



POLITECHNIKA  
GDĄSKA



WYDZIAŁ ELEKTRONIKI,  
TELEKOMUNIKACJI  
I INFORMATYKI

# AITECH Sieci samouczące się

## Uczenie ze wzmocnieniem

Jerzy Dembski

24 stycznia 2024



Fundusze  
Europejskie  
Polska Cyfrowa



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz  
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

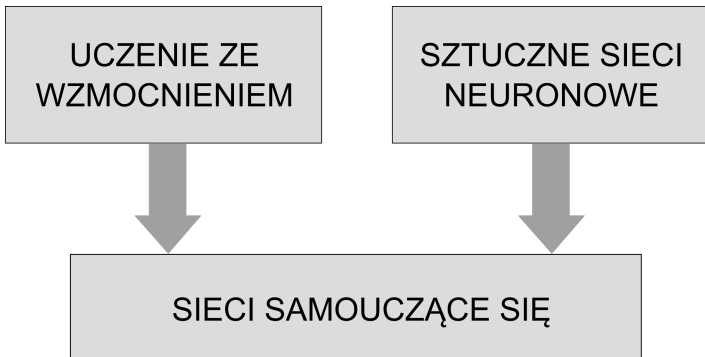
Tytuł projektu: Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)

- Przedstawienie zagadnień związanych z wieloetapowymi procesami decyzyjnymi oraz wyjaśnienie algorytmów uczenia ze wzmocnieniem jako metody szukania optymalnej strategii
- Omówienie wykorzystania sztucznej sieci neuronowej z głębokim uczeniem jako aproksymatora funkcji użyteczności i funkcji strategii

- Wieloetapowe procesy decyzyjne
- Typy procesów i środowisk
- Proces decyzyjny Markowa
- Programowanie dynamiczne i metody Monte Carlo
- Metoda różnic czasowych
- Eksploatacja i eksploracja
- Metody przyspieszania uczenia
- Kodowanie stanów i akcji
- Aproksymacja funkcji użyteczności i strategii
- Aproksymatory neuronowe + głębokie uczenie
- Przykładowe zastosowania

- Richard Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 2018.  
<http://incompleteideas.net/book/the-book-2nd.html>
- Paweł Cichosz, Systemy uczące się, Wydawnictwa Naukowo-Techniczne, Warszawa 2000, str. 712-792.
- Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare and Joelle Pineau (2018), "An Introduction to Deep Reinforcement Learning", Foundations and Trends in Machine Learning: Vol. 11, No. 3-4.
- DeepMind - artykuły, tutoriale, wykłady.

**Sieci samouczące się** są połączeniem metod uczenia ze wzmocnieniem z aproksymacją neuronową z możliwością wykorzystania technik głębokiego uczenia.



# Sieci samouczące się jako systemy uczenia z krytykiem



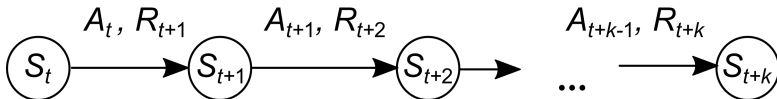
**Uczenie się systemu** - każda autonomiczna zmiana w systemie zachodząca na podstawie doświadczeń, która prowadzi do poprawy jakości jego działania.

**Rodzaje systemów uczących się** - ze względu na rodzaj i dostępność informacji ukierunkowującej:

- Uczenie z nauczycielem (informacja co zrobić w każdej sytuacji)
- Uczenie z krytykiem (informacja na ile system działa dobrze)
- Samoorganizacja (użycie heurystyk zamiast informacji ukierunkowującej np. zastosujemy to co zwykle działa w podobnych zadaniach)

- Symulowane wyżarzanie
- Algorytmy genetyczne
- Programowanie genetyczne
- Systemy klasyfikatorowe LCS/ XCS
- Metody roju cząstek - PSO

Procesy polegające na wielokrotnej interakcji systemu (ucznia) ze środowiskiem. W wyniku podjęcia jednej z możliwych akcji  $A_t$  w stanie  $S_t$ , system przechodzi do nowego stanu  $S_{t+1}$ , środowisko zwraca nagrodę  $R_{t+1}$

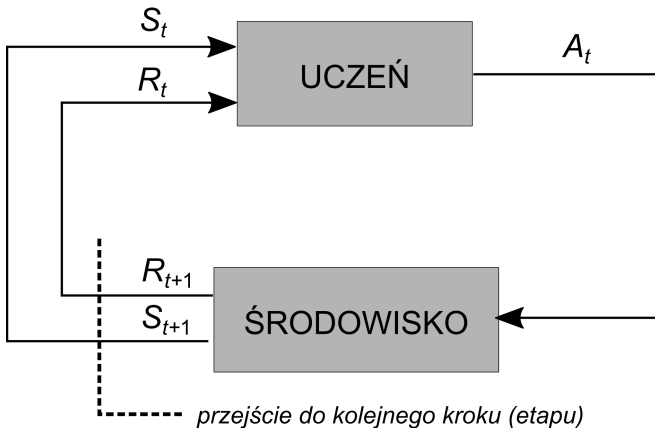


Uczenie dla wieloetapowych procesach decyzyjnych:

- Celem uczenia jest maksymalizacja użyteczności - ważonej średniej sumy nagród uzyskanych w ciągu całego procesu, niezależnie od stanu początkowego
- Wniosek: należy szukać optymalnej strategii (policy) zachowania ucznia (wyboru odpowiedniej akcji w każdym ze stanów)



Interakcja ucznia ze środowiskiem:



- Uczenie odbywa się na podstawie doświadczeń
- Akcje (decyzje) podejmowane w każdym kroku procesu mają dalekosiężne konsekwencje
- Czasami lepiej jest poświęcić najbliższą nagrodę lub ponieść karę, by osiągnąć lepsze skutki długofalowe (większą sumę nagród)
- Nagrody mogą być przypisane do poszczególnych stanów, do przejść pomiędzy stanami, do poszczególnych akcji w poszczególnych stanach lub, w przypadku najbardziej ogólnym, do akcji -  $a$  powodującej przejście pomiędzy stanami  $s$  i  $s'$  -  $r(s, a, s')$

## Przykłady:

- Gry planszowe np. tis-tac-toe, szachy
- Gry strategiczne czasu rzeczywistego (RTS)
- Kierowanie samochodem
- Sterowanie układem odwróconego wahadła na wózku
- Zarządzanie przedsiębiorstwem
- Modelowanie ruchu postaci ludzkiej wraz z indywidualizacją
- Sterowanie ruchem ulicznym w podejściu agentowym
- Życie

## Cechy środowiska:

- Przydziela nagrody i wyznacza bieżący stan
- Jest niezależne od ucznia, czyli oznacza wszystko to, na co uczeń nie ma wpływu

## Typy środowisk:

- Stacjonarne / niestacjonarne - zmienia się w czasie (np. zmiana reguł gry)
- Deterministyczne / niedeterministyczne - taka sama akcja może spowodować przejście do różnych stanów, a przy przejściu do takiego samego stanu można uzyskać różne nagrody z tym, że wartości oczekiwane nagród i prawdopodobieństwa przejść są stałe
- Niedeterministyczne o znanym / nieznanym modelu
- O parametrach ciągłych / dyskretnych (policzalnych)
- O pełnej informacji o stanie / o niepełnej informacji o stanie

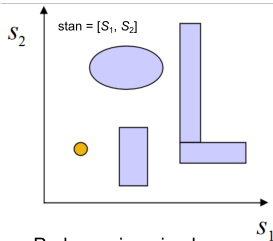
# Wieloetapowe procesy decyzyjne - typy procesów

Typy procesów (poza wynikającymi ze środowiska):

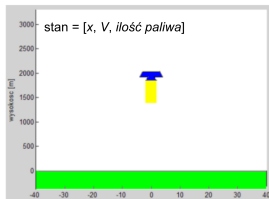
- Ze względu na liczbę etapów procesu: nieskończone / epizodyczne (kończące się po pewnej liczbie kroków)
- Ze względu na umiejscowienie nagród: tylko w stanach końcowych (terminalnych) / tylko w stanach pośrednich / w stanach końcowych oraz pośrednich
- Ze względu na wartość nagrody końcowej: procesy do sukcesu/ do porażki

# Przykładowe procesy o parametrach ciągłych

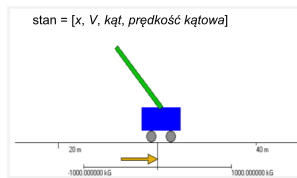
Robot wśród przeszkód



Lądownik

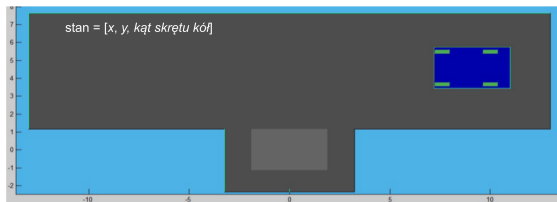


Odwrócone wahadło na wózku

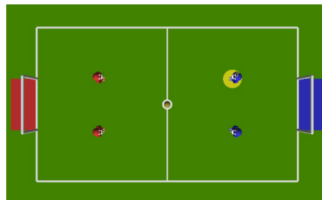


Parkowanie pojazdem

$S_1$



Piłka nożna 2x2



Użyteczność akcji to średnia ważona suma nagród, jaka jest otrzymywana po wykonaniu akcji  $a$  w stanie  $s$  do końca procesu (epizodu) przy danej strategii:

$$\begin{aligned} q_{\pi}(s, a) &= \mathbb{E}_{\pi}[\sum_{k=0}^n \gamma^k R_{t+k+1} | S_t = s, A_t = a] \\ &= \mathbb{E}_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^n R_{t+n+1}], \end{aligned}$$

gdzie  $R_{t+1}$  - nagroda bezpośrednio po wykonaniu akcji w kroku  $t$ ,  
 $\gamma$  - współczynnik dyskontowania,  $\pi$  - strategia.

W środowiskach deterministycznych często wykorzystywana jest użyteczność stanu  $v(s)$ , która jest użytecznością akcji należącej do danej strategii:

$$v_{\pi}(s) = q_{\pi}(s, a = \pi(s))$$

# Czas w wieloetapowych procesach decyzyjnych



W wielu praktycznych zastosowaniach poza sumą nagród, na użyteczność wpływa też czas, w którym nagrody są osiągane. Ze względu na skończoność życia czy niestacjonarność środowiska, nagrody zwykle opłaca się zdobywać jak najszybciej (zadania *do-sukcesu*), podczas gdy kary (nagrody ujemne) jak najdłużej odwlekać (zadania *do-porażki*).

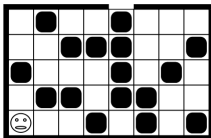
Jednym z rozwiązań jest użycie współczynnika dyskontowania  $\gamma \in [0, 1]$ , którego wartość jest zwykle nieco mniejsza od 1. W skrajnych przypadkach:

- gdy  $\gamma = 0$  system staje się „krótkowzroczny” - bierze pod uwagę tylko najbliższe nagrody,
- gdy  $\gamma = 1$  system uczy się traktowania wszystkich nagród jednakowo, bez względu na odległość czasową.



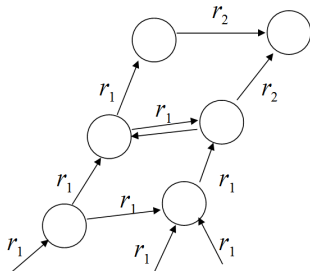
Poza uwzględnieniem czasu w definiowaniu użyteczności, współczynnik  $\gamma$  ma dodatkowe funkcje. Jedną z nich jest przyspieszenie osiągnięcia celów, nawet gdy użyteczność nagród za ich osiągnięcie nie zależy od czasu. W przykładowym labiryncie z przesuwными blokami wymuszenie jak najszybszego jego opuszczenia można uzyskać na 2 sposoby:

- przyjmując wysoką nagrodę za wyjście z labiryntu i karę (nagrodę ujemną) za każdą akcję, która nie powoduje wyjścia,
- przyjmując nagrodę za wyjście z labiryntu i  $\gamma < 1$ .



Należy przy tym pamiętać o tym, że zmiana wartości  $\gamma$  może zmienić problem i optymalną strategię.

Niech  $r_2$  oznacza wartość nagrody za dojście do stanu końcowego,  
 $r_1$  - wartość nagrody dla pozostałych stanów,  $r_2 \geq r_1$ .



Suma nagród powinna być większa dla krótszej drogi:

$$\sum_{t=0}^{n-1} \gamma^t r_1 + \gamma^n r_2 > \sum_{t=0}^n \gamma^t r_1 + \gamma^{n+1} r_2 = \sum_{t=0}^{n-1} \gamma^t r_1 + \gamma^n r_1 + \gamma^{n+1} r_2,$$

stąd:

$$\gamma < 1 - \frac{r_1}{r_2}.$$

Proces decyzyjny Markowa można zdefiniować jako czwórkę  $(\mathbb{S}, \mathbb{A}, p, r)$ ,  
gdzie:

$\mathbb{S}$  - skończony zbiór stanów,

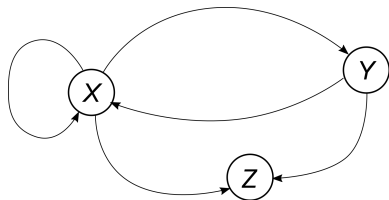
$\mathbb{A}$  - skończony zbiór akcji,

$p(s'|s, a)$  - stałe prawdopodobieństwo przejść pomiędzy  
stanami  $s$  i  $s'$  na skutek wykonania akcji  $a$ ,

$r(s, a) = \mathbb{E}[R|s, a]$  - stała oczekiwana wartość nagrody  
za wykonanie akcji  $a$  w stanie  $s$ .

Konsekwencją przyjętych założeń jest własność Markowa -  
prawdopodobieństwa przejść i średnie nagrody zależą tylko od bieżącego  
stanu bez względu na to, co działo się wcześniej:

$$\begin{aligned} p(s'|s, a) &= p(S_{t+1} = s' | S_t = s, A_t = a) \\ &= p(S_{t+1} = s' | S_t = s, A_t = a, S_{t-1}, A_{t-1}, \dots) \end{aligned}$$



prawdopodobieństwa przejść pomiędzy stanami w zależności od akcji:

$$P(X|X, A_1) = 0,5 \quad P(Y|X, A_1) = 0,2 \\ P(Z|X, A_1) = 0,3$$

$$P(X|X, A_2) = 0 \quad P(Y|X, A_2) = 0,5 \\ P(Z|X, A_2) = 0,5$$

$$P(X|X, A_3) = 0,1 \quad P(Y|X, A_3) = 0,6 \\ P(Z|X, A_3) = 0,3$$

$$P(X|Y, A_4) = 0,4 \quad P(Z|Y, A_4) = 0,6 \\ P(X|Y, A_5) = 0,7 \quad P(Z|Y, A_5) = 0,3$$

zbiór stanów:

$$\mathbb{S} = \{X, Y, Z\},$$

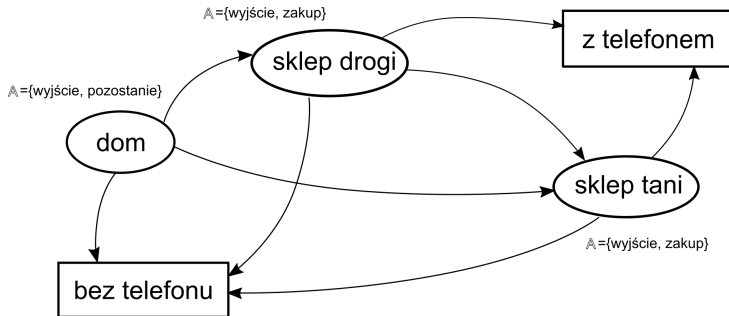
zbiór akcji:

$$\mathbb{A} = \{A_1, A_2, A_3, A_4, A_5\},$$

nagrody przypisane do przejść:

$$r(s, a, s') = \begin{cases} 1 & s' = Z \\ 0 & s' \neq Z. \end{cases}$$

# Przykład MDP - zakup telefonu



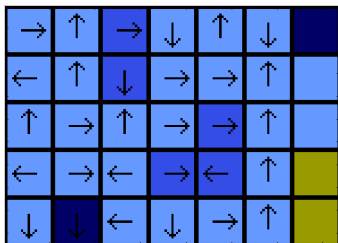
$P(\text{sklep drogi}|\text{dom}, \text{wyjście}) = 0,7$   
 $P(\text{sklep tani}|\text{dom}, \text{wyjście}) = 0,3$   
 $P(\text{bez telefonu}|\text{dom}, \text{pozostanie}) = 1,0$   
 $P(\text{bez telefonu}|\text{sklep drogi}, \text{wyjście}) = 0,4$   
 $P(\text{sklep tani}|\text{sklep drogi}, \text{wyjście}) = 0,6$   
 $P(\text{z telefonem}|\text{sklep drogi}, \text{zakup}) = 0,95$   
 $P(\text{sklep tani}|\text{sklep drogi}, \text{zakup}) = 0$   
 $P(\text{bez telefonu}|\text{sklep drogi}, \text{zakup}) = 0,05$   
 $P(\text{bez telefonu}|\text{sklep tani}, \text{wyjście}) = 1,0$   
 $P(\text{z telefonem}|\text{sklep tani}, \text{wyjście}) = 0$   
 $P(\text{bez telefonu}|\text{sklep tani}, \text{zakup}) = 0,1$   
 $P(\text{z telefonem}|\text{sklep tani}, \text{zakup}) = 0,9$

