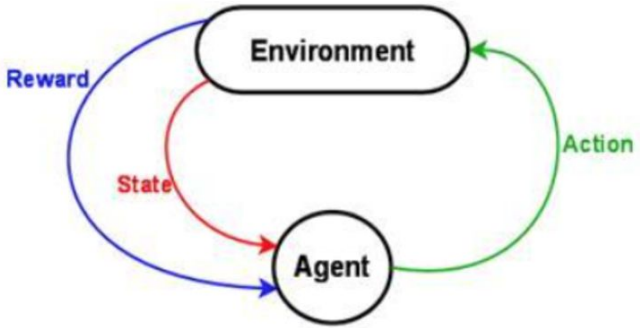
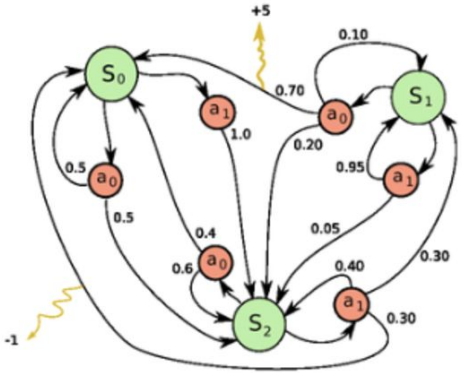
Uczenie Maszynowe

Wprowadzenie do Reinforcement Learning 2, 4/24/2017

Wprowadzenie

- W momencie gdy znamy graf stanów przejścia w naszym środowisku i odpowiadające im nagrody, problem da się rozwiązać używając klasycznych algorytmów





Notacje

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s, a_0 = a\right]$$

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s\right] \\ &= \mathbb{E}_{a\sim\pi}\left[Q^{\pi}(s,a)\right] \end{aligned}$$

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

Równanie Bellmana

$$Q^{\pi}(s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} [r_0 + \gamma V^{\pi}(s_1)]$$
$$= \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} [r_0 + \gamma \mathbb{E}_{a_1 \sim \pi} [Q^{\pi}(s_1, a_1)]]$$

Można pokazać, że jeśli zaczniemy od dowolnej inicjalizacji funkcji Q to wartości zbiegną do poprawnych wartości Q dla naszej funkcji π

Q-Learning

Zauważmy, że podobne rozumowanie powinno zadziałać nawet gdy nie znamy π, a zamiast tego będziemy wybierać najlepsze akcje zgodnie z funkcją Q!

$$Q^{\pi}(s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} \left[r_0 + \gamma \mathbb{E}_{a_1 \sim \pi} \left[Q^{\pi}(s_1, a_1) \right] \right]$$

zamieni się na:

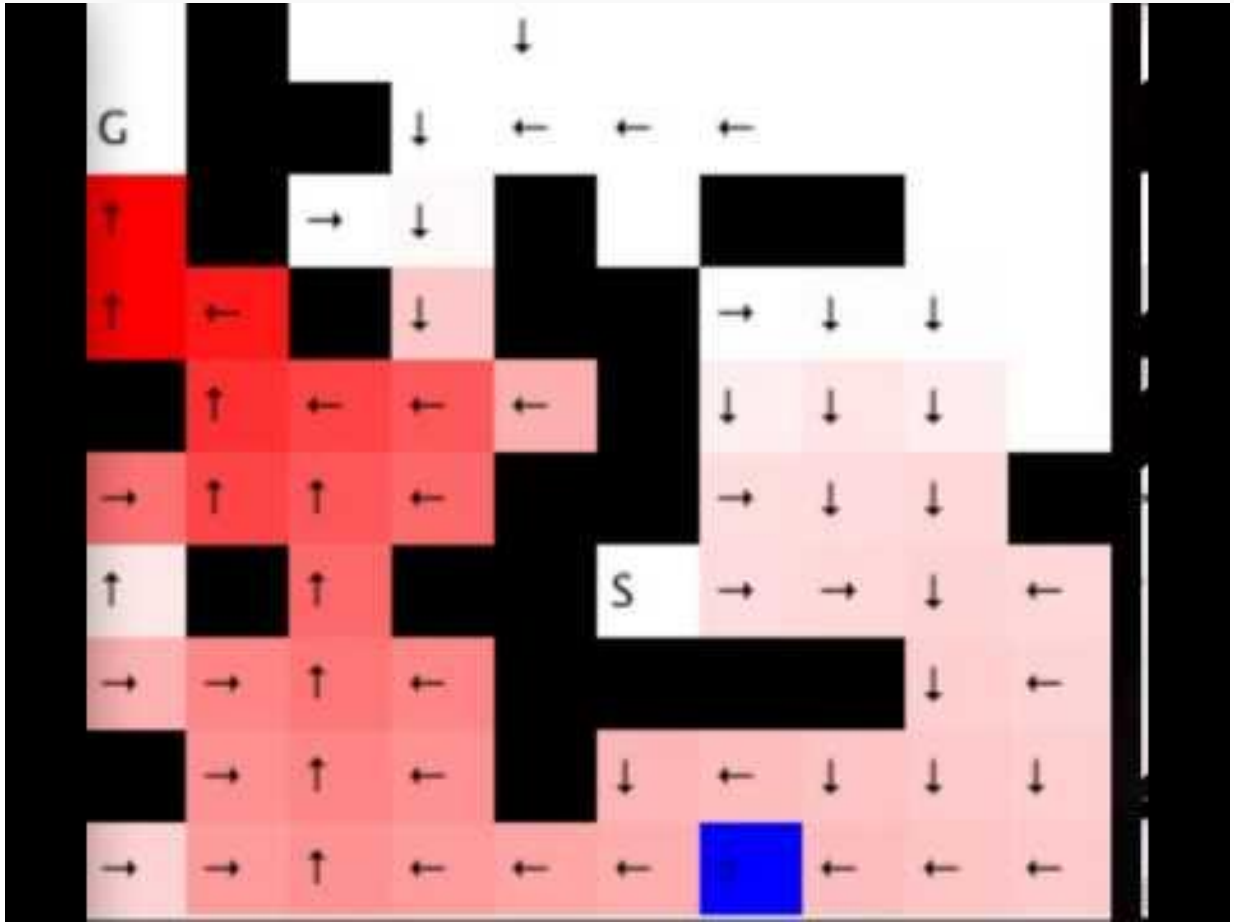
$$Q^{*}(s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} \left[r_0 + \gamma \max_{a_1} Q^{*}(s_1, a_1) \right]$$

(W praktyce możemy zignorować oczekiwaną wartość po s_1 i wciąż dostajemy niezbiasowany estymator prawej strony równania mając tylko pojedynczą próbkę)

Q-learning - algorytm dla problemów ze skończoną liczbą stanów

initialize $Q[num\_states, num\_actions]$ arbitrarily
observe initial state s
repeat
    select and carry out an action a
    observe reward r and new state s'
    $Q[s,a] = Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
    $s = s'$
until terminated

Przykład działania algorytmu



Dlaczego używamy Q zamiast V?

- Możemy obliczyć $\max_a Q(s_0, a)$ nie znając $P(s_1 \mid s_0, a)$
- Możemy wykonać update Q mając tylko (s, a, r, s')
- Możemy również wykonać update Q mając wcześniej zebrane dane!

Co z problemami z ogromną liczbą stanów?

Użyjmy sieci neuronowych do aproksymacji Q(s, a) !

Funkcja kosztu:

$$L = \frac{1}{2}[\underbrace{r + max_{a'}Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

Co z problemami z ogromną liczbą stanów?

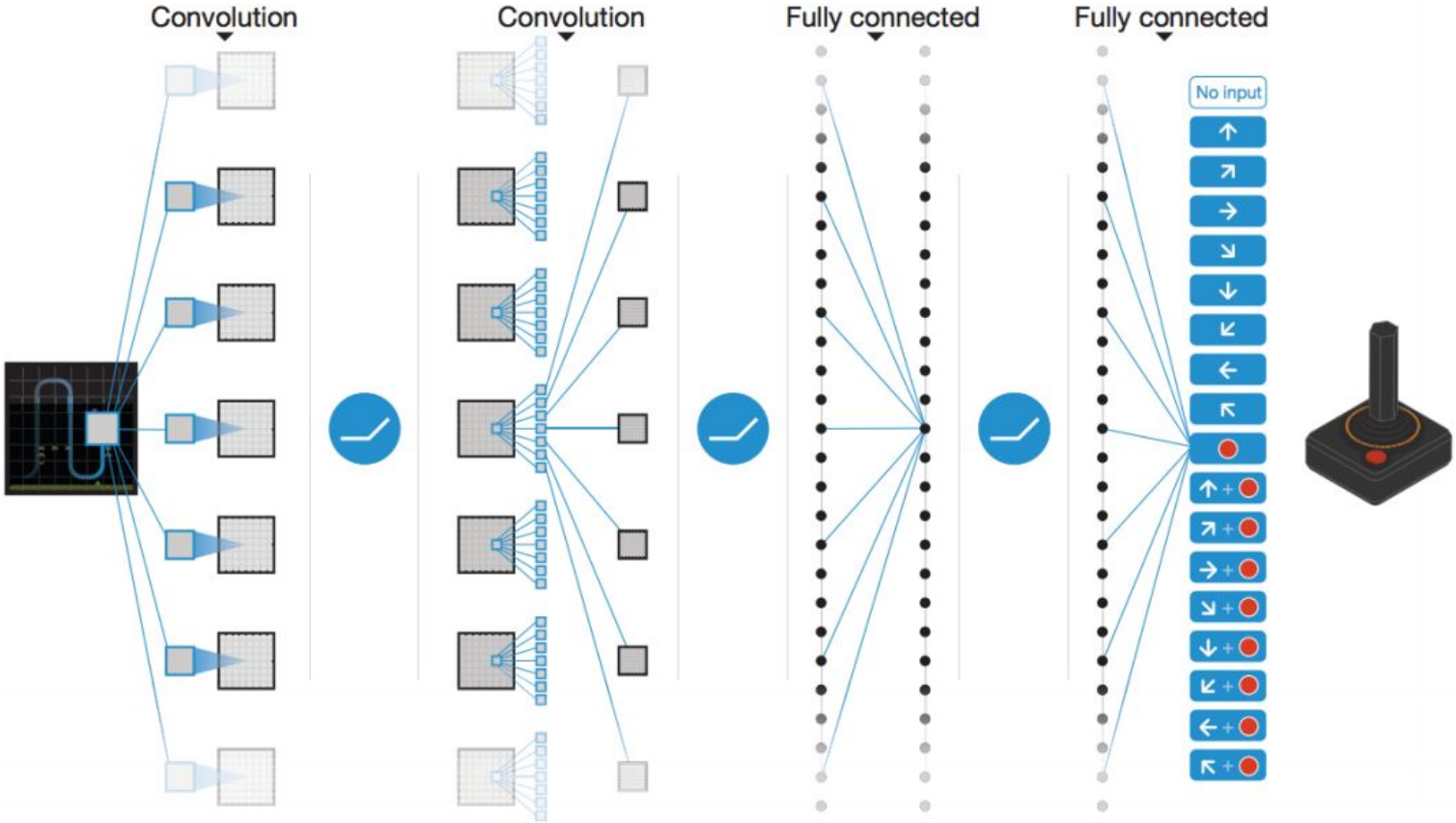
Użyjmy sieci neuronowych do aproksymacji Q(s, a) !

Funkcja kosztu:

$$L = \frac{1}{2}[\underbrace{r + max_{a'}Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

Niestety to nie działa w praktyce. Optymalizacja Q zmienia funkcję kosztu co powoduje, że trening jest bardzo niestabilny się rozbiega dość szybko. Będziemy potrzebowali kilku usprawnień.

DQN


Convolution
Convolution
Fully connected
Fully connected
No input

DQN - architektura

Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

DQN - algorytm

Initialize replay memory $\mathcal{D}$ to capacity N ← Zapamiętujemy ostatnie N interakcji

Initialize action-value function Q with random weights

for episode $= 1, M$ **do**

  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

  for $t = 1, T$ **do**

    With probability ϵ select a random action a_t

    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ ← Zamrożona wersja sieci Q

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$

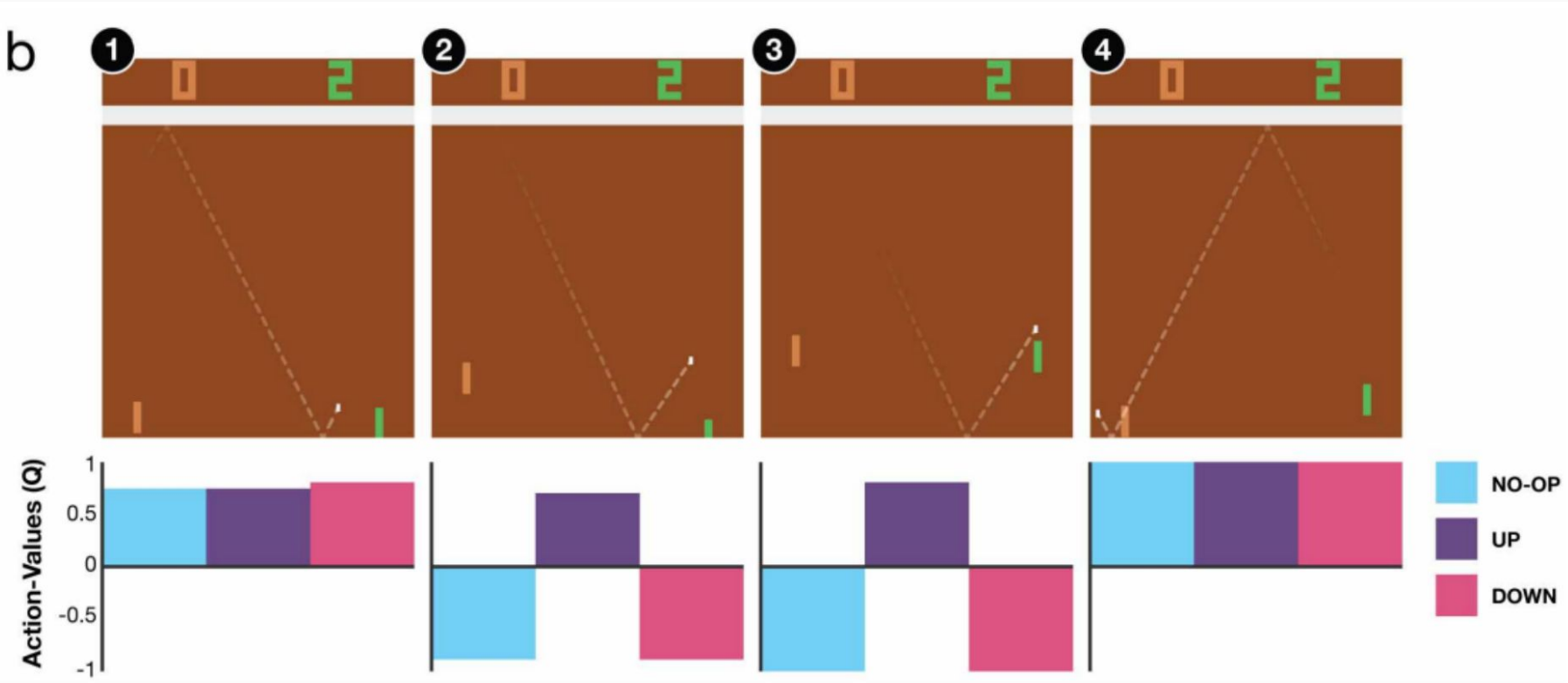
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

  end for

end for

Funkcja Q po zakończeniu treningu



Triki usprawniające DQN - Double Q-learning

Operator max powoduje, że wartości Q są przeszacowane:

$$\mathbb{E}_{X_1,X_2}\left[\max(X_1, X_2)\right] \geq \max(\mathbb{E}_{X_1,X_2}\left[X_1\right], \mathbb{E}\left[X_2\right])$$

Triki usprawniające DQN - Double Q-learning

Operator max powoduje, że wartości Q są przeszacowane:

$$\mathbb{E}_{X_1,X_2}\left[\max(X_1, X_2)\right] \geq \max(\mathbb{E}_{X_1,X_2}\left[X_1\right], \mathbb{E}\left[X_2\right])$$

Rozwiązanie - użyj 2 sieci neuronowych szacujących Q:

$$Q_A(s,a) \leftarrow r + \gamma Q(s', \arg\max_{a'} Q_B(s', a'))$$

$$Q_B(s,a) \leftarrow r + \gamma Q(s', \arg\max_{a'} Q_A(s', a'))$$

Double DQN

Standard DQN:

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q^{(\text{target})}(s', a')$$

$$Q(s,a) \leftarrow r + \gamma Q^{(\text{target})}(s', \arg\max_{a'} Q^{(\text{target})}(s', a'))$$

Double DQN:

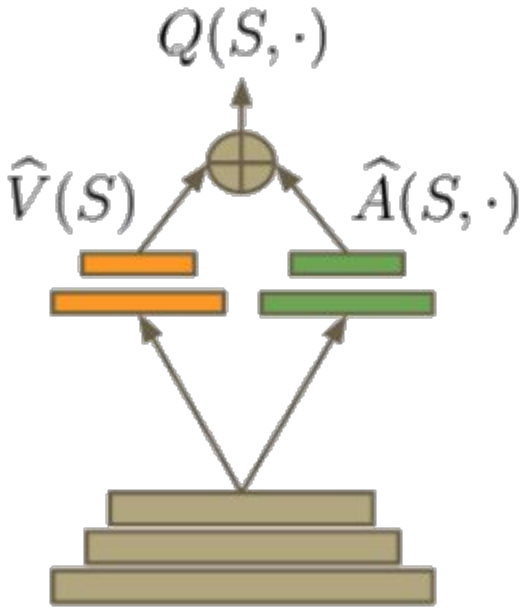
$$Q(s,a) \leftarrow r + \gamma Q^{(\text{target})}(s', \arg\max_{a'} Q(s', a'))$$

Dueling Networks

Można rozbić estymację do V i A:

- Mają one inna skalę wartości
- A nam wystarczy do podejmowania decyzji

$$Q(s,a;\theta,\alpha\,\beta) = \hat{V}\,(s;\theta,\beta) + \left(\hat{A}(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'}\hat{A}(s,a';\theta,\alpha)\right),$$



Prioritized Replay

- Można dodatkowo przyspieszyć trening przez wybieranie danych do optymalizacji w trochę sprytniejszy sposób
- Pomysł jest taki, żeby częściej losować przypadki, które miały duże wartości pochodnych

Prioritized Replay

- Można dodatkowo przyspieszyć trening przez wybieranie danych do optymalizacji w trochę sprytniejszy sposób
- Pomysł jest taki, żeby częściej losować przypadki, które miały duże wartości pochodnych
- W praktyce patrzymy na ostatnią wartość błędu estymacji $\delta_i = Q_\theta(s_i, a_i) - \hat{Q}_t$ i użyjemy kolejki priorytetowej do wyciągania kandydatów

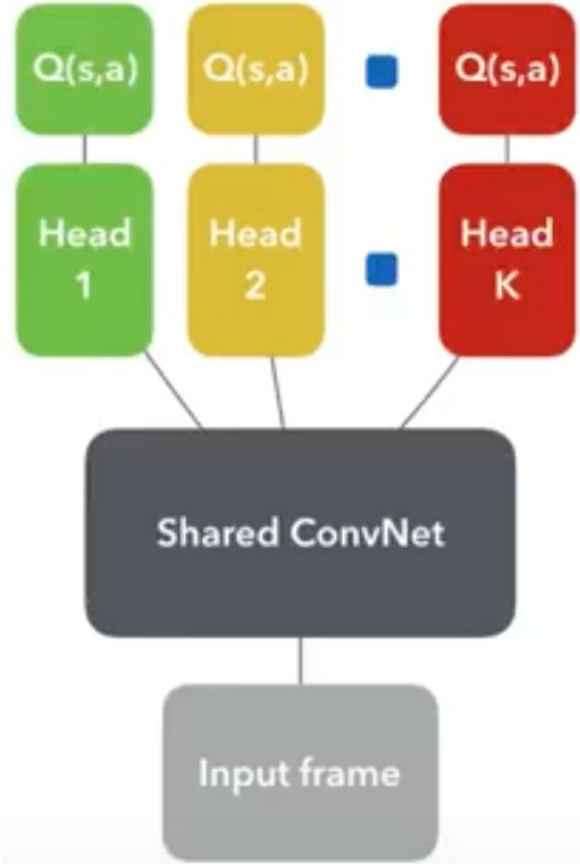
Ewaluacja na Atari

	30 no-ops		Human Starts	
	Mean	**Median**	**Mean**	**Median**
Prior. Duel Clip	**591.9%**	**172.1%**	**567.0%**	**115.3%**
Prior. Single	434.6%	123.7%	386.7%	112.9%
Duel Clip	**373.1%**	**151.5%**	**343.8%**	**117.1%**
Single Clip	341.2%	132.6%	302.8%	114.1%
Single	307.3%	117.8%	332.9%	110.9%
Nature DQN	227.9%	79.1%	219.6%	68.5%

Inne warianty - Bootstrapped DQN

- W tej wersji trenujemy K funkcji Q na różnych podzbiorach danych
- W każdym epizodzie wybieramy sobie jedną z nich i używamy do interakcji ze środowiskiem

- Takie podejście pozwala nam na szacowanie niepewności modelu (przez głosowanie)
- Funkcje Q reprezentują inne strategie i pozwalają na lepszą eksplorację



Bootstrapped DQN - demo