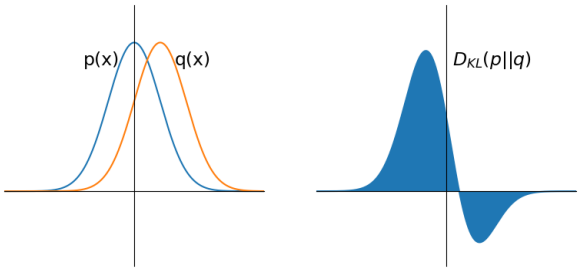
### Dywergencja Kullbacka-Leiblera

Chcąc sprawdzić odległość dyskretnego rozkładu $q$ od zadanego dyskretnego rozkładu $p$ niech

$$D_{KL}(p \mid\mid q) = \sum_i p(i)\log\frac{p(i)}{q(i)}$$

Jeżeli rozkłady $p$ i $q$ są ciągłe to $D_{KL}(p \mid\mid q)$ definiujemy jako $\int_{-\infty}^{\infty} p(x)\log\frac{p(x)}{q(x)}dx$.

Przykład odległości pomiędzy rozkładem Gaussa $\mathcal{N}(0,2)$ a $\mathcal{N}(2,2)$.




W powyższym przypadku $D_{KL}(p \mid\mid q) \approx 0.5$.

Funkcja $\pi_\theta(s,\cdot)$ jest rozkładem na przestrzenii akcji. Dla danego $\theta$ zdefinujmy

$$\overline{D}_{KL}(\theta \mid\mid \theta_t) = \mathop{\mathbb{E}}_{s\sim\pi_{\theta_t}} [D_{KL}(\pi_\theta(s,\cdot) \mid\mid \pi_{\theta_t}(s,\cdot))]$$

Pilnując, żeby $\overline{D}_{KL}(\theta \mid\mid \theta_t) < \delta$, dla odpowiedniego $\delta > 0$, dwie kolejne strategie nie będą się zbytnio różniły.
W każdej iteracji algorytm TRPO stara się znaleźć

$$\theta_{t+1} = \text{argmax}_\theta \mathcal{L}(\theta_t, \theta) \quad \text{dla} \quad \overline{D}_{KL}(\theta \mid\mid \theta_t) < \delta \tag{5}$$

## Aproksymacje $\mathcal{L}$ oraz wartości dywergencji

Żeby było możliwe zaimplementowanie powyższego rozumowania stosuje się przybliżenia funkcji $\mathcal{L}^{\theta_t} := \mathcal{L}(\theta_t, \cdot)$ oraz $\overline{D}^{\theta_t}_{KL} := \overline{D}_{KL}(\theta \mid\mid \theta_t)$ za pomocą rozwinięcia w szereg Taylora w $\theta_t$.

$$\mathcal{L}^{\theta_t}(\theta) \approx \mathcal{L}^{\theta_t}(\theta_t) + \nabla_\theta \mathcal{L}^{\theta_t}(\theta_t)^T(\theta - \theta_t) + \cdots$$

$$\overline{D}^{\theta_t}_{KL}(\theta) \approx \overline{D}^{\theta_t}_{KL}(\theta_t) + \nabla_\theta \overline{D}^{\theta_t}_{KL}(\theta_t)^T(\theta - \theta_t) + \frac{1}{2}(\theta - \theta_t)^T \nabla^2_\theta \overline{D}^{\theta_t}_{KL}(\theta)(\theta - \theta_t) + \cdots$$

Zarówno $\mathcal{L}^{\theta_t}$, $\overline{D}^{\theta_t}_{KL}$ jak i $\nabla_\theta \overline{D}^{\theta_t}_{KL}$ są równe zero w $\theta_t$. Stosując optymalizację wypukłą dla (5) z powyższymi wartościami, obliczone jest ([**8**], dodatek C)

$$\theta_{t+1} \approx \theta_t + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g,$$

gdzie $g = \nabla_\theta \mathcal{L}^{\theta_t}(\theta_t)$ oraz $H$ jest hesjanem $\overline{D}^{\theta_t}_{KL}(\theta)$.

## Algorytm Proximal Policy Optimization

Problemem powyższego podejścia korzystającego z dywergencji KL jest skomplikowanie implementacji oraz złożoność obliczeniowa wyznaczania odwrotności hesjanu $H^{-1}$. Podejście algorytmu PPO jest nieco inne.
Zdefiniujmy funkcję

$$L(s,a,\theta_t,\theta) = \min\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_t}(s,a)} A_{\pi_{\theta_t}}(s,a), \text{clip}\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_t}(s,a)}, 1-\epsilon, 1+\epsilon\right) A_{\pi_{\theta_t}}(s,a)\right)$$

gdzie $\epsilon$ jest hiperparametrem a $\text{clip}(x,a,b)$ jest równe $a$ jeśli $x \le a$, $b$ jeśli $x \ge b$ oraz $x$ w pozostałych przypadkach. Celem PPO jest znalezienie $\theta_{t+1}$ maksymalizującego wartość

$$\mathcal{L}^{\text{clip}}(\theta_t, \theta) = \mathop{\mathbb{E}}_{a\sim\pi} [L(s,a,\theta_t,\cdot)]$$

Rozważmy sytację gdy funkcja korzyści jest nieujemna. Wtedy zachodzi

$$L(s,a,\theta_t,\theta) = \min\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_t}(s,a)}, 1+\epsilon\right) A_{\pi_{\theta_t}}(s,a)$$

Wraz ze wzrostem $\pi_\theta(s,a)$ rośnie cała wartość $L$ do momentu aż ułamek nie będzie równy $1+\epsilon$, wtedy dalszy wzrost nie będzie powodował polepszania się $L$. Analogicznie, dla ujemnej funkcji korzyści wartość $L$ wzrasta gdy $\pi_\theta(s,a)$ maleje. Dzieje się to do czasu aż ułamek nie będzie mniejszy niż $1-\epsilon$. Wynika stąd,

że nowa strategia nie będzie czerpać korzyści ze zbytniego odejścia od obecnej. Hiperparametr $\epsilon$ oznacza jak daleko pozwalamy na odejście nowej strategii od obecnej.

### Algorytm actor-critic

Dla danej strategii $\pi_\theta$ nieznana jest dokładna wartość funkcji korzyści. Można ją jednak przybliżać stosując strategię przez pewne *range* kroków. Wtedy

$$Q_{\pi_\theta}(s_t, a_t) \approx \sum_{t'=t}^{\text{range}} \gamma^{t'-t} r_{t'},$$

gdzie $r_{t'}$ są otrzymanymi nagrodami.

Do obliczenia $A_{\pi_\theta}$ potrzebna jest jeszcze wartość funkcji $V$, która może być aproksymowana przez sieć neuronową. Wtedy taki algorytm zawiera dwie sieci neuronowe i nazywa się algorytmem *actor-critic* ([**10**]). Jedna sieć (aktor) jest odpowiedzialna za przewidywanie akcji - uczy się optymalnej strategii. Celem drugiej sieci (krytyka) jest nauczenie się funkcji wartości.

**Algorytm 2** Proximal Policy Optimization

Inicjalizacja sieci neuronowej $nn_a$ z parametrami $\theta$ przewidującej optymalne akcje
Inicjalizacja sieci neuronowej $nn_c$ aproksymującej wartości $V$
Tablica $T$ do zapisywania interakcji
Pobranie stanu początkowego $s$
**for all** $1 \leq t \leq$ episodes **do** ▷ *episodes* - maksymalny czas interakcji
    $T \leftarrow []$
    **for all** $0 \leq k <$ range **do** ▷ Przybliżanie $A_{\pi_\theta}$
        $s_k \leftarrow s$
        Wykonaj akcję $a_k$ zgodną z $nn_a(s_k)$ oraz pobierz $r_k, s_{k+1}$
        $d_k = \text{dist}(nn_a(s_k))$ ▷ $d_k$ jest rozkładem akcji $nn_a(s_k)$
        $T$.push$(s_k, a_k, r_k, s_{k+1}, d_k)$
    **end for**
    **for all** $0 \leq i <$ batch_size **do**
        **for all** $0 \leq k <$ *range* **do**
            $\hat{A}(s_k, a_k) \leftarrow (\sum_{k=0}^{\text{range}-1} \gamma^k r_k) - nn_c(s_k)$
            $\text{ratio}(s_k, a_k) \leftarrow (\text{dist}(nn_a(s_k))/d_k)$
        **end for**
        $\mathcal{L}^{\text{clip}} = \text{mean}(\text{clip}(\text{ratio}, 1-\epsilon, 1+\epsilon) * \hat{A})$
        Zaktualizuj wagi sieci $nn_a$ i $nn_c$ zgodnie z optymalizacją względem $\mathcal{L}^{\text{clip}}$
    **end for**
    $s \leftarrow s_{\text{range}}$
**end for**

### Implementacja

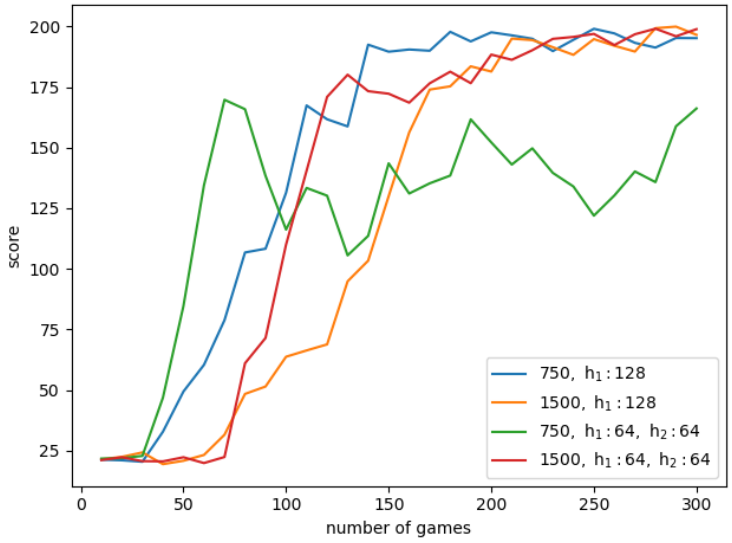
Do porównania wariantów algorytmu PPO zostanie wykorzystane środowisko *CartPole-v0* opisane w poprzednim rozdziale. Parametry od których zależny jest algorytm:

- gamma $= 0.99$ - współczynnik dyskontowania
- epsilon $= 0.1$ - hiperparametr użyty w *clip*
- range - liczba obserwacji na których aproksymowana jest wartość funkcji korzyści

Architektury sieci neuronowej krytyka oraz aktora będą takie same, za wyjątkiem liczby nauronów w warstwie wyjściowej - aktor będzie miał ich tyle ile jest akcji, a krytyk przewiduje tylko jedną wartość, która aproksymuje funkcję wartości $V$.

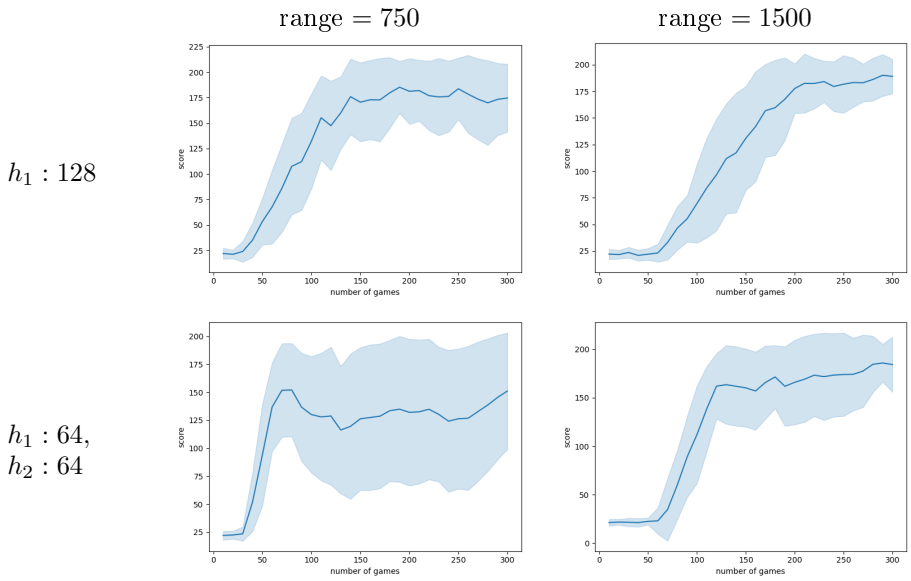
| Nazwa | range | Architektura sieci |
|---|---|---|
| 750, $h_1 : 128$ | 750 | $\{h_1 : 128\}$ |
| 750, $h_1 : 64$, $h_2 : 64$ | 750 | $\{h_1 : 64, h_2 : 64\}$ |
| 1500, $h_1 : 128$ | 1500 | $\{h_1 : 128\}$ |
| 1500, $h_1 : 64, h_2 : 64$ | 1500 | $\{h_1 : 64, h_2 : 64\}$ |

Porównywane są dwie architektury sieci neuronowych - z jedną warstwą ukrytą wielkośc 128 oraz z dwiema warstwami po 64 neurony.




Rysunek 1. Mediany krzywych uczenia dla agentów PPO

Na rysunkach (1) oraz (2) widać, że agentom z większą wartością *range* więcej zajmuje rozpoczęcie uczenia. Agent z dwoma ukrytymi warstwami oraz *range* równym 750 początkowo uczy się najszybciej, jednak ostatecznie okazuje się słabszy od pozostałych. Jego krzywa uczenia ma także bardzo dużą wariancję.




RYSUNEK 2. Uśrednione krzywe uczenia wraz z odchyleniem standardowym

ROZDZIAŁ 4

# Podsumowanie

Wszystkie zaproponowane algorytmy w rozważanym środowisku znalazły strategie dające sumę nagród bliską maksymalnej. W pracy [**7**] porównano warianty DQN i DDQN wraz z priorytetyzacją oraz ze zwykłą kolejką w różnych środowiskach. W niektórch z nich priorytetyzacja nie dała znaczącego polepszenia wyników. Wyniki otrzymane w tej pracy również nie wskazują na znaczącą przewagę priorytetyzacji.

Algorytmy DQN oraz PPO mają nieco inne przeznaczenie. Poprzez wykorzystanie pamięci w algorytmie DQN może on więcej czasu spędzić na swoje szkolenie korzystając z przeszłych doświadczeń jednocześnie ograniczając długość interakcji ze środowiskiem. Tego samego nie można powiedzieć o algorytmie PPO, który co prawda również zapamiętuje stany, jednak wykorzystuje je jedynie do przybliżania wartości funkcji korzyści. Wynika stąd, że ciężko porównać wydajność opisanych algorytmów gdyż uczenie w algorytmie DQN jest mniej zależne od obecnych interakcji.

# Bibliografia


[1] Silver, David, et al. *Mastering the game of Go with deep neural networks and tree search.* nature 529.7587 (2016): 484-489.

[2] Watkins, Christopher JCH, and Peter Dayan. *Q-learning.* Machine learning 8.3-4 (1992): 279-292.

[3] Schulman, John, et al. *Proximal policy optimization algorithms.* arXiv preprint arXiv:1707.06347 (2017).

[4] Brockman, Greg, et al. *Openai gym.* arXiv preprint arXiv:1606.01540 (2016).

[5] Van Hasselt, Hado. *Reinforcement learning in continuous state and action spaces.* Reinforcement learning. Springer, Berlin, Heidelberg, 2012. 207-251.

[6] Mnih, Volodymyr, et al. *Human-level control through deep reinforcement learning.* nature 518.7540 (2015): 529-533.

[7] Schaul, Tom, et al. *Prioritized experience replay.* arXiv preprint arXiv:1511.05952 (2015).

[8] Schulman, John, et al. *Trust region policy optimization.* International conference on machine learning. 2015.

[9] Achiam, Joshua, et al. *Constrained policy optimization.* Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017.

[10] Konda, Vijay R., and John N. Tsitsiklis. *Actor-critic algorithms.* Advances in neural information processing systems. 2000.