$R(\text{dom}, \text{bez telefonu}) = 0$   
 $R(\text{dom}, \text{sklep drogi}) = -1$   
 $R(\text{dom}, \text{sklep tani}) = -3$   
 $R(\text{sklep drogi}, \text{bez telefonu}) = -1$   
 $R(\text{sklep drogi}, \text{z telefonem}) = 5$   
 $R(\text{sklep drogi}, \text{sklep tani}) = -2$   
 $R(\text{sklep tani}, \text{bez telefonu}) = -3$   
 $R(\text{sklep tani}, \text{z telefonem}) = 10$

# Przykład MDP - żeglarz



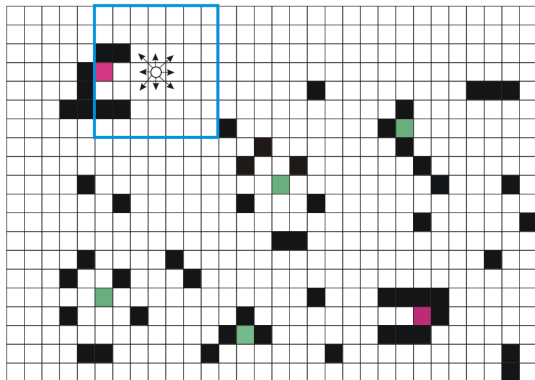
Problem żeglarza: żeglarz może się poruszać w 4 kierunkach: w lewo, w prawo, w górę lub w dół. Zaczyna w losowym wierszu pierwszej kolumny, by zakończyć epizod w ostatniej kolumnie. Jeśli trafi na plażę (żółte pole) - otrzyma nagrodę. Jeśli w trakcie epizodu trafi na ciemne pole, lub zderzy się ze ścianą - otrzyma karę. Problem może być niedeterministyczny, gdy istnieje prawdopodobieństwo  $0 < P < 1$  trafienia w inne pole, niż zgodne z wybranym kierunkiem ruchu.



W przykładowej strategii strzałkami oznaczono akcje w poszczególnych polach (stanach). Inny [przykład MDP](#).

Procesy, które nie są procesami decyzyjnymi Markowa:

- procesy niestacjonarne - o zmiennym środowisku np. gry z nieprzewidywalnym przeciwnikiem,
- procesy o niepełnej informacji o stanie (np. gdy agent widzi tylko fragment środowiska nie znając jego kompletnego stanu) - w takiej sytuacji, z punktu widzenia agenta, środowisko jest niestacjonarne,
- procesy o nieskończonej liczbie stanów lub akcji,
- procesy o parametrach ciągłych - stanów jest nieskończenie wiele i są niepoliczalne (np. zadanie stabilizacji odwróconego wahadła na wózku).



- pułapka
- ściana
- nagroda
- położenie agenta
- pole widzenia - stan obserwowany

Problem ANIMAT: stan obserwowany zawiera tylko niewielką część informacji o stanie rzeczywistym.



Dwa sposoby reprezentacji strategii:

- $\pi(s) = a$  oznacza akcję  $a$ , jaką należy wykonać w stanie  $s$ ,
- $\pi(a|s)$  oznacza prawdopodobieństwo wyboru akcji  $a$  w stanie  $s$  w ramach strategii mieszanej, co może być istotne w grach dwuosobowych lub w trakcie uczenia się. W tym przypadku wartość stanu dana jest wzorem:  $v_\pi(s) = \sum_a \pi(a|s)q_\pi(s, a)$ .

Poza strategią, która jest celem uczenia w MDP, konieczne jest przyjęcie strategii samego uczenia, która jest najczęściej połączeniem aktualnie wyuczonej strategii systemu z elementami eksploracji np. w postaci częściowo losowej metody wyboru akcji.

Strategia  $\pi'$  jest lepsza od strategii  $\pi$ , jeśli dla każdej pary  $(s, a)$ :

$$q_{\pi'}(s, a) \geq q_{\pi}(s, a),$$

oraz istnieje taka para  $(s, a)$ , że

$$q_{\pi'}(s, a) > q_{\pi}(s, a).$$

Strategia  $\pi^*$  jest optymalna, gdy dla każdej innej strategii  $\pi$  i dla każdego stanu  $s$ :

$$q_*(s, \pi^*(s)) \geq q_{\pi}(s, \pi(s)).$$

W problemach ze stacjonarnym środowiskiem zawsze istnieje co najmniej jedna strategia optymalna. W przypadku gier dwuosobowych (z adaptacją obu strategii) najczęściej można mówić jedynie o strategiach wzajemnie optymalnych lub o punktach równowagi.

- **Iteracja strategii** - początkowo wybieramy strategię losową, następnie wyznaczamy wartości użyteczności dla każdej pary (stan, akcja) dla tej strategii, następnie modyfikujemy strategię wybierając w poszczególnych stanach akcje o największych użytecznościach, następnie wyznaczamy wartości użyteczności dla każdej pary (stan, akcja) dla nowej strategii itd. Kończymy cały proces szukania, gdy aktualna strategia nie różni się od strategii z poprzedniej iteracji, co wskazuje na to, że jest ona strategią optymalną.

$$\pi_1 \rightarrow Q^{\pi_1} \rightarrow \pi_2 \rightarrow Q^{\pi_2} \rightarrow \dots \rightarrow \pi_N \rightarrow Q^{\pi_N} \rightarrow \pi_{N+1} = \pi_N = \pi^*$$

- **Iteracja wartości** - zaczynając od losowej tablicy  $Q$ , dla kolejnych lub losowo wybieranych par (stan akcja) obliczamy użyteczność stosując zachłanną metodę wyboru akcji polegającą na wyborze akcji o aktualnie największej estymowanej użyteczności.

W przypadku dysponowania tablicą lub funkcją użyteczności odpowiadającą strategii optymalnej, jej realizacją jest wybór akcji o największej użyteczności w każdym ze stanów.

W przypadku tablicy użyteczności akcji -  $q_*(s, a)$ :

$$\pi^*(s) = \operatorname{argmax}_a q_*(s, a).$$

W przypadku tablicy użyteczności stanów -  $v_*(s)$ :

$$\pi^*(s) = \operatorname{argmax}_a \sum_i p(s_i | s, a) [r(s, a, s_i) + \gamma v_*(s_i)].$$

# Metody szukania optymalnej strategii poprzez obliczanie wartości użyteczności



- Monte Carlo (MC)
- Programowanie dynamiczne (DP)
- Metoda różnic czasowych (TD)

Dla każdej pary (stan, akcja) wykonywana jest duża liczba  $L$  symulacji procesów z wykorzystaniem modelu środowiska i aktualnej strategii  $\pi$ , a następnie obliczana jest średnia suma nagród:

$$Q_{\pi}(s, a) = \frac{\sum_{e=1}^L \sum_{i=0}^{n_e} \gamma^i R_{e,i+1}}{L},$$

gdzie  $n_e$  - liczba kroków  $e$ -tego epizodu,  $R_{e,i+1}$  - nagroda w uzyskana po wykonaniu akcji w kroku  $i$   $e$ -tego epizodu.

Uwagi:

- nagrody mogą być zliczane również dla akcji innych niż początkowa w danym epizodzie,
- liczba  $L$  może być ustalona lub może wynikać z pomiarów dokładności przeprowadzanych co pewien czas,
- im większa liczba epizodów, tym lepsze przybliżenie wartości  $q$  lub  $v$ .

# Metoda Monte Carlo - iteracja strategii



$\pi$  – strategia - początkowo dowolna

**repeat**

$\pi_{pom} \leftarrow \pi$

**for all** para  $(s, a)$  **do**

**for all** dla każdego z  $L$  epizodów **do**

zainicjuj początkowy stan  $S = s$  i akcję  $A = a$

**while** stan  $S$  nie jest końcowy **do**

wykonaj akcję  $A$

obserwuj i zapamiętaj  $R, S', A' = \pi(S')$

$S \leftarrow S'$

$A \leftarrow A'$

**end while**

**end for**

$$Q_{\pi}(s, a) \leftarrow \frac{\sum_{e=1}^L \sum_{i=0}^{n_e} \gamma^i R_{e,i+1}}{L}$$

**end for**

**for all**  $s \in \mathbb{S}$  **do**

$\pi(s) \leftarrow \operatorname{argmax}_a Q_{\pi}(s, a)$

**end for**

**until**  $\pi = \pi_{pom}$

# Metoda Monte Carlo - iteracja wartości



$Q$  – tablica wartości akcji zainicjowana zerami

$\alpha$  – współczynnik zależny od numeru epizodu lub numeru wykonania akcji (np.  $\alpha = 1/N$ )

**repeat**

**for all** para  $(s, a)$  **do**

    zainicjuj początkowy stan  $S = s$  i akcję  $A = a$

**while** stan  $S$  nie jest końcowy **do**

      wykonaj akcję  $A$

      obserwuj i zapamiętaj  $R, S', A' = \operatorname{argmax}_x Q_\pi(S', x)$

$S \leftarrow S'$

$A \leftarrow A'$

**end while**

$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \sum_{i=0}^n \gamma^i R_{i+1}$

**end for**

**until** spełniony warunek końca

**for all**  $s \in \mathbb{S}$  **do**

$\pi(s) \leftarrow \operatorname{argmax}_a Q_\pi(s, a)$

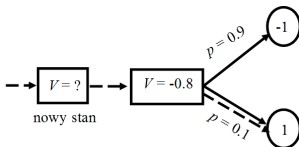
**end for**



- Uczenie modelu - rozkładów prawdopodobieństw przejść  $p$  i nagród  $r$ . Znając dokładny model, optymalną strategię można wyznaczyć metodą planowania wykorzystując np. programowanie dynamiczne (DP).
- Równoczesne uczenie strategii i funkcji użyteczności z wykorzystaniem techniki *Importance Sampling* poprzez używanie innej strategii podczas uczenia niż oceniana wraz z modyfikacją wartości zwrotu  $G$ .
- Symulacje Monte Carlo do wartościowania akcji w symulowanych środowiskach niestacjonarnych *Monte Carlo Tree Search* np. w grze GO.

## Wady:

- wymóg epizodyczności procesów,
- powolna zbieżność - obliczenie wartości użyteczności nowego stanu bez uwzględnienia wartości stanów następujących po nim (*bootstrapping*).



## Zalety:

- pewna zbieżność do wartości użyteczności  $v_{\pi}(s)$  lub  $q_{\pi}(s, a)$  dla ustalonej strategii przy odpowiedniej dużej liczbie epizodów,
- nie jest wymagana znajomość modelu środowiska (prawdopodobieństw przejść pomiędzy stanami i rozkładów wartości nagród).

Czasami wygodnie jest przedstawić ważoną sumę nagród w postaci zwrotu:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^n R_{t+n+1} = \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots + \gamma^{n-1} R_{t+n+1}) = \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned}$$

Użyteczność stanu, dla strategii  $\pi$  można przedstawić wzorem:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}] = \\ &= \sum_{s'} p(s'|s, a = \pi(s)) r(s, a, s') + \gamma \mathbb{E}_\pi[G_{t+1}] = \\ &= r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_\pi(s'), \end{aligned}$$

Jest to tzw. równanie równowagi Bellmana wiążące użyteczność stanu z użytecznościami stanów możliwych do osiągnięcia w kolejnym kroku.

$r(s, a)$  jest średnią nagrodą za wykonanie akcji  $a$  w stanie  $s$ ,  $p(s'|s, a)$  jest prawdopodobieństwem przejścia ze stanu  $s$  do  $s'$  po wykonaniu akcji  $a$ .

W przypadku użyteczności akcji, akcja  $a$  może być dowolną akcją stanie  $s$  – niekoniecznie należącą do strategii  $\pi$ , akcja  $a' = \pi(s')$  jest akcją należącą do strategii  $\pi$ :

$$\begin{aligned} q_{\pi}(s, a) &= r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_{\pi}(s') = \\ &= r(s, a) + \gamma \sum_{s'} p(s'|s, a) q_{\pi}(s', a') \end{aligned}$$

W przypadku, gdy znany jest model środowiska w postaci rozkładów  $p(s'|s, a)$  i  $r(s, a)$ , każdą wartość użyteczności można przedstawić w postaci równania liniowego. Mamy zatem tyle równań, ile jest niewiadomych.

Podczas realizacji strategii optymalnej wybierane są akcje o największej użyteczności:

$$\begin{aligned} v_*(s) &= r(s, a = \pi^*(s)) + \gamma \sum_{s'} p(s'|s, a = \pi^*(s)) v_*(s') = \\ &= \max_a [r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_*(s')]. \end{aligned}$$

Użyteczność dowolnej akcji  $a$  w strategii optymalnej zależy od użyteczności najlepszych akcji możliwych do wykonania w następnym kroku:

$$\begin{aligned} q_*(s, a) &= r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_*(s') = \\ &= r(s, a) + \gamma \sum_{s'} p(s'|s, a) \max_{a'} q_*(s', a'). \end{aligned}$$

- Programowanie dynamiczne polega na bezpośrednim wykorzystaniu równań Bellmana do obliczania użyteczności stanów lub akcji. Konieczna jest przy tym znajomość modelu środowiska w postaci rozkładów  $p(s'|s, a)$  i  $r(s, a)$ .
- Istnieją dwie metody wyznaczania optymalnej strategii:
  - iteracja strategii wykorzystująca równania równowagi Bellmana do wyznaczania użyteczności stanów lub akcji dla zadanej strategii
  - iteracja wartości wykorzystująca równania optymalności Bellmana do wyznaczania użyteczności stanów lub akcji dla strategii optymalnej.
- W przypadku iteracji strategii wartości użyteczności stanów lub akcji można obliczyć na dwa sposoby:
  - poprzez rozwiązanie układu  $N$  równań liniowych o złożoności obliczeniowej  $\mathcal{O}(N^3)$ , gdzie  $N$  jest liczbą możliwych stanów lub par (stan,akcja),
  - metodą iteracyjną.

# Programowanie dynamiczne - iteracja strategii



$\pi$  – strategia - początkowo dowolna

$\delta_{max}$  – dopuszczalny błąd iteracji

**repeat**

$\delta \leftarrow \delta_{max}$  – aktualny błąd iteracji

$V$  – tablica wartości stanów wypełniona zerami

$\pi_{pom} \leftarrow \pi$

**while**  $\delta \geq \delta_{max}$  **do**

$V_{pom} \leftarrow V$

$\delta \leftarrow 0$

**for all**  $s \in \mathbb{S}$  **do**

$a = \pi(s)$

$V(s) \leftarrow r(s, a) + \gamma \sum_{s'} p(s'|s, a) V_{pom}(s')$

$\delta \leftarrow \max(\delta, |V(s) - V_{pom}(s)|)$

**end for**

**end while**

**for all**  $s \in \mathbb{S}$  **do**

$\pi(s) \leftarrow \operatorname{argmax}_a [r(s, a) + \gamma \sum_{s'} p(s'|s, a) V(s')]$

**end for**

**until**  $\pi = \pi_{pom}$

# Programowanie dynamiczne - iteracja wartości



$\delta_{max}$  – dopuszczalny błąd iteracji

$\delta \leftarrow \delta_{max}$  – aktualny błąd iteracji

$V$  – tablica wartości stanów wypełniona zerami

**while**  $\delta \geq \delta_{max}$  **do**

$V_{pom} \leftarrow V$

$\delta \leftarrow 0$

**for all**  $s \in \mathbb{S}$  **do**

$V(s) \leftarrow \max_a (r(s, a) + \gamma \sum_{s'} p(s'|s, a) V_{pom}(s'))$

$\delta \leftarrow \max(\delta, |V(s) - V_{pom}(s)|)$

**end for**

**end while**

**for all**  $s \in \mathbb{S}$  **do**

$\pi(s) \leftarrow \operatorname{argmax}_a [r(s, a) + \gamma \sum_{s'} p(s'|s, a) V(s')]$

**end for**



## Wady:

- konieczna jest znajomość modelu środowiska –  $p(s'|s, a), r(s, a)$ , co jest raczej rzadkie w nietrywialnych problemach,
- w przypadku dużej liczby stanów – duża złożoność metody algebraicznej z uwagi na konieczność odwrócenia macierzy, w przypadku metody iteracyjnej – taki sam nakład obliczeń przy wyznaczaniu użyteczności każdego ze stanów bez względu na ich użyteczność.

## Zalety:

- pewna zbieżność do strategii optymalnej o ile problem nie jest osobiłowy (użyteczności nie zmierzają do nieskończoności) i w przypadku metod iteracyjnych przy odpowiedniej dokładności obliczania estymowanych wartości  $v$  lub  $q$ ,
- możliwość ograniczenia obliczeń do wyznaczania użyteczności stanów – znając model środowiska, użyteczność akcji można obliczyć ze wzoru  $Q(s, a) = r(s, a) + \gamma \sum_{s'} p(s'|s, a) V(s')$ .

Porównanie różnych sposobów reprezentowania użyteczności akcji w metodzie Monte Carlo i w programowaniu dynamicznym dla strategii  $\pi$ :

$$\begin{aligned} q_{\pi}(s, a) &= \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a] \quad (\text{Monte Carlo}) \\ &= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a], \\ q_{\pi}(s, a) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \quad (\text{DP}). \end{aligned}$$

Metoda różnic czasowych (*Temporal Differences*) łączy cechy metody Monte Carlo i programowania dynamicznego w kierunku poprawy szybkości uczenia i większej elastyczności procesu uczenia (uczenie *on-line*).

Modyfikacja użyteczności stanu lub akcji odbywa się w każdym kroku epizodu na podstawie różnicy pomiędzy estymowanym bieżącym zwrotem -  $\hat{G}_t$  a dotychczasową użytecznością  $V(S_t)$  lub  $Q(S_t, A_t)$  - również estymowaną:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\hat{G}_t - Q(S_t, A_t)],$$

gdzie  $\alpha$  jest współczynnikiem szybkości uczenia. Po uwzględnieniu  $\hat{G}_t = R_{t+1} + \gamma Q(S_{t+1}, a)$ , gdzie  $a$  jest kolejną wykonaną akcją ( $a = A_{t+1}$ ) lub akcją o największej użyteczności, modyfikacja przyjmuje postać:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, a) - Q(S_t, A_t)].$$

Taka estymacja wartości użyteczności na podstawie innych estymat nazywana jest w literaturze anglojęzycznej terminem *bootstrapping*.

# Metoda różnic czasowych - iteracja strategii



$\pi$  – strategia - początkowo dowolna

**repeat**

$\alpha$  – współczynnik szybkości uczenia

$Q$  – tablica wartości akcji wypełniona zerami

$\pi_{pom} \leftarrow \pi$

**for all** dla każdego epizodu **do**

zainicjuj początkowy stan  $S$  i akcję  $A$

**while** stan  $S$  nie jest końcowy **do**

wykonaj akcję  $A$

obserwuj  $R, S', A' = \pi(S')$

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'$

$A \leftarrow A'$

**end while**

**end for**

**for all**  $s \in \mathbb{S}$  **do**

$\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a)$

**end for**

**until**  $\pi = \pi_{pom}$

Inicjacja początkowej pary  $(S, A)$  może być losowa lub polegająca na wyborze kolejnych par z list y wszystkich możliwych.

$\alpha$  – współczynnik szybkości uczenia

$Q$  – tablica wartości akcji wypełniona zerami

**for all** dla każdego epizodu **do**

zainicjuj początkowy stan  $S$

**while** stan  $S$  nie jest końcowy **do**

wybierz i wykonaj akcję  $A$  w stanie  $S$  zgodnie ze strategią  
uczenia opartą na  $Q$  np.  $\epsilon$ -zachłanną

obserwuj  $R, S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

**end while**

**end for**

# Algorytm *Q-learning* - dodatkowe informacje



- Wartość współczynnika  $\alpha$  powinna być dobrana odpowiednio do problemu. Zbyt duża jego wartość może być przyczyną niebezpieczeństwa algorytmu, co jest szczególnie ważne w przypadku stosowania aproksymatorów. Wartość  $\alpha$  może być też zmienna i zwykle jest zmniejszana w czasie uczenia zgodnie z założeniem, że w późniejszych etapach uczenia wartości użyteczności  $Q$  są bliskie rzeczywistym wartościom optymalnej strategii i zbyt duże zmiany mogą zdestabilizować proces uczenia.
- Algorytm *Q-learning* jest określany uczeniem *off-policy* ze względu na to, że użyteczność akcji  $Q(S, A)$  jest modyfikowana w oparciu o użyteczność akcji w następnym kroku o maksymalnej estymowanej wartości, a nie o wartość użyteczności akcji rzeczywiście wykonanej zgodnie ze strategią eksploracyjną. Przeciwnie odbywa się to w algorytmie *SARSA* - typu *on-policy* (następny slajd).

$\alpha$  – współczynnik szybkości uczenia

$Q$  – tablica wartości akcji wypełniona zerami

**for all** dla każdego epizodu **do**

zainicjuj początkowy stan  $S$

wybierz akcję  $A$  w stanie  $S$  zgodnie ze strategią uczenia opartą na  $Q$

**repeat**

wykonaj akcję  $A$  i obserwuj  $R, S'$

wybierz akcję  $A'$  w stanie  $S'$  zgodnie ze strategią uczenia

opartą na  $Q$  np.  $\epsilon$ -zachłanną

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$$

$$S \leftarrow S', A \leftarrow A'$$

**until** stan  $S$  jest końcowy

**end for**

- Pomimo pozornie niewielkiej zmiany, schemat algorytmu *SARSA* jest inny niż w przypadku *Q-learning*. Wynika to z konieczności wyboru akcji  $A'$  w stanie  $S'$  przed modyfikacją użyteczności  $Q(S, A)$ .
- W przypadku procesów MDP, przy odpowiednio dobranym współczynniku szybkości uczenia  $\alpha$  i odpowiednio dobranej eksploracji (kolejne slajdy), zarówno *Q-learning*, jak i *SARSA* są zbieżne do strategii optymalnej, jednak *SARSA* ma przewagę, gdy liczą się koszty uczenia. Weźmy pod uwagę problem żeglarza w wersji deterministycznej (każda akcja prowadzi do pola docelowego):  
*Q-learning* szybko zacznie uczyć się korzystania z najkrótszej ścieżki do największej nagrody, ponosząc przy tym kary wynikające z eksploracji lub niedokładności aktualnie wyuczonej strategii, podczas gdy algorytm *SARSA* będzie od początku uczył się omijania kar, by zminimalizować koszty uczenia.



# SARSA z modyfikacją o wartość oczekiwaną



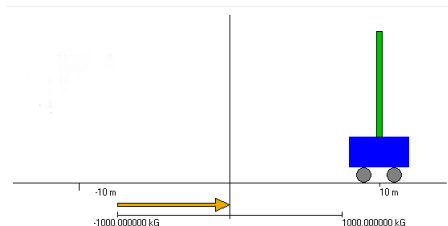
Algorytm uczenia jest prawie taki sam jak w metodzie **Q-learning**, a jedyną różnicą jest wzór na modyfikację wartości użyteczności:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1})] - Q(S_t, A_t)] \\ &= Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})[Q(S_{t+1}, a)] \\ &\quad - Q(S_t, A_t)], \end{aligned}$$


gdzie  $\pi(a|S_{t+1})$  - strategia uczenia wyrażona prawdopodobieństwem wyboru akcji  $a$  w stanie  $S_{t+1}$

Algorytm ten ma szybszą zbieżność nieco większym kosztem obliczeniowym.

- W algorytmach różnic czasowych polegających na interaktywnym uczeniu funkcji użyteczności i strategii w każdym kroku czasowym (metodą *on-line*), konieczna jest eksploracja przestrzeni stanów i akcji. W przypadku niewystarczającej eksploracji lub czystej eksploatacji polegającej na wyborze akcji o największych estymowanych użytecznościach, niektóre akcje, a nawet całe fragmenty przestrzeni stanów mogą nie zostać nigdy sprawdzone.
- Optymalizacja strategii uczenia polega na opracowaniu kompromisu pomiędzy eksploracją a eksploatacją, tak by z jednej strony zbadać całą przestrzeń stanów i akcji, a z drugiej strony skupić się na jej najbardziej obiecujących fragmentach.
- W przypadku występowania kosztów uczenia istotna może być optymalizacja funkcji wartości parametru losowości (np.  $\epsilon$  lub  $T$ ) w zależności od czasu uczenia. Przykładem jest problem wielorękiego bandyty.



Problem odwróconego wahadła na wózku: należy ustawić wózek w położeniu zerowym z wahadłem w pionie. Siła może być przyłożona z lewej lub prawej strony (jak na rysunku). Z której strony opłaca się przyłożyć siłę?

|  |  |  |    |   |    |
|--|--|--|----|---|----|
|  |  |  | -1 |   | 10 |
|  |  |  | -1 |   |    |
|  |  |  | -1 | -1  | -1 |
|  |  |  |    |  |    |
|  |  |  |    |   | 5  |

Problem żeglarza: w którym z 4 kierunków powinien próbować płynąć żeglarz, by zmaksymalizować sumę nagród? Zakładamy, że wszystkie pola w ostatniej kolumnie od lewej są końcowe.

Istnieją dwa główne typy strategii uczenia (strategii eksploracji):

- strategie nieukierunkowane polegające na losowaniu akcji ze z góry zadanego rozkładu prawdopodobieństwa,
- strategie ukierunkowane gdzie rozkład prawdopodobieństwa wyboru akcji zależy od dodatkowych parametrów związanych z postępami uczenia lub specyfiką zadania.

Dwie najczęściej stosowane strategie eksploracyjne:

- strategia  $\epsilon$ -zachłanna ( $\epsilon$ -greedy):  
z prawdopodobieństwem  $\epsilon$  wybieraj akcję w sposób losowy, a z prawdopodobieństwem  $1 - \epsilon$  akcję o aktualnie największej użyteczności,
- strategia losowa z użyciem rozkładu Boltzmana (*softmax*) - akcja wybierana jest z prawdopodobieństwem proporcjonalnym do jej aktualnej użyteczności zgodnie ze wzorem:

$$\pi_T(a_i|s) = \frac{\exp \frac{Q(s,a_i)}{T}}{\sum_j \exp \frac{Q(s,a_j)}{T}}.$$

# Strategie uczenia nieukierunkowane - dobór wartości parametrów



- Wartości parametrów  $\epsilon$  lub  $T$  mogą się zmieniać w czasie uczenia i zwykle na początku przyjmują maksymalne wartości umożliwiając dużą eksplorację, by w późniejszych etapach uczenia, dzięki zmniejszaniu ich wartości, wymuszać większą eksploatację obiecujących fragmentów przestrzeni stanów i akcji.
- W skrajnych przypadkach, gdy  $\epsilon = 1$  lub  $T \rightarrow \infty$  strategia uczenia jest czysto losowa a zachowanie agenta przypomina błędzenie losowe, gdy  $\epsilon = 0$  lub  $T \rightarrow 0$  agent wybiera akcje o aktualnie największej estymowanej użyteczności.

# Strategie uczenia ukierunkowane w metodach tablicowych



- Strategie licznikowe - polegające na zliczaniu liczb wykonań poszczególnych akcji w poszczególnych stanach, a następnie wykonywanie rzadko stosowanych akcji z nieco większym prawdopodobieństwem.
- Zainicjowanie tablicy  $Q$  największym możliwym zwrotem dla każdej akcji, co wymusza sprawdzenie wszystkich akcji (akcje, które nie zostały jeszcze sprawdzone będą miały przypisane zwykle większe wartości w tablicy  $Q$ ).
- Technika *Prioritized Sweeping* polegająca na przyjęciu pewnego progu różnicy aktualnej wartości  $Q(S_t, A_t)$  i jej estymaty  $\hat{G}_t$ , po którego przekroczeniu akcje z danego stanu, jak i stanu poprzedniego są częściej eksploatowane aż do uzyskania różnic poniżej progu.

- Ze względu na brak modelu środowiska (rozkładów prawdopodobieństw przejść pomiędzy stanami i bieżących nagród), wymagana jest eksploracja przestrzeni poszukiwań.
- Uczenie odbywa się podczas interakcji ucznia ze środowiskiem w sposób ciągły - możliwe jest zatem uczenie w środowiskach niestacjonarnych i podczas dowolnie długich (niekończących się) epizodów.
- Modyfikacja wartości użyteczności odbywa się na bieżąco na podstawie bieżących nagród i wartości użyteczności w kolejnych stanach (*bootstrapping*), co pozwala na połączenie dobrych cech metod MC i DP.
- Dzięki odpowiedniej strategii uczenia możliwe jest skupienie się na „obietujących” rejonach przestrzeni stanów i akcji.



**Aproksymacja funkcji**  $f(\mathbf{x})$  jest jej przedstawieniem w postaci modelu parametrycznego  $\hat{f}(\mathbf{x}, \mathbf{w})$  o odpowiednio dobranych parametrach wagowych modelu  $\mathbf{w} = [w_1, w_2, \dots, w_L]$ .

W problemach uczenia ze wzmocnieniem aproksymowane są najczęściej funkcje:

- funkcja użyteczności stanu –  $v(s)$  lub akcji –  $q(s, a)$ ,
- funkcja strategii –  $\pi(a|s)$ .

**Kodowanie stanów** – transformacja parametrów stanu do nowej przestrzeni cech:  $s \rightarrow \Phi(s) = [\phi_1(s), \phi_2(s), \dots, \phi_L(s)]$ .

Aproksymacja i kodowanie często stosowane są łącznie np. [aproksymacja liniowa](#) funkcji użyteczności z [kodowaniem pokryciowym](#) *tile-coding*.

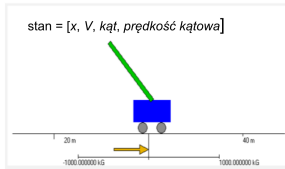
- Wielomiany - szeregi Taylora (ST)
- Reprezentacja harmoniczna - szeregi Fouriera (SF)
- Drzewa wyrażeń symbolicznych (DWS)
- Aproksymator liniowy (AL)
- Sieć o podstawie radialnej (RBF)
- Sztuczna sieć neuronowa (SSN) o architekturze klasycznej
- Głęboko uczona SSN zawierająca specjalizowane warstwy np. splotowe (DQN)

- Reprezentowanie ciągłych funkcji użyteczności lub funkcji strategii bez konieczności dyskretyzacji przestrzeni stanów i akcji (np. w zadaniu sterowania odwróconym wahadłem).
- Stała wielkość struktury i wektora parametrów w problemach dyskretnych o dużej liczbie możliwych stanów i akcji (np. w grach GO czy szachach).
- Możliwość uogólniania wiedzy poprzez wewnętrzną ekstrakcję istotnych cech stanu i agregację stanów o podobnym znaczeniu (można nauczyć system brania pod uwagę tylko istotnych cech stanu dla problemu, jak również uzyskiwać decyzje lub ocenę użyteczności w sytuacjach, które nie były brane pod uwagę podczas uczenia).

# Reprezentowanie stanów w problemach ciągłych

Wady dyskretyzacji:

Odwrócone wahadło na wózku



Piłka nożna 2x2



- zjawisko przekleństwa wymiarowości (*course of dimensionality*) - w przypadku dyskretyzacji - podziału dziedziny każdego z parametrów stanu na  $K$  przedziałów, liczba elementów tablicy (hiperkostek)  $K^{L_s}$  rośnie wykładniczo wraz z liczbą parametrów stanu  $L_s$ ,
- częściowa obserwowalność stanów - stany w reprezentacji ciągłej leżące na końcach przedziału traktowane są jako ten sam stan w reprezentacji dyskretnej.

Inne rozwiązania:

- aproksymacja,
- aproksymacja i kodowanie.

# Uczenie gradientowe aproksymatora funkcji użyteczności



Na początku konieczne jest zdefiniowanie funkcji straty lub celu uczenia. W najprostszym przypadku może nią być błąd średniokwadratowy pomiędzy rzeczywistą  $q_\pi(s, a)$  lub estymowaną akcją  $Q_\pi(s, a)$ , a wartością otrzymaną z aproksymatora  $\hat{Q}_\pi(s, a, \mathbf{w})$ :

$$J(\mathbf{w}) = (q_\pi(s, a) - \hat{Q}_\pi(s, a, \mathbf{w}))^2.$$

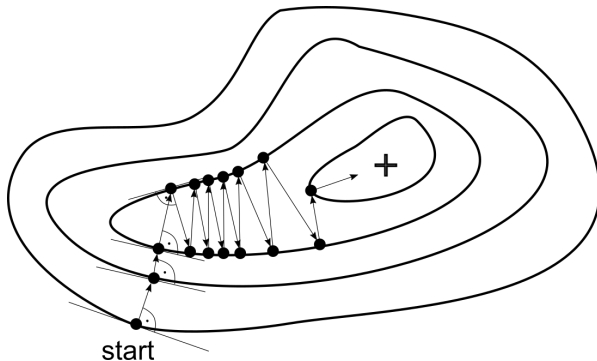
Gradient funkcji straty  $\nabla_{\mathbf{w}} J(\mathbf{w}) = \left[ \frac{\partial J(\mathbf{w})}{\partial w_1}, \frac{\partial J(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial J(\mathbf{w})}{\partial w_L} \right]$  wyznacza kierunek jej największego wzrostu, stąd w celu zmniejszenia  $J(\mathbf{w})$ , w najprostszym podejściu należy zmodyfikować wektor wag w kierunku przeciwnym:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(\mathbf{w}),$$

gdzie  $\alpha$  jest współczynnikiem szybkości uczenia.

Kierunek gradientu jest zawsze prostopadły do hiperpłaszczyzny stycznej do hiperpowierzchni będącej zbiorem punktów o tej samej wartości funkcji, co w punkcie, dla którego wyznaczany jest gradient.

W przypadku funkcji dwuwymiarowej kierunek gradientu jest prostopadły do stycznej do warstwy:



# Uczenie gradientowe - interpretacja gradientu



Gradient  $\nabla_{\mathbf{w}} J(\mathbf{w})$  jest w tym przypadku wektorem pochodnych cząstkowych funkcji straty po poszczególnych wagach aproksymatora. Można go przestawić za pomocą gradientu funkcji aproksymatora:

$$\begin{aligned}\nabla_{\mathbf{w}} J(\mathbf{w}) &= \nabla_{\mathbf{w}} (q_{\pi}(s, a) - \hat{Q}_{\pi}(s, a, \mathbf{w}))^2 \\ &= -2(q_{\pi}(s, a) - \hat{Q}_{\pi}(s, a, \mathbf{w})) \nabla_{\mathbf{w}} \hat{Q}_{\pi}(s, a, \mathbf{w}),\end{aligned}$$

stąd końcowy wzór na korektę wektora wag:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla_{\mathbf{w}} \hat{Q}_{\pi}(s, a, \mathbf{w}),$$

gdzie  $\delta = (q_{\pi}(s, a) - \hat{Q}_{\pi}(s, a, \mathbf{w}))$  jest błędem aproksymacji,  $\nabla_{\mathbf{w}} \hat{Q}_{\pi}(s, a, \mathbf{w})$  jest wektorem pochodnych cząstkowych funkcji aproksymacji użyteczności akcji względem wag w punkcie  $(s, a)$ .

- Aproksymacja funkcji  $q_\pi(s, a)$  nie ma praktycznego sensu z uwagi na to, że trzeba tę funkcję wcześniej wyznaczyć np. w postaci tablicowej.
- Dokładna postać funkcji  $q$  jest więc najczęściej zastępowana jej próbką w postaci bieżącej sumy nagród  $G_t$  (metody Monte Carlo), nieco lepszą estymatą  $R_{t+1} + \gamma \hat{Q}_\pi(s', a', \mathbf{w})$  w metodach TD lub wartością oczekiwaną w metodach DP.
- W przypadku metod TD wartość błędu  $\delta = R_{t+1} + \gamma \hat{Q}_\pi(s', a', \mathbf{w}) - \hat{Q}_\pi(s, a, \mathbf{w})$ .



- Uczenie ze wzmocnieniem (metody TD i DP) z aproksymacją funkcji użyteczności jest trudne, wartość docelowa w każdym kroku modyfikacji wag nie jest stała (tak jak w uczeniu z nauczycielem) ale jest oparta na estymowanych wartościach użyteczności (*semi-gradient*).
- Modyfikacja wag aproksymatora powoduje zmianę nie tylko wartości bieżącego stanu, ale również wszystkich pozostałych wartości stanów lub par (stan, akcja).
- Stabilność i skuteczność uczenia zależy w bardzo dużym stopniu od doboru parametrów uczenia, w tym szczególnie współczynnika  $\alpha$ . Zbyt duża wartość  $\alpha$  powoduje niestabilność procesu uczenia. Zbyt mała może być przyczyną utknięcia w minimum lokalnym funkcji błędu. Dobrym rozwiązaniem, ze względu na zbieżność, jest zmniejszanie wartości  $\alpha$  w trakcie uczenia.
- Nie ma uniwersalnych reguł doboru struktury aproksymatora, parametrów uczenia czy metody kodowania . Każde zadanie wymaga oddzielnej analizy!

# Obliczanie użyteczności w ciągłej wersji metody Monte Carlo



$\alpha$  – współczynnik szybkości uczenia

$\pi$  – określona strategia

$\hat{Q}(s, a, \mathbf{w})$  – aproksymowana funkcja użyteczności dla wstępnie zainicjowanego wektora wag  $\mathbf{w}$

**for all** dla każdego epizodu **do**

zainicjuj początkowy stan  $S_0$  i akcję  $A_0$  np. w sposób losowy  
przejdź epizod zapisując wszystkie stany, akcje i nagrody

**for all** dla każdego kroku epizodu  $t$  **do**

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G_t - \hat{Q}_\pi(S_t, A_t, \mathbf{w})] \nabla_{\mathbf{w}} \hat{Q}_\pi(S_t, A_t, \mathbf{w})$$

**end for**

**end for**

# Obliczanie użyteczności w ciągłej wersji metody TD(0)



$\alpha$  – współczynnik szybkości uczenia

$\pi$  – określona strategia

$\hat{Q}(s, a, \mathbf{w})$  – aproksymowana funkcja użyteczności dla wstępnie  
zainicjowanego wektora wag  $\mathbf{w}$

**for all** dla każdego epizodu **do**

zainicjuj początkowy stan  $S$

**repeat**

wykonaj akcję  $A$  i obserwuj  $R, S'$

wybierz akcję  $A'$  w stanie  $S'$  zgodnie ze strategią  $\pi$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma \hat{Q}_\pi(S', A', \mathbf{w}) - \hat{Q}_\pi(S, A, \mathbf{w})] \nabla_{\mathbf{w}} \hat{Q}_\pi(S, A, \mathbf{w})$

$S \leftarrow S', A \leftarrow A'$

**until** stan  $S$  jest końcowy

**end for**

# Ciągła wersja *Q-learning* - najprostszy wariant



$\alpha$  – współczynnik szybkości uczenia

$\hat{Q}(s, a, \mathbf{w})$  – aproksymowana funkcja użyteczności dla wstępnie  
zainicjowanego wektora wag  $\mathbf{w}$

**for all** dla każdego epizodu **do**

zainicjuj początkowy stan  $S$

**while** stan  $S$  nie jest końcowy **do**

wybierz i wykonaj akcję  $A$  w stanie  $S$  zgodnie ze strategią  
uczenia opartą na  $\hat{Q}(s, a, \mathbf{w})$  np.  $\epsilon$ -zachłanną

obserwuj  $R, S'$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \max_a \hat{Q}_\pi(S', a, \mathbf{w}) -$   
 $\hat{Q}_\pi(S, A, \mathbf{w})] \nabla_{\mathbf{w}} \hat{Q}_\pi(S, A, \mathbf{w})$

$S \leftarrow S'$

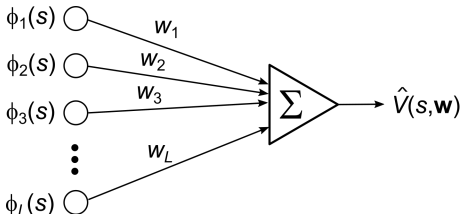
**end while**

**end for**

Aproksymacja użyteczności stanów:

$$\hat{V}(s, \mathbf{w}) = \Phi(s) \mathbf{w}^T = \sum_{i=1}^L \phi_i(s) w_i,$$

gdzie  $\Phi(s)$  - wektor cech stanu,  $\mathbf{w}$  - wektor parametrów aproksymatora (wag),  $L$  - liczba cech zakodowanej postaci wektora stanu.



Gradient względem wektora wag:

$$\nabla_{\mathbf{w}} \hat{V}(s, \mathbf{w}) = \Phi(s) = [\phi_1(s), \phi_2(s), \dots, \phi_L(s)]$$

Modyfikacja wektora wag podczas uczenia:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \Phi(s),$$

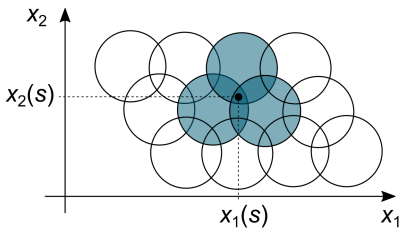
Aproksymacji użyteczności akcji  $Q(s, a)$  można dokonać na 2 sposoby:

- zakodowanie parametrów akcji wraz z parametrami stanu -  $\Phi(s, a)$ ,
- w przypadku niewielkiej liczby akcji - przydzielenie dla każdej akcji oddzielnego aproksymatora lub oddzielnego wyjścia (np. gdy aproksymatorem jest sieć neuronowa).

Liczba wag aproksymatora liniowego jest równa liczbie wejść. Z tego względu stosowany jest on najczęściej wraz kodowaniem binarnym pozwalającym zwiększyć liczbę wejść w stosunku do liczby parametrów stanu lub pary (stan, akcja) przy małym koszcie obliczeniowym.

- Zamiana parametrów stanu na postać lepiej odzwierciedlający zadanie, np. zamiana współrzędnych bezwzględnych na względne (np. problem drapieżców i ofiary).
- Dodanie dodatkowych parametrów stanu poprawiających uogólnianie (np. przewaga materialna czy rozwój w szachach).
- Kodowanie binarne - stosowane najczęściej wraz z aproksymatorem liniowym:
  - przybliżone (*coarse coding*),
  - metodą pokryć (*CMAC, tile coding*),
  - prototypowe z odległością metryczną,
  - prototypowe Kanervy (*Kanerva coding*).

W dwuwymiarowej przestrzeni parametrów stanu, gdzie  $\mathbf{x}(s) = [x_1(s), x_2(s)]$  jest wektorem parametrów stanu, każde pole (tutaj w kształcie okręgu) jest związane z jedną cechą binarną, równą 1 jeśli wektor parametrów stanu znajduje się wewnątrz pola.



Wektor cech w nowej przestrzeni:

$$\Phi(s) = \Phi(\mathbf{x}(s)) = [0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0],$$

biorąc pod uwagę kolejne pola w kolejnych wierszach.



- Poszczególne pola mogą mieć dowolny kształt (nie tylko kołowy) i zagęszczenie.
- Zagęszczenie pól może być zmienne w pierwotnej przestrzeni parametrów stanu  $x$  i zależeć np. od istotności fragmentu przestrzeni.
- Zwiększanie wielkości pól przy niezmiennym zagęszczeniu zwiększa stopień pokrywania się pól (*overlapping*) i tym samym pozwala na lepsze uogólnianie ale kosztem precyzji.
- W przypadku trójwymiarowej przestrzeni parametrów stanu cechy reprezentowane mogą być przez sfery, a w przypadku jeszcze większej liczby wymiarów przez hipersfery. Wraz z liczbą wymiarów liczba cech rośnie wykładniczo - zjawisko przekleństwa wymiarowości (*course of dimensionality*).
- Zalecana wartość współczynnika szybkości uczenia  $\alpha \sim \frac{1}{n}$ , gdzie  $n$  jest liczbą cech o wartości 1.

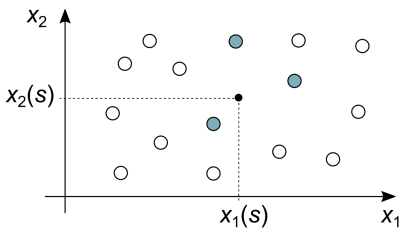
73 / 151

- Podobnie jak w przypadku kodowania przybliżonego, liczba cech rośnie wykładniczo wraz z liczbą wymiarów (parametrów pierwotnej przestrzeni cech).
- W porównaniu z kodowaniem przybliżonym, mniejszy jest koszt obliczeniowy wyznaczania wartości cech. Łatwiej jest też kontrolować szybkość uczenia, gdyż liczba cech o wartościach równych 1 jest zawsze równa liczbie pokryć.
- Zalecana wartość współczynnika szybkości uczenia  $\alpha \sim \frac{1}{n}$ , gdzie  $n$  jest liczbą pokryć.

# Kodowanie prototypowe z odległością metryczną



W przypadku kodowanie prototypowe nie występuje problem przekleństwa wymiarowości, gdyż w przestrzeni parametrów stanu rozmieszczana jest z góry ustalona liczba stanów prototypowych odpowiadających cechom binarnym.



Cecha przyjmuje wartość 1, jeśli jej prototyp znajduje się dostatecznie blisko wektora parametrów stanu w sensie odległości euklidesowej lub innej. W wersji zaawansowanej prototypy mogą być przemieszczane w kierunku miejsc, dla których potrzebna jest większa dokładność, dzięki większemu zagęszczeniu stanów prototypowych.

Stany prototypowe, jak i każdy inny stan przedstawione są w postaci pośrednich wektorów cech  $\Psi(\mathbf{x}(s))$  (najczęściej binarnych). Cała sekwencja przekształceń dla aktualnego stanu  $s$ :

$$s \rightarrow \mathbf{x}(s) \rightarrow \Psi(\mathbf{x}(s)) \rightarrow \Phi(s)$$

Każdy aktualny stan w postaci  $\Psi(\mathbf{x}(s))$  jest porównywany do prototypów pod względem stopnia zgodności (np. odwrotność odległości Hamminga). W nowej przestrzeni cech, których liczba odpowiada liczbie prototypów,  $n$ -ta cecha przyjmuje wartość 1, gdy stopień zgodności wektora cech stanu z  $n$ -tym prototypem przekracza ustaloną wartość progową.

Przykładowe stany prototypowe:

```
0001010100101110
1100101101010001
0011010100001100
1010111000111101
1111111110000011
0000000000001111
0000000000000001
```

przykładowy stan:

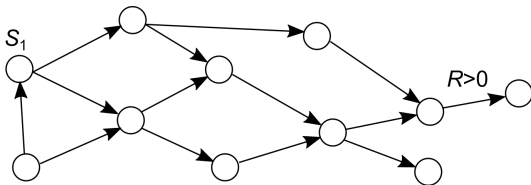
```
0001010100001110
```

próg zgodności = 80%

wektor cech w nowej  
przestrzeni:

$$\Phi(s) = [1, 0, 1, 0, 0, 0, 0]$$

Zastosowanie przybliżonego zwrotu TD(0):  $\hat{G}_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$  wymaga wielu epizodów by powiązać akcję (np. w stanie  $S_1$ ) z oddaloną w czasie nagrodą  $R$ .



Ślady aktywności (*eligibility traces*) pozwalają na szybszą propagację informacji o nagrodach oddalonych w czasie przy znacznie większym koszcie obliczeniowym w metodach tablicowych i nieznacznie większym w przypadku zastosowania aproksymacji funkcji użyteczności.

Inne przybliżenia zwrotu:

$$\begin{aligned}\hat{G}_{t:t+2} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2}) \\ \hat{G}_{t:t+3} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 Q(S_{t+3}, A_{t+3}) \\ &\dots \\ \hat{G}_{t:t+n} &= R_{t+1} + \gamma R_{t+2} + \dots + \gamma^n Q(S_{t+n}, A_{t+n}).\end{aligned}$$

Możliwe jest stosowanie ważonych sum przybliżeń - przykładowo:

$$\hat{G} = 0,5\hat{G}_t + 0,5\hat{G}_{t:t+2}$$

Można też zastosować sumę ważoną przybliżeń przyjmując, że bliższe przybliżenia są bardziej istotne:

$$\hat{G}_t^\lambda = (1 - \lambda) \sum_{i=1}^n \lambda^{i-1} \hat{G}_{t:t+i},$$

gdzie  $\lambda$  jest **współczynnikiem świeżości**,  $1 - \lambda$  jest czynnikiem normalizującym, by przy  $n \rightarrow \infty$  znormalizowana suma wag dążyła do 1.

# Ślady aktywności - wyznaczenie poprawki 1/2



Wyznaczenie poprawki wartości użyteczności dla akcji  $A_t$ , która przyczyniła się do uzyskania nagrody  $R_{t+n}$  i dojścia do stanu  $S_{t+n}$ :

$$\begin{aligned}\hat{G}_t^\lambda = & (1 - \lambda)\lambda^0[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})] + \\ & + (1 - \lambda)\lambda^1[R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2})] + \\ & + \dots + \\ & + (1 - \lambda)\lambda^{n-1}[R_{t+1} + \gamma R_{t+2} + \dots + \gamma^n Q(S_{t+n}, A_{t+n})],\end{aligned}$$

sumując elementy w kolumnach i uwzględniając  $\lim_{n \rightarrow \infty} (1 - \lambda) \sum_{i=1}^n \lambda^{i-1} \rightarrow 1$ , otrzymujemy:

$$\begin{aligned}\hat{G}_t^\lambda \cong & (\gamma\lambda)^0[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - \gamma\lambda Q(S_{t+1}, A_{t+1})] + \\ & + (\gamma\lambda)^1[R_{t+2} + \gamma Q(S_{t+2}, A_{t+2}) - \gamma\lambda Q(S_{t+2}, A_{t+2})] + \\ & + \dots + \\ & + (\gamma\lambda)^n[R_{t+n+1} + \gamma Q(S_{t+n+1}, A_{t+n+1}) - \gamma\lambda Q(S_{t+n+1}, A_{t+n+1})].\end{aligned}$$



# Ślady aktywności - wyznaczenie poprawki 2/2



Do obu stron równania dodajemy  $-Q(S_t, A_t)$  oraz przesuwamy ostatnią kolumnę o jeden element w dół, by w puste miejsce w pierwszym wierszu i ostatniej kolumnie wstawić  $-Q(S_t, A_t)$ :

$$\begin{aligned}\hat{G}_t^\lambda - Q(S_t, A_t) &\cong (\gamma\lambda)^0[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] + \\ &\quad + (\gamma\lambda)^1[R_{t+2} + \gamma Q(S_{t+2}, A_{t+2}) - Q(S_{t+1}, A_{t+1})] + \\ &\quad + \dots + \\ &\quad + (\gamma\lambda)^n[R_{t+n+1} + \gamma Q(S_{t+n+1}, A_{t+n+1}) - Q(S_{t+n}, A_{t+n})] + \\ &\quad - (\gamma\lambda)^{n+1}Q(S_{t+n+1}, A_{t+n+1}) \\ &\cong \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \delta_k,\end{aligned}$$

gdzie  $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$  jest błędem estymacji w kroku  $t$ .  
Do każdej wartości użyteczności wcześniej wykonanej akcji można więc dodać poprawkę z kroku  $t$ :

$$Q(S_{t-k}, A_{t-k}) \leftarrow Q(S_{t-k}, A_{t-k}) + \alpha(\gamma\lambda)^k \delta_t$$

# Ślady aktywności - wyznaczenie wartości $Q$ met. tablicową $TD(\lambda)$



Wyznaczenie wartości akcji -  $Q$  przy zadanej strategii  $\pi$  w wersji tablicowej:

$\alpha$  – współczynnik szybkości uczenia

$Q$  – tablica wartości akcji wypełniona zerami

**for all** dla każdego epizodu **do**

$e$  – tablica mnożników poprawek wypełniona zerami

zainicjuj początkowy stan  $S$  i akcję  $A$

**while** stan  $S$  nie jest końcowy **do**

wykonaj akcję  $A$

obserwuj  $R, S', A' = \pi(S')$

$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

$e(S, A) \leftarrow e(S, A) + 1$

**for all** wszystkie wykonane akcje w tym epizodzie  $(s, a)$  **do**

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$

$e(s, a) \leftarrow \gamma \lambda e(s, a)$

**end for**

$S \leftarrow S'$

$A \leftarrow A'$

**end while**

**end for**

# Ślady aktywności - *SARSA* w wersji tablicowej



$\alpha$  – współczynnik szybkości uczenia

$Q$  – tablica wartości akcji wypełniona zerami

**for all** dla każdego epizodu **do**

$e$  – tablica mnożników poprawek wypełniona zerami

zainicjuj początkowy stan  $S$

wybierz akcję  $A$  w stanie  $S$  zgodnie ze strategią uczenia opartą na  $Q$

**repeat**

wykonaj akcję  $A$  i obserwuj  $R, S'$

wybierz akcję  $A'$  w stanie  $S'$  zgodnie ze strategią uczenia np.  $\epsilon$ -zachłanną  
opartą na  $Q$

$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

$e(S, A) \leftarrow e(S, A) + 1$

**for all** wszystkie wykonane akcje w tym epizodzie  $(s, a)$  **do**

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$

$e(s, a) \leftarrow \gamma \lambda e(s, a)$

**end for**

$S \leftarrow S', A \leftarrow A'$

**until** stan  $S$  jest końcowy

**end for**

W przypadku metody *Q-learning* i innych metod typu *off-policy* bezpośrednio stosowanie poprawek na więcej niż jeden krok wstecz obarczone jest błędem związanym z różnicą pomiędzy aktualną wyuczoną strategią  $\pi$  a strategią uczenia  $\pi_u$  np  $\epsilon$ -zachłanną. W celu zminimalizowania tego błędu, wartość  $\delta$  z bieżącego kroku jest dodatkowo przemnażana przez współczynnik uwzględniający różnice rozkładów prawdopodobieństw wyboru akcji w poszczególnych krokach (*importance sampling ratio*). Przykładowo błąd estymacji w kroku  $t + n$ :  $\delta_{t+n}$  jako poprawka użyteczności akcji w kroku  $t$  powinien być dodatkowo pomnożony przez współczynnik:

$$\rho_{t:t+n-1} = \prod_{k=t}^{t+n-1} \frac{\pi(A_k|S_k)}{\pi_u(A_k|S_k)} = \frac{\pi(A_t|S_t)}{\pi_u(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\pi_u(A_{t+1}|S_{t+1})} \cdots \frac{\pi(A_{t+n-1}|S_{t+n-1})}{\pi_u(A_{t+n-1}|S_{t+n-1})}$$

gdzie  $A_k$  jest akcją która została wykonana w kroku  $k$  w stanie  $S_k$ . W przypadku, gdy strategia wyuczona polega na wyborze jednej akcji z prawdopodobieństwem 1 (w stacjonarnych problemach MDP), współczynnik  $\rho$  często przyjmuje wartość 0 przy dużej eksploracji.

# Ślady aktywności - wersja z aproksymacją użyteczności



W przypadku aproksymacji funkcji użyteczności, wartości wszystkich wcześniejszych akcji epizodu nie muszą być każdorazowo modyfikowane w każdym kolejnym kroku, a w zamian za to jednorazowo modyfikowany jest wektor wag o ważoną sumę gradientów z poprzednich kroków epizodu:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \mathbf{z},$$

gdzie  $\delta$  jest błędem estymacji użyteczności akcji w bieżącym kroku  $t$ ,  $\mathbf{z}$  jest ważoną sumą gradientów modyfikowaną w każdym kroku epizodu:

$$\mathbf{z} \leftarrow \gamma \lambda \mathbf{z} + \nabla_{\mathbf{w}} \hat{Q}_{\pi}(S_t, A_t, \mathbf{w}),$$

Gradient można więc skojarzyć z adresem, na który będą przychodziły wypłata przyszłych nagród, a  $\mathbf{z}$  może być skojarzone z książką adresową uzupełnioną o rozkład zastęg.

# Obliczanie użyteczności w ciągłej wersji metody TD( $\lambda$ )



$\alpha$  – współczynnik szybkości uczenia

$\pi$  – określona strategia

$\hat{Q}(s, a, \mathbf{w})$  – aproksymowana funkcja użyteczności dla wstępnie zainicjowanego wektora wag  $\mathbf{w}$

**for all** dla każdego epizodu **do**

$\mathbf{z}$  – wyzerowany wektor ważonych gradientów

zainicjuj początkowy stan  $S$  i akcję  $A$

**repeat**

wykonaj akcję  $A$  i obserwuj  $R, S'$

wybierz akcję  $A'$  w stanie  $S'$  zgodnie ze strategią  $\pi$  i w oparciu o  $\hat{Q}$

$\mathbf{z} \leftarrow \gamma \lambda \mathbf{z} + \nabla_{\mathbf{w}} \hat{Q}(S, A, \mathbf{w})$

$\delta \leftarrow R + \gamma \hat{Q}(S', A', \mathbf{w}) - \hat{Q}(S, A, \mathbf{w})$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \mathbf{z}$

$S \leftarrow S', A \leftarrow A'$

**until** stan  $S$  jest końcowy

**end for**

## Zalety:

- przyspieszenie uczenia dzięki szybszej propagacji informacji o nagrodach w odległych stanach, zwłaszcza w przypadku aproksymacji funkcji użyteczności stanów lub par (stan, akcja),
- poprawa stabilności procesu uczenia w niektórych problemach (analogia do uczenia z momentem),
- płynna regulacja procesu uczenia pomiędzy TD( $\lambda = 0$ ) a metodą Monte Carlo - TD( $\lambda = 1$ ).

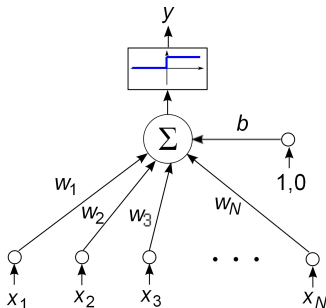
# Sztuczne sieci neuronowe (SSN) jako aproksymatory funkcji użyteczności



- Możliwość reprezentowania dowolnych funkcji ciągłych w przypadku architektury co najmniej 2-warstwowej z nieliniowymi funkcjami aktywacji.
- Tworzenie wewnętrznej reprezentacji stanu lub pary (stan, akcja) z ekstrakcją cech stanu lub selekcją parametrów istotnych dla określenia użyteczności.
- Uczenie metodą gradientową, a w przypadku sieci wielowarstwowych - z wykorzystaniem propagacji wstecznej błędu.
- Możliwość wykorzystania dobrze opracowanych technik regularyzacji poprawiających uogólnianie (uzyskanie poprawnych wartości wyjściowych dla nieznanych w trakcie uczenia stanów).
- Możliwość przenoszenia wiedzy pomiędzy podobnymi zadaniami (*transfer learning*, *multitask learning*).
- Możliwość wykorzystania technik głębokiego uczenia, w tym sieci z warstwami splotowymi (*convolutional layers*) do przetwarzania obrazów i dźwięków.



# Model neuronu z progową unipolarną funkcją aktywacji



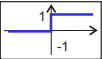
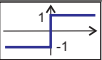
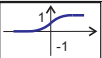
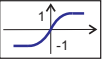
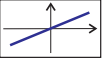
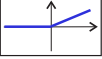
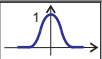
Funkcja, którą taka sieć reprezentuje, wyraża się wzorem:

$$f(\mathbf{x}) = y = f_{akt}(a) = \text{hardlim}\left(\sum_{i=1}^N w_i x_i + b\right) = \text{hardlim}(\mathbf{w}^T \mathbf{x} + b),$$

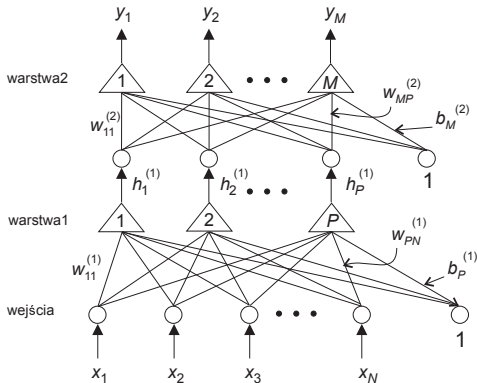
Uwaga: progowe funkcje aktywacji nie mogą być używane w przypadku uczenia gradientowego z uwagi na brak różniczkowalności w punkcie 0.

# Zestawienie funkcji aktywacji neuronu



| funkcja aktywacji<br>$f_{akt}$   | zakres wartości<br>na wyjściu | wykres<br>funkcji   | opis  |
|--|-------------------------------|---|---|
| model neuronu z iloczynem skalarnym $a = \mathbf{w}^T \mathbf{x} + b$          |                               |   |   |
| $\begin{cases} 0 & \text{gdy } a < 0 \\ 1 & \text{gdy } a \geq 0 \end{cases}$  | $y \in \{0, 1\}$              |  | funkcja progowa unipolarna                    |
| $\begin{cases} -1 & \text{gdy } a < 0 \\ 1 & \text{gdy } a \geq 0 \end{cases}$ | $y \in \{-1, 1\}$             |  | funkcja progowa bipolarna                     |
| $1/(1 + e^{-a})$   | $y \in (0, 1)$                |  | funkcja sigmoidalna unipolarna                |
| $2/(1 + e^{-2a}) - 1$  | $y \in (-1, 1)$               |  | funkcja sigmoidalna bipolarna                 |
| $a$  | $y \in (-\infty, \infty)$     |  | funkcja liniowa – identycznościowa            |
| $\begin{cases} 0 & \text{gdy } a < 0 \\ a & \text{gdy } a \geq 0 \end{cases}$  | $y \in (0, \infty)$           |  | funkcja ReLU ( <i>rectified linear unit</i> ) |
| model z miarą odległościową $a = \ \mathbf{w} - \mathbf{x}\  b$                |                               |   |   |
| $e^{-a^2}$   | $y \in (0, 1)$                |  | funkcja radialna                              |

# Dwuwarstwowa sztuczna sieć neuronowa



Funkcja reprezentowana przez  $i$ -te wyjście sieci:

$$y_i = f_{akt}^{(2)} \left( \sum_{j=1}^P w_{ij}^{(2)} f_{akt}^{(1)} \left( \sum_{k=1}^N w_{jk}^{(1)} x_k + b_j^{(1)} \right) + b_i^{(2)} \right)$$

# Uczenie sieci wielowarstwowych: algorytm propagacji wstecznej błędu 1/5



Za miarę błędu klasyfikacji przyjmuje się w klasycznych SSN błąd średniokwadratowy:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{\mu=1}^{K_u} \sum_{i=1}^M (d_{i\mu} - y_{i\mu})^2,$$

gdzie  $K_u$  jest liczbą przykładów do uczenia,  $M$  jest liczbą neuronów w warstwie wyjściowej, zaś  $d_{i\mu}$  jest pożądaną wartością na wyjściu  $i$ -tego neuronu, gdy na wejściu sieci podano  $\mu$ -ty obraz wejściowy.

Uczenie polega na obliczaniu gradientu funkcji błędu:

$\nabla J(\mathbf{w}) = [\partial J / \partial w_1, \partial J / \partial w_2, \dots, \partial J / \partial w_L]^T$ . Kierunek gradientu wyznacza kierunek największego wzrostu funkcji błędu w  $L$ -wymiarowej przestrzeni wag, więc w celu zmniejszenia błędu jest modyfikowany w kierunku przeciwnym:

$$\mathbf{w}' = \mathbf{w} + \Delta \mathbf{w} = \mathbf{w} - \alpha \nabla J(\mathbf{w}),$$

gdzie  $\alpha$  jest współczynnikiem szybkości uczenia.

# Uczenie sieci wielowarstwowych: algorytm propagacji wstecznej błędu 2/5



Po rozwinięciu wyjść  $y_{i\mu}$  z ostatniej warstwy:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{\mu=1}^{K_u} \sum_{i=1}^M \left[ d_{i\mu} - f_{akt}^{(2)} \left( \sum_{j=1}^P w_{ij}^{(2)} h_{j\mu} + b_i^{(2)} \right) \right]^2.$$

Po rozwinięciu wyjść  $h_{j\mu}$  z pierwszej warstwy:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{\mu=1}^{K_u} \sum_{i=1}^M \left[ d_{i\mu} - f_{akt}^{(2)} \left( \sum_{j=1}^P w_{ij}^{(2)} f_{akt}^{(1)} \left( \sum_{n=1}^N w_{jn}^{(1)} x_{n\mu} + b_j^{(1)} \right) + b_i^{(2)} \right) \right]^2.$$

# Uczenie sieci wielowarstwowych: algorytm propagacji wstecznej błędu 3/5

Pochodne cząstkowe wag z ostatniej warstwy:

$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}^{(2)}} = - \sum_{\mu=1}^{K_u} \sum_{m=1}^M (d_{m\mu} - y_{m\mu}) \frac{\partial y_{m\mu}}{\partial w_{ij}^{(2)}} = - \sum_{\mu=1}^{K_u} (d_{i\mu} - y_{i\mu}) \frac{\partial y_{i\mu}}{\partial w_{ij}^{(2)}}.$$

Suma po neuronach w warstwie wyjściowej redukuje się do  $i$ -tego neuronu, ze względu na zerowanie się pochodnych  $\frac{\partial y_{m\mu}}{\partial w_{ij}^{(2)}}$  dla  $m \neq i$ . Różniczkując

$y_{i\mu} = f_{akt}^{(2)}(\sum_{j=1}^P w_{ij}^{(2)} h_{j\mu} + b_i^{(2)}) = f_{akt}^{(2)}(a_{i\mu}^{(2)})$  względem  $w_{ij}^{(2)}$  otrzymujemy:

$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}^{(2)}} = - \sum_{\mu=1}^{K_u} (d_{i\mu} - y_{i\mu}) f_{akt}^{(2)'}(a_{i\mu}^{(2)}) h_{j\mu}$$

gdzie  $a_{i\mu}^{(2)}$  jest sygnałem pobudzenia, czyli ważoną sumą sygnałów na wyjściu neuronów z poprzedniej warstwy, zaś  $f_{akt}^{(2)'}(a_{i\mu}^{(2)})$  pochodną funkcji aktywacji względem  $a_{i\mu}^{(2)}$ .

# Uczenie sieci wielowarstwowych: algorytm propagacji wstecznej błędu 4/5



Pochodną cząstkową funkcji straty względem wagi  $w_{ij}^{(2)}$  wygodnie jest przedstawić w postaci:

$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}^{(2)}} = - \sum_{\mu=1}^{K_u} \delta_{i\mu}^{(2)} h_{j\mu},$$

gdzie  $\delta_{i\mu}^{(2)} = (d_{i\mu} - y_{i\mu}^{(2)}) f_{akt}^{(2)'}(a_{i\mu}^{(2)})$  jest tzw. sygnałem błędu  $i$ -tego neuronu w warstwie wyjściowej.

Podobnie obliczane są pochodne cząstkowe w niższych warstwach sieci, z tą różnicą, że sygnał błędu neuronu niższej warstwy jest sumą ważoną sygnałów błędów neuronów wyższej warstwy przemnożoną przez pochodną funkcji aktywacji neuronu niższej warstwy. Dla sieci dwuwarstwowej jest to:

$$\delta_{j\mu}^{(1)} = f_{akt}^{(1)'}(a_{j\mu}^{(1)}) \sum_{i=1}^M w_{ij}^{(2)} \delta_{i\mu}^{(2)}.$$

# Uczenie sieci wielowarstwowych: algorytm propagacji wstecznej błędu 5/5



W przypadku, gdy niższa warstwa jest jednocześnie pierwszą warstwą, wartość pochodnej zależy bezpośrednio od wartości na wejściu sieci:

$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}^{(1)}} = - \sum_{\mu=1}^{K_u} \delta_{i\mu}^{(1)} x_{j\mu}.$$

W ogólności, dla sieci o wielu warstwach ukrytych, sygnał błędu dla warstwy  $t$ :

$$\delta_{j\mu}^{(t)} = f_{akt}'(a_{j\mu}^{(t)}) \sum_{i=1}^{P^{(t)}} w_{ij}^{(t+1)} \delta_{i\mu}^{(t+1)},$$

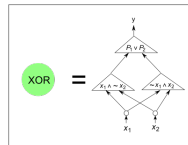
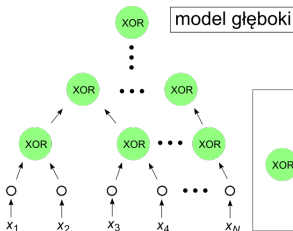
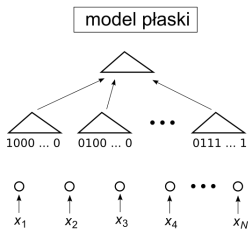
gdzie  $P^{(t)}$  jest liczbą neuronów w warstwie  $t$ , zaś wartość pochodnej:

$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}^{(t)}} = - \sum_{\mu=1}^{K_u} \delta_{i\mu}^{(t)} h_{j\mu}^{(t-1)}.$$



# Klasyczne (płaskie) sztuczne sieci neuronowe: problemy z uczeniem

- Zanikający gradient w przypadku sieci o dużej liczbie warstw.
- Słabe uogólnianie w przypadku dużej liczby warstw.
- Wymagana duża liczba neuronów w przypadku sieci dwuwarstwowych (teoretycznie wystarczających do reprezentowania dowolnych funkcji). W przypadku  $N$ -wymiarowego problemu parzystości zastosowanie sieci dwuwarstwowej (płaskiej) wymaga użycia  $2^{N-1} + 1$  neuronów, podczas gdy w sieci o architekturze głębokiej potrzeba ich tylko  $3(N - 1)$ .

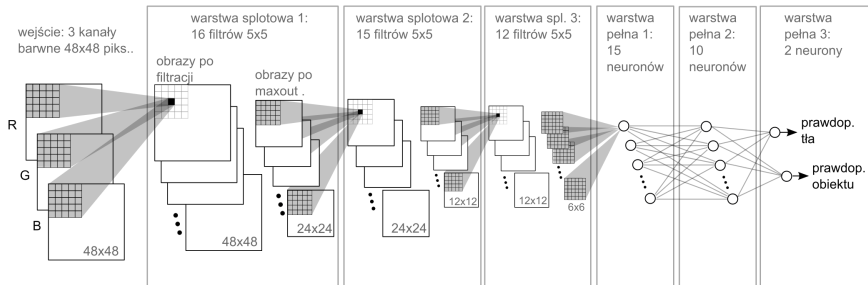


Techniki poprawiające szybkość uczenia i uogólnianie:

- funkcja aktywacji ReLU:  $f(x) = \max(x, 0)$ ,
- regularyzacja metodą *dropout*: aktywacja losowo wybranych neuronów,
- funkcja entropii skrośnej (*cross entropy*)  
 $J = -\frac{1}{L} \sum_{i=1}^M (d_i \log y_i + (1 - d_i) \log(1 - y_i))$  jako funkcji błędu zamiast średniego błędu kwadratowego,
- warstwa wyjściowa *softmax* pozwala na uzyskanie ciągu prawdopodobieństw przypisania wektora obserwacji do poszczególnych klas,
- użycie specjalnej struktury sieci do różnych zastosowań: sieci z warstwami splotowymi (*convolutional neural networks*) do rozpoznawania obrazów, GRNN lub LSTM do rozpoznawania sekwencji (np. treści wypowiedzi),
- redukcja rozdzielczości obrazu w warstwie splotowej (*subsampling*).

- Warstwę splotową można traktować jako zbiór filtrów macierzowych (zwykle  $3 \times 3$  lub  $5 \times 5$ ) np. do wyostrzania, uśredniania obrazu lub uwypuklania krawędzi i innych cech obrazu. Wartość każdego piksela obrazu po filtracji jest sumą ważoną pikseli otoczenia tego piksela w obrazie oryginalnym. W przypadku wielu obrazów na wejściu danej warstwy (np. kanałów barwnych dla pierwszej warstwy), sumowanie odbywa się również po obrazach wejściowych.
- Obraz po filtracji często jest zmniejszany poprzez redukcję rozdzielczości, dzięki czemu, poza zmniejszeniem liczby wag, uzyskiwana jest większa odporność (inwariantność) względem niewielkich przesunięć obrazu. Najczęściej stosowaną operacją redukcji rozdzielczości jest operacja *maxout* polegająca na zamianie dwuwymiarowej tablicy pikseli na wartość piksela o największej jasności.
- Parametry filtrów (wartości wag) są współdzielone w obrębie obrazu i podlegają uczeniu.
- Sieci z warstwami splotowymi zawierają również warstwy o pełnej liczbie połączeń pomiędzy neuronami.

Przykładowa sieć z warstwami splotowymi (*convolutional layers*) do detekcji obiektów:



# Sieci z warstwami splotowymi - zastosowania w sieciach samouczących się

W warstwach splotowych w trakcie uczenia powstaje specyficzna hierarchia przekształceń - kolejne warstwy splotowe reprezentują coraz bardziej złożone cechy obrazów - od prostych kształtów typu krawędzie i okręgi po złożone obiekty.

Dzięki hierarchii przekształceń i wykorzystaniu informacji przestrzennej - rozmieszczenia obiektów w dwu i trójwymiarowej przestrzeni, sieci z warstwami splotowymi znajdują zastosowania:

- w grach planszowych, takich jak szachy czy Go,
- w grach strategicznych czy FPS, takich jak Starcraft czy Quake,
- w systemach sterowania w połączeniu z widzeniem komputerowym - optymalizacja strategii poruszania się autonomicznych pojazdów czy robotów na podstawie obrazu wideo.

Możliwe jest stosowanie techniki *transfer learning* polegającej na wstępnym uczeniu sieci z nauczycielem na zadaniach wykorzystujących podobne obrazy co w zadaniu docelowym, a następnie wykorzystaniu warstw splotowych w zadaniu docelowym jako ekstraktora cech dla obrazów tego samego typu.

# Uczenie strategii w grach jednoosobowych na Atari 2600



Główne idee i osiągnięcia projektu:

- zastosowanie sieci z warstwami splotowymi do aproksymacji funkcji użyteczności stanów (*DQN*) na podstawie obrazów dwuwymiarowych,
- uczenie tylko na podstawie obrazów z monitora i bieżących nagród, bez jakiegokolwiek wiedzy o poszczególnych grach,
- zastosowanie tej samej uniwersalnej metody uczenia dla każdej z 49 różnych gier,
- wprowadzenie szeregu innowacji zapewniających stabilność procesu uczenia.

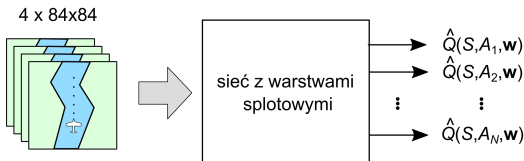
Projekt został opisany w 2 publikacjach:

- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013): Playing atari with deep reinforcement learning. ArXiv:1312.5602.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D. (2015): Human-level control through deep reinforcement learning. Nature, 518(7540):529–533.

# Uczenie strategii w grach na Atari

## - budowa systemu

- Na wejście sieci podawane są 4 kolejne klatki obrazu wideo o rozmiarach 84x84 piksele, co przybliża problem do problemu decyzyjnego Markowa (*MDP*).
- Sieć składa się z 3 warstw splotowych i dwóch warstw o pełnej liczbie połączeń z warstwami wcześniejszymi.
- Ostatnia warstwa zawiera 18 neuronów reprezentujących użyteczności możliwych akcji (stan drążka dżojstika z lub bez wciśniętego przycisku).



# Uczenie strategii w grach na Atari

## - przebieg uczenia



- Do uczenia użyto algorytmu *Q-learning* bez śladów aktywności ze strategią uczenia  $\epsilon$ -zachłanną.
- Nagrody za przejścia do poszczególnych stanów były takie same dla wszystkich gier: 1 jeśli punktacja gry wzrosła, -1 jeśli zmalała i 0 jeśli pozostała bez zmian.
- Modyfikacja wag przebiegała według wzoru:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \max_a \hat{Q}(S', a, \tilde{\mathbf{w}}) - \hat{Q}(S, A, \mathbf{w})] \nabla_{\mathbf{w}} \hat{Q}(S, A, \mathbf{w}),$$

gdzie  $\tilde{\mathbf{w}}$  to parametry wagowe sieci docelowej (*target network*), które są zamieniane na  $\mathbf{w}$  co pewien czas. Uczenie odbywa się z wykorzystaniem 32-elementowej paczki (*minibatch*) zróżnicowanych przykładów do uczenia.



# Uczenie strategii w grach na Atari

## - poprawa stabilności uczenia



- Doświadczenia w postaci czwórek  $(S, A, R, S')$  zapisywane są w tzw. pamięci powtórek (*replay memory*). Paczki (*minibatch*) do uczenia są składane z losowych przykładów z różnych epizodów, dzięki czemu nie ma pomiędzy nimi korelacji mogących destabilizować proces uczenia.
- Co pewną liczbę epok uczenia zapisywana jest kopia sieci (*duplicated target network*) i to z niej pochodzi estymata maksymalnej użyteczności akcji w kolejnym stanie. Technika zduplikowanej sieci celu pozwala uniknąć oscylacji związanych z jednoczesną zmianą wartości  $\hat{Q}(S, A, \mathbf{w})$  i  $\max_a \hat{Q}(S', a, \mathbf{w})$ .
- Wartość  $R + \gamma \max_a \hat{Q}(S', a, \tilde{\mathbf{w}}) - \hat{Q}(S, A, \mathbf{w})$  jest zawężana do przedziału  $[-1, 1]$ , co również sprzyja stabilności uczenia.

# Uczenie strategii w grach na Atari

## - wyniki



- Dla każdej gry wyuczono oddzielną sieć reprezentującą strategię z użyciem 50 milionów klatek wideo wraz z informacją o nagrodach.
- Dla 43 gier na 49 uzyskano wyniki lepsze niż w przypadku innych systemów opartych na uczeniu ze wzmocnieniem.
- Do testowania systemu zatrudniono profesjonalnych graczy, z których każdy po 2 godzinach treningu rozegrał po 20 epizodów dla każdej gry bez ścieżki dźwiękowej. Dla 22 gier system osiągnął wyniki lepsze od ludzi.
- Są gry, w przypadku których system nie był w stanie nauczyć się lepszej strategii niż losowa. Najgorsze wyniki zaobserwowano dla gry *Montezuma's Revenge*.

# Inne metody poprawy stabilności uczenia w DQN

- Miękkie kopiowanie sieci – zamiast kopiowania wag co pewien czas, ciągłe uśrednianie sieci docelowej metodą Polyaka:  $\tilde{\mathbf{w}} \leftarrow \tau \mathbf{w} + (1 - \tau) \tilde{\mathbf{w}}$
- Stosowanie priorytetów w doborze przykładów uczących do paczek (*minibatches*)
- Kształtowanie nagrody (*reward shaping*) – gdy nagrody są trudne do zdobycia lub bardzo oddalone, warto jest nagradzać za bliskość celu wprowadzając sztuczne nagrody i kary (poza celem optymalizacji)
- Zróżnicowanie stopnia trudności podczas uczenia (*curriculum learning*) – najpierw uczenie prostych zadań, później coraz trudniejszych

- W grach dwuosobowych, z punktu widzenia wybranego gracza, ma on do czynienia z niestacjonarnym środowiskiem, którego częścią jest drugi gracz. Najczęściej nie ma więc optymalnej strategii, poza trywialnymi przypadkami, a jedynie punkty równowagi lub optymalne strategie drużynowe w grach z sumą niezerową.
- Punkt równowagi Nasha to taka para strategii obu graczy, że żadnej z nich nie opłaca się zmieniać bez zmiany strategii drugiego gracza.
- Istnieją punkty równowagi w strategiach mieszanych  $\pi(a|s)$ , czyli strategiach, w których akcje wybierane są losowo z ustalonymi prawdopodobieństwami.

Macierz zawiera wypłaty gracza **Wiersz** - którego akcje odpowiadają wierszom macierzy oraz wypłaty gracza **Kolumna** - którego akcje odpowiadają kolumnom macierzy.

## Dylemat więźnia

tablica wypłat + przejścia:

|        |           | Kolumna   |        |
|--------|-----------|-----------|--------|
|        |           | lojalność | zdrada |
| Wiersz | lojalność | 3\3       | 0\5    |
|        | zdrada    | 5\0       | 1\1    |

## Gra w orła i reszkę

tablica wypłat + przejścia:

|        |        | Kolumna |        |
|--------|--------|---------|--------|
|        |        | orzeł   | reszka |
| Wiersz | orzeł  | 1\1     | -1\1   |
|        | reszka | -1\1    | 1\1    |

# Strategie mieszane w problemach z niepełną informacją o stanie



Gdy agent widzi tylko sąsiednie pola w pionie i poziomie, to pola zaznaczone na szaro są traktowane jako ten sam stan. W przypadku strategii czystej - deterministycznej (zawsze wybierana jest jedna akcja), z niektórych pól w pierwszym wierszu agent nigdy nie osiągnie dodatniej nagrody:

|    |   |   |   |    |
|----|---|---|---|----|
| →  | ← | ↓ | ← | ←  |
| -1 |   | 1 |   | -1 |

W przypadku strategii mieszanej - niedeterministycznej np.

$\pi(\text{w lewo}|\text{mur od góry i od dołu}) = 0,5$ ,  $\pi(\text{w prawo}|\text{mur od góry i od dołu}) = 0,5$ :

|    |   |   |   |    |
|----|---|---|---|----|
| →  | ↔ | ↓ | ↔ | ←  |
| -1 |   | 1 |   | -1 |

agent prawie zawsze osiąga dodatnią nagrodę.

Istnieją zatem problemy, w których optymalna może być strategia mieszana - niedeterministyczna, czego nie można osiągnąć w oparciu o samą funkcję wartości stanów lub par (stan,akcja).

Aproksymacja funkcji strategii pozwala na bezpośrednie wyznaczenie akcji w danym stanie bez konieczności wyznaczania jej wartości. Na wyjściu aproksymatora mogą być podawane prawdopodobieństwa wyboru akcji lub wartości preferencji poszczególnych akcji w danym stanie.

Można wyróżnić 2 podejścia w uczeniu z aproksymacją funkcji strategii:

- uczenie samej strategii (*Policy gradient*),
- uczenie strategii wyboru akcji i funkcji wartości (*Actor-Critic*).

# Uczenie samej strategii

## - *Policy gradient*



Metoda *Policy gradient* polega na uczeniu aproksymowanej funkcji strategii, która wyznacza prawdopodobieństwa lub preferencje poszczególnych akcji w poszczególnych stanach.

Prawdopodobieństwa  $\pi(a|s, \theta)$ , gdzie  $\theta$  jest wektorem parametrów aproksymatora, mogą być reprezentowane siecią neuronową z funkcją softmax jako funkcją aktywacji ostatniej warstwy.

W przypadku użycia aproksymatora liniowego, wartości wyjściowe - preferencje nie muszą się sumować do jedności i aby uzyskać prawdopodobieństwa konieczne jest użycie zewnętrznej funkcji softmax:

$$\pi(a|s, \theta) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}},$$

gdzie  $h(s, a, \theta)$  jest aproksymacją preferencji wykonania akcji  $a$  w stanie  $s$  i może być aproksymacją liniową:

$$h(s, a, \theta) = \theta^T \Phi(s, a),$$

gdzie  $\Phi(s, a)$  jest wektorem parametrów stanu w postaci zakodowanej.



- Reprezentowanie strategii mieszanych - niedeterministycznych, które mogą być optymalnymi strategiami docelowymi.
- Możliwość łatwego dołączania wiedzy o problemie (np. co system powinien zrobić w danej sytuacji).
- Szybsza zbieżność w niektórych problemach - strategia jest zwykle łatwiejsza do aproksymacji niż wartości akcji (wartości preferencji nie muszą być dokładne, wystarczy że będą w odpowiednich relacjach większy/mniejszy).

Uczenie polega na modyfikacji wektora wag  $\theta$  w kierunku maksymalnego wzrostu funkcji wartości strategii  $J(\theta)$ :

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta),$$

gdzie funkcja wartości strategii (*performance*) może być wartością stanu początkowego  $s_0$  przy strategii  $\pi_\theta$ :

$$J(\theta) = v_{\pi_\theta}(s_0).$$

Po podstawieniu  $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$ , zróżniczkowaniu, przekształceniu i uproszczeniu:

$$\begin{aligned} \nabla J(\theta) &\cong \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s, \theta) \\ &= \mathbb{E} \left[ \sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \theta) \right], \end{aligned}$$

gdzie  $\mu(s)$  jest prawdopodobieństwem wystąpienia stanu  $s$ :

$$\mu(s) \geq 0, \quad \sum_s \mu(s) = 1.$$

# Policy gradient - zastąpienie funkcji wartości próbkami



Prawdziwą wartość  $q_\pi(S_t, a)$  można zastąpić wartością aproksymowaną  $\hat{Q}(S_t, a, \mathbf{w})$ , co wymaga dodatkowego uczenia aproksymatora funkcji użyteczności akcji, tak jak w metodach *Actor-Critic*. Innym wyjściem jest zastąpienie  $q_\pi(S_t, a)$  próbkami zwrotu  $G_t$ :

$$\begin{aligned}\nabla J(\theta) &\cong \mathbb{E} \left[ \sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \theta) \right] \\ &= \mathbb{E} \left[ \sum_a \pi(a|S_t, \theta) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)} \right] \\ &= \mathbb{E} \left[ q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] && \text{(zastępujemy } a \text{ próbką } A_t \sim \pi) \\ &= \mathbb{E} \left[ G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] && (\mathbb{E}[G_t|S_t, A_t] = q_\pi(S_t, A_t)) \\ &= \mathbb{E} [G_t \nabla \ln \pi(A_t|S_t, \theta)] && (\nabla \ln x = \frac{\nabla x}{x}).\end{aligned}$$

Jak wynika z przedostatniego członu, wielkość gradientu zależy proporcjonalnie od zwrotu  $G_t$  za wykonanie akcji  $A_t$  i odwrotnie proporcjonalnie od prawdopodobieństwa wyboru akcji  $A_t$ , dzięki czemu nie są faworyzowane akcje, które są często wykonywane.

# *Policy gradient* - algorytm **REINFORCE** (Williams, 1992)



$\alpha$  – współczynnik szybkości uczenia

$\theta$  – wektor wag aproksymatora funkcji strategii  $\pi(a|s, \theta)$

**for all** dla każdego epizodu **do**

przejdź epizod zgodnie ze strategią  $\pi(a|s, \theta)$  zapisując wszystkie stany,  
akcje i nagrody:  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

**for all** dla każdego kroku epizodu  $t = 0, 1, \dots, T - 1$  **do**

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

$$\theta \leftarrow \theta + \alpha \gamma^t G \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)}$$

**end for**

**end for**

# *Policy gradient* - cechy algorytmu **REINFORCE**



Algorytm działa podobnie jak metoda Monte Carlo z iteracją strategii - posiada więc podobne cechy:

- wymagana jest epizodyczność procesu decyzyjnego,
- konieczne jest przejście całego epizodu, zanim zmodyfikowane zostaną wagi aproksymatora funkcji strategii,
- powolna zbieżność związana z dużą wariancją zwrotu i brakiem wykorzystania wartości akcji w kolejnym stanie (*bootstrapping*).

# Policy gradient z wartością odniesienia (*baseline*)



We wzorze na funkcję wartości strategii można umieścić dowolną wartość, np.  $b(s)$  niezależną od akcji:

$$\nabla J(\theta) \cong \mathbb{E} \left[ \sum_a (q_\pi(S_t, a) - b(s)) \nabla \pi(a|S_t, \theta) \right]$$

Jest to możliwe, gdyż wartość niezależna od  $a$  nie wpływa na wartość gradientu:

$$\sum_a b(s) \nabla \pi(a|S_t, \theta) = b(s) \nabla \sum_a \pi(a|S_t, \theta) = b(s) \nabla 1 = 0.$$

Jeśli dodatkowo wartością odniesienia będzie wartość stanu  $b(s) = \widehat{V}(s, \mathbf{w})$ , wzór na modyfikację wektora wag strategii przyjmie postać:

$$\theta \leftarrow \theta + \alpha (G_t - \widehat{V}(S_t, \mathbf{w})) \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)}.$$

Dodanie wartości odniesienia znacznie stabilizuje proces uczenia poprzez zmniejszenie wariancji związanej ze zwrotem, jednak wymaga dodatkowego aproksymatora wartości stanów, który może być uczony również metodą Monte Carlo.

# Policy gradient - algorytm REINFORCE z wartością odniesienia

$\mathbf{w}$  – wektor wag aproksymatora funkcji wartości stanów  $\hat{V}(s, \mathbf{w})$

$\theta$  – wektor wag aproksymatora funkcji strategii  $\pi(a|s, \theta)$

$\alpha_{\mathbf{w}}$  – współczynnik szybkości uczenia aproksymatora  $\hat{V}(s, \mathbf{w})$

$\alpha_{\theta}$  – współczynnik szybkości uczenia aproksymatora  $\pi(a|s, \theta)$

**for all** dla każdego epizodu **do**

przejdź epizod zgodnie ze strategią  $\pi(a|s, \theta)$  zapisując wszystkie stany,  
akcje i nagrody:  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

**for all** dla każdego kroku epizodu  $t = 0, 1, \dots, T - 1$  **do**

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

$$\delta \leftarrow G - \hat{V}(S_t, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}} \delta \nabla \hat{V}(S_t, \mathbf{w})$$

$$\theta \leftarrow \theta + \alpha_{\theta} \gamma^t \delta \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)}$$

**end for**

**end for**

- W odróżnieniu od metody *REINFORCE*, metoda *Actor-Critic* pozwala na uczenie w trybie *on-line* w trakcie epizodów.
- Estymacja wartości zwrotu odbywa się z wykorzystaniem próbki bieżącej nagrody i wartości kolejnego stanu (*bootstrapping*), co przyspiesza uczenie i upodobnia algorytm do metody SARSA bez dodatkowej eksploracji.
- Podobnie jak w przypadku metody *REINFORCE* możliwe jest wyuczenie strategii mieszanej.



$\mathbf{w}$  – wektor wag aproksymatora funkcji wartości stanów  $\hat{V}(s, \mathbf{w})$

$\theta$  – wektor wag aproksymatora funkcji strategii  $\pi(a|s, \theta)$

$\alpha_{\mathbf{w}}$  – współczynnik szybkości uczenia aproksymatora  $\hat{V}(s, \mathbf{w})$

$\alpha_{\theta}$  – współczynnik szybkości uczenia aproksymatora  $\pi(a|s, \theta)$

**for all** dla każdego epizodu **do**

wybierz stan początkowy  $S$

$I \leftarrow 1$

**repeat**

wybierz i wykonaj akcję  $A$  zgodnie z  $\pi(\cdot|S, \theta)$ , obserwuj  $R, S'$

$\delta \leftarrow R + \gamma \hat{V}(S', \mathbf{w}) - \hat{V}(S, \mathbf{w})$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}} \delta \nabla \hat{V}(S, \mathbf{w})$

$\theta \leftarrow \theta + \alpha_{\theta} I \delta \frac{\nabla \pi(A|S, \theta)}{\pi(A|S, \theta)}$

$S \leftarrow S'$

$I \leftarrow \gamma I$

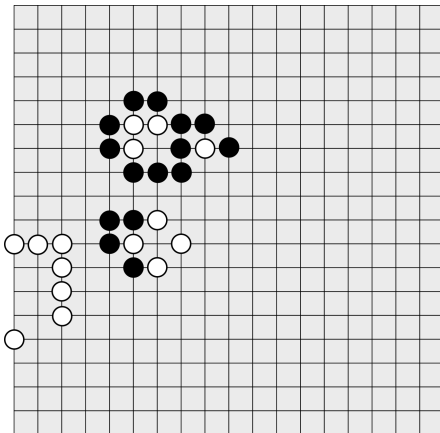
**until** stan  $S$  jest końcowy

**end for**

# Algorytm *Actor-Critic* ze śladami aktywności

$\mathbf{w}, \alpha_{\mathbf{w}}, \lambda_{\mathbf{w}}$  – wektor wag, wsp. szybkości uczenia, wsp. świeżości  
 dla aproksymatora funkcji wartości stanów  $\hat{V}(s, \mathbf{w})$   
 $\theta, \alpha_{\theta}, \lambda_{\theta}$  – wektor wag, wsp. szybkości uczenia, wsp. świeżości  
 dla aproksymatora funkcji strategii  $\pi(a|s, \theta)$   
**for all** dla każdego epizodu **do**  
 wybierz stan początkowy  $S$   
 $I \leftarrow 1$   
 $\mathbf{z}_{\mathbf{w}}$  - suma ważona gradientów dla  $\hat{V}(s, \mathbf{w})$  zainicjowana zerami  
 $\mathbf{z}_{\theta}$  - suma ważona gradientów dla  $\pi(a|s, \theta)$  zainicjowana zerami  
**repeat**  
 wybierz i wykonaj akcję  $A$  zgodnie z  $\pi(\cdot|S, \theta)$ , obserwuj  $R, S'$   
 $\delta \leftarrow R + \gamma \hat{V}(S', \mathbf{w}) - \hat{V}(S, \mathbf{w})$   
 $\mathbf{z}_{\mathbf{w}} \leftarrow \gamma \lambda_{\mathbf{w}} \mathbf{z}_{\mathbf{w}} + \nabla \hat{V}(S, \mathbf{w})$   
 $\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}} \delta \mathbf{z}_{\mathbf{w}}$   
 $\mathbf{z}_{\theta} \leftarrow \gamma \lambda_{\theta} \mathbf{z}_{\theta} + I \frac{\nabla \pi(A|S, \theta)}{\pi(A|S, \theta)}$   
 $\theta \leftarrow \theta + \alpha_{\theta} \delta \mathbf{z}_{\theta}$   
 $S \leftarrow S'$   
 $I \leftarrow \gamma I$   
**until** stan  $S$  jest końcowy  
**end for**

- Ruchy wykonywane są naprzemiennie zaczynając od gracza grającego czarnymi. Ruch polega na postawianiu kamienia na przecięciu linii lub rezygnacji z ruchu (*pass*).
- Jeśli obaj gracze spasują jeden po drugim to gra jest zakończona.
- Otoczenie kamieni przeciwnika własnymi kamieniami pozbawiając go sąsiedztwa z pustymi polami (tzw. oddechów) powoduje zabicie wszystkich otoczonych kamieni przeciwnika.
- Nie wolno wykonać ruchu pozbawiającego grupę własnych kamieni ostatniego sąsiedniego wolnego pola.
- Jeśli jeden z graczy zbije drugiemu kamień, to drugi nie może go odbić wracając do sytuacji sprzed zbicia (zapobieżenie nieskończonym cyklom).
- Grę wygrywa gracz, który uzyskał maksymalną sumę otoczonych pól, zбитych kamieni oraz kamieni, które zostałyby przez niego zbite, gdyby gra trwała dłużej.



- Bardzo trudna heurystyczna ocena wartości stanu gry.
- Bardzo duża liczba dozwolonych ruchów w jednym stanie - średnio 250 (w szachach 35), duża liczba ruchów w grze (kroków epizodu) - średnio 150 (w szachach 80) i duża liczba możliwych stanów rzędu  $3^{19 \times 19}$ .
- Trudności w zastosowaniu prostych heurystyk w algorytmach mini-max czy alfa-beta.
- Trudność z wykorzystaniem wiedzy ekspertów.
- Trudność w zastosowaniu standardowego uczenia ze wzmocnieniem z aproksymowaną funkcją wartości stanów lub funkcją strategii (*policy gradient*).

Opracowano 3 główne wersje systemu gry w Go z wykorzystaniem sieci samouczących się:

- **AlphaGo** - wykorzystanie wiedzy ekspertów, ręcznie dobieranych cech stanu gry oraz uczenia na grach pomiędzy własnymi strategiami,
- **AlphaGo Zero** - uczenie tylko na grach pomiędzy własnymi strategiami (bez wykorzystania gier pomiędzy ludźmi i bez ręcznie dobieranych cech stanów),
- **AlphaZero** - uczenie strategii dowolnych gier planszowych

# Uczenie strategii gry w Go - literatura



## AlphaGo:

Silver, D. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016).

## AlphaGo Zero:

Silver, D. et al. Mastering the game of Go without human knowledge. Nature 550, 354–359 (2017).

## AlphaZero:

Silver, D. et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362, 1140–1144 (2018).

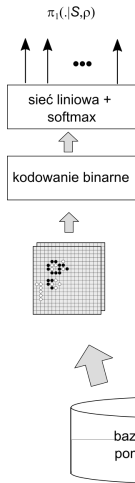
- Wykorzystanie sieci z warstwami splotowymi - na wejściu 48 macierzy (kanałów) o rozmiarach 19x19.
- Wykorzystanie wiedzy heurystycznej: poza macierzami zawierającymi informacje o rozmieszczeniu kamieni, macierz cech heurystycznych związanych z wiedzą ekspercką.
- Dwa etapy uczenia:
  - z nauczycielem (*supervised learning* (*SL*)) na podstawie ruchów profesjonalnych graczy,
  - uczenie ze wzmocnieniem (*RL*) podczas partii rozgrywanych pomiędzy uczonymi strategiami (*self-play*), z użyciem strategii wyuczonej w pierwszym etapie (uczenia z nauczycielem).
- Użycie techniki **MCTS** do obliczania wartości akcji w trakcie eksploatacji.
- Użycie specjalnej sieci o małej złożoności obliczeniowej z kodowaniem binarnym realizującej strategię **Rollout**.



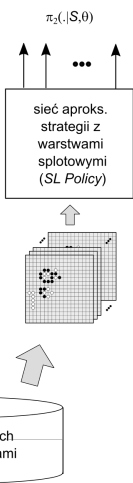
# AlphaGo - schemat uczenia



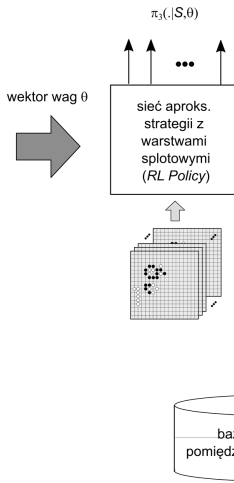
*SL Rollout Policy*



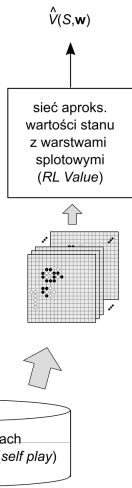
*SL Policy*



*RL Policy*



*RL Value*



- ***SL Rollout Policy*** - sieć liniowa z kodowaniem binarnym reprezentująca strategię *Rollout* wykorzystywaną w fazie eksploatacji. Sieć na wyjściu prezentuje  $19 \times 19 + 1$  wartości prawdopodobieństw wyboru akcji dla podanego na wejściu stanu obliczanych z wykorzystaniem funkcji softmax. Wiedza heurystyczna jest zawarta w sposobie kodowania stanów. Mała złożoność obliczeniowa pozwala na wyznaczanie wartości akcji w węzłach końcowych drzewa *MCTS* podczas wykonywania ruchu. Sieć jest uczona z nauczycielem wyboru ruchów zgodnych z wyborami ekspertów poprzez minimalizację błędu różnicy pomiędzy odpowiedzią własną, a tym co zaproponował ekspert.
- ***SL Policy*** - sieć o 13 warstwach splotowych i jednej warstwie pełnej z funkcją aktywacji softmax jest uczona podobnie jak sieć liniowa *SL Rollout Policy* ale z innym algorytmem uczenia oraz z wiedzą heurystyczną zawartą w macierzach wejściowych (48 macierzy  $19 \times 19$ ) np. liczba przylegających pustych pól, liczba kamieni, jaką można zbić wykonując ruch w tym polu.

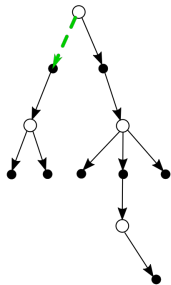
- **RL Policy** - sieć o tej samej strukturze i początkowych wartościach wag co *SL Policy* jest uczona metodą *policy gradient* podczas gier pomiędzy aktualną strategią a strategiami przeciwnika wylosowanymi z poprzednich iteracji (co zapobiega nadmiernemu dopasowaniu i zmniejszeniu uogólniania). Osiąga znaczną poprawę dokładności w stosunku do sieci *SL Policy*.
- **RL Value** - sieć o tej samej strukturze co sieci *SL Policy* i *RL Policy* z wyjątkiem pierwszej i ostatniej warstwy. Pierwsza warstwa ma na wejściu 49 macierzy. Dodatkowa macierz zawiera informację o tym kto wykonuje dany ruch. Ostatnia warstwa zawiera tylko jedno wyjście (zamiast  $19 \times 19 + 1$ ) odpowiadające wartości użyteczności stanu zaprezentowanego na jej wejściu. Sieć jest uczona poprzez minimalizację błędu średniokwadratowego pomiędzy odpowiedzią własną a zwrotem uzyskanym podczas realizacji strategii aproksymowanej siecią *RL Policy* (przypomina algorytm Monte Carlo oceny wartości stanów dla ustalonej strategii).

- Symulacje *Rollout* to inaczej symulacje Monte Carlo (MC) pozwalające na wyznaczenie wartości akcji przy ustalonej strategii reprezentowanej przez aproksymator funkcji wartości lub funkcji strategii.
- W odróżnieniu do typowych zastosowań MC, symulacje *Rollout* służą do obliczania wartości akcji w tylko jednym stanie przed podjęciem decyzji w trakcie eksploatacji systemu (np. w trakcie gry). Po wybraniu akcji, wyniki symulacji są usuwane.
- Symulacje *Rollout* wymagają modelu środowiska o małej złożoności obliczeniowej.
- Użycie symulacji *Rollout* jest opłacalne w przypadku dobrze ogólniającego ale mało dokładnego aproksymatora funkcji wartości lub funkcji strategii i modelu środowiska o małej złożoności (np. gry planszowe, takie jak szachy, Go).

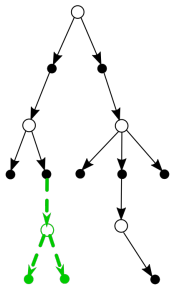
Symulacje *MCTS* pełnią podobną funkcję co *Rollout*, ale z możliwością wartościowania akcji w większej liczbie stanów, niż w bieżącym węźle decyzyjnym oraz z wyborem akcji w oparciu o własną strategię (*MCTS policy*), która jest najczęściej złożeniem wyników symulacji i strategii wcześniej wyuczonej. Korzeniem drzewa *MCTS* jest bieżący węzeł decyzyjny. Cztery podstawowe operacje *MCTS*:

- selekcja akcji - odbywa się z wykorzystaniem własnej strategii na podstawie średnich wartości zwrotu przypisanych do akcji lub łącznie ze strategią wyuczoną poza *MCTS*. Stosowana jest eksploracja akcji,
- ekspansja - co pewien czas krawędzie odpowiadające obiecującym akcjom są dołączane do drzewa,
- symulacja - przeprowadzana jest w obrębie drzewa *MCTS* od jego korzenia do jednego z węzłów końcowych, a następnie z tego węzła końcowego przeprowadzana jest symulacja *Rollout* aż do zakończenia epizodu (np. do zakończenia gry).
- uaktualnienie wartości akcji w krawędziach drzewa *MCTS* - odbywa się na podstawie zwrotu z kolejnych węzłów *MCTS* oraz z symulacji *Rollout*.

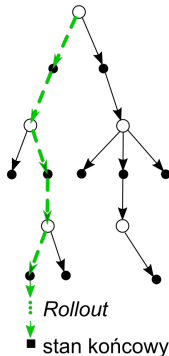
1. Wybór akcji



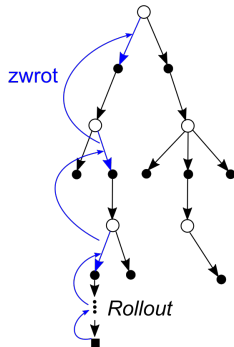
2. Ekspansja



3. Symulacja



4. Modyfikacja wartości użyteczności akcji



- stan decyzyjny uczonej strategii
- stan decyzyjny przeciwnika

Przed wykonaniem każdego ruchu w bieżącym stanie budowane jest drzewo MCTS.

W każdej krawędzi drzewa przechowywana jest wartość związanej z nią akcji -  $Q(s, a)$ , liczba odwiedzin -  $N(s, a)$  oraz wstępne prawdopodobieństwo wyboru akcji -  $P(s, a)$  uzyskane z sieci *SL Policy*. Podczas budowy drzewa MCTS przeprowadzane są symulacje - epizody od bieżącego stanu (korzeń drzewa) do końca gry. Akcja w kroku  $t$ :

$$A_t = \underset{a}{\operatorname{argmax}}(Q(S_t, a) + u(S_t, a)),$$

gdzie

$$u(s, a) \sim \frac{P(s, a)}{1 + N(s, a)}$$

jest wartością proporcjonalną do wstępnego prawdopodobieństwa wyboru akcji ale obniżoną proporcjonalnie do liczby wykonań, by wymusić eksplorację akcji rzadziej wykonywanych.

W węźle końcowym drzewa MCTS uruchamiana jest symulacja *Rollout* ze strategią reprezentowaną przez sieć *SL Rollout Policy* od tego węzła do końca gry. Wartość użyteczności stanu  $s_L$  w węźle końcowym obliczana jest ze wzoru:

$$V(s_L) = (1 - \eta)\hat{V}(s_L, \mathbf{w}) + \eta G_L,$$

gdzie  $\eta$  jest współczynnikiem wykorzystania zwrotu *Rollout*,  $\hat{V}(s_L, \mathbf{w})$  jest wartością użyteczności reprezentowaną przez sieć *RL Value*, natomiast  $G_L$  jest zwrotem z symulacji *Rollout*.

Po zakończeniu każdej  $n$ -tej symulacji obliczana są na nowo: licznik odwiedzin i użyteczność akcji ze wzorów:

$$\begin{aligned}N(s, a) &= \sum_{i=1}^n 1(s, a, i) \\Q(s, a) &= \frac{1}{N(s, a)} \sum_{i=1}^n 1(s, a, i) V(s_L^i),\end{aligned}$$

gdzie  $1(s, a, i) = 1$  gdy w symulacji  $i$ -tej wykonano akcję  $a$  w stanie  $s$ ,  $s_L^i$  jest stanem końcowym MCTS w  $i$ -tej symulacji.

Po zakończeniu symulacji MCTS, w stanie bieżącym wykonywana jest akcja najczęściej wybierana w symulacjach MCTS:

$$a_{S_t} = \operatorname{argmax}_a N(S_t, a)$$



- Do ustalenia wstępnego prawdopodobieństwa wyboru akcji w **MCTS** użyto strategii sieci *SL Policy*, pomimo że sieć *RL Policy* przewiduje akcje z dużo większą dokładnością. Jest to związane z tym, że sieć *RL Policy* była uczona strategii wzajemnie optymalnej w punkcie równowagi, podczas gdy sieć *SL Policy* uczona była gry z człowiekiem, co pozwala na lepszą decyzję w grze z człowiekiem.
- Po wielu próbach ustalono optymalną wartość współczynnika  $\eta \cong 0.5$ , co wskazuje na to, że tak samo ważny jest zwrot *Rollout* związany z uczeniem na danych eksperckich, co wartość stanu uzyskana metodą uczenia *self play*.

- Uczenie tylko na podstawie symulowanych gier pomiędzy strategiami (self-play) bez wykorzystania wiedzy ekspertów.
- Brak wiedzy ekspertów w strukturze modelu.
- Jeden model (sieć neuronowa) reprezentujący funkcję strategii i funkcję wartości, dzięki dwóm warstwom wyjściowym.
- Wykorzystanie symulacji [MCTS](#) w procesie uczenia.
- Dużo lepsze wyniki w porównaniu do AlphaGo.

- Na wejście sieci podawanych jest 17 macierzy binarnych o rozmiarach 19x19 (rozmiary planszy do gry w Go), po 8 macierzy zawiera układy kamieni graczy w 8 ostatnich stanach, dodatkowa macierz zawiera informację o graczu, który wykonuje ruch w danym stanie.
- Sieć zawiera 40 warstw splotowych typu *residual* (z dodatkowymi połączeniami do wyższych warstw niż kolejna).
- Ostatnia warstwa składa się z dwóch części:
  - neuron reprezentujący wartość stanu  $\hat{V}(s)$ ,
  - warstwa z funkcją aktywacji *softmax* reprezentująca funkcję strategii – wektor rozkładu prawdopodobieństw akcji  $\mathbf{p}(a|s)$ .

- Uczenie odbywa się wyłącznie poprzez symulacje gier aktualnych strategii, które są na bieżąco modyfikowane zaczynając od strategii losowych.
- Do modyfikacji wag sieci (modelu funkcji wartości i funkcji strategii) wykorzystywane są wyniki symulacji **MCTS** (w AlphaGo używano ich tylko w gotowym - działającym systemie) w uproszczonej wersji (bez symulacji *Rollout*).
- Funkcja straty:

$$L = (G - \hat{V}(s))^2 - \pi(a|s)^T \log \mathbf{p}(a|s) + c\|\theta\|^2,$$

gdzie:

$[\mathbf{p}(a|s), \hat{V}(s)] = f_{\theta}(s)$  – wartości na wyjściu sieci:  $\mathbf{p}(a|s)$  – wektor prawdopodobieństw akcji w stanie  $s$ ,  $\hat{V}(s)$  – użyteczność stanu  $s$ ,  
 $G$  – zwrot z bieżącej symulacji (1 za wygraną, -1 za przegraną),  
 $\pi(a|s)$  – wektor prawdopodobieństw akcji w stanie  $s$  uzyskany na podstawie 1600 symulacji MCTS,  
 $c\|\theta\|^2$  – człon regularyzacyjny L2 redukujący nadmierne dopasowanie do danych uczących.

- Uczenie odbywa się poprzez uaktualnianie wag sieci dla paczek danych (*minibatch*) zawierających po 2048 przykładów -  $(\mathbf{p}(a|s), \pi(a|s), \hat{V}(s), G)$ .

AlphaZero jest ulepszoną i uogólnioną wersją algorytmu AlphaGo Zero na Go, szachy i shogi.

Różnice pomiędzy AlphaZero a AlphaGo Zero:

- w AlphaZero nie są uwzględniane symetrie - z uwagi na szachy i shogi,
- w AlphaGo Zero uczenie podzielone jest na iteracje, w których wykorzystywane są najlepsze dotychczasowe strategie obu graczy, podczas gdy w AlphaZero uczenie jest ciągłe - bez podziału na iteracje,

# Ciekawe zastosowania sieci samouczących się



Rozszerzenia algorytmu AlphaZero:

- Uczenie strategii gry w Starcraft ([AlphaStar](#))
- Optymalizacja mnożenia macierzy ([AlphaTensor](#)) - minimalizacja liczby mnożeń.
- Optymalizacja kodu programu rozwiązującego wybrany problem - AlphaDev
- Sterowanie polem magnetycznym w kontrolowaniu plazmy w reaktorze termojądrowym.
- Przewidywanie struktury białek - AlphaFold.
- Uczenie strategii w grze Stratego - DeepNash.

Inne ciekawe zastosowania:

- Uczenie strategii odpowiedzi na zapytania w systemach LLM typu GPT-x *Reinforcement Learning from Human Feedback*(RLHF) - opracowane przez OpenAI.

# Systemy wieloagentowe jako gry wieloosobowe



- Agent – obiekt autonomiczny i racjonalny - stara się zmaksymalizować zysk w interakcji ze środowiskiem, w tym w interakcji z innymi agentami.
- Agenty, z zależności od sytuacji, mogą ze sobą rywalizować lub współpracować (tworzyć koalicje), nawet w grach o sumie zerowej.
- Możliwa jest komunikacja pomiędzy agentami.
- System może być wieloagentowy nawet pomimo założenia o pełnej współpracy pomiędzy agentami (nie jest grą), dzięki autonomii agentów wynikającej np. z ograniczonej komunikacji, zbyt dużej złożoności gdyby stosować centralne sterowanie czy innych przyczyn technicznych.
- „Uczenie wieloagentowe” ma wiele znaczeń (np. uczenie w środowisku wielu agentów lub uczenie z wykorzystaniem wielu agentów jako przeciwników).

Algorytm AlphaStar służy do szukania jak najlepszej strategii w grze StarCraft II.  
Literatura:

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019. ISSN 1476-4687. doi:10.1038/s41586-019-1724-z. URL <https://www.nature.com/articles/s41586-019-1724-z>.



- Gra strategiczna czasu rzeczywistego (*Real Time Strategy* - *RTS*) - ruchy wykonywane są asynchronicznie w dowolnym tempie.
- Gracz (agent) dysponuje dużą liczbą jednostek, które mogą wykonywać różne polecenia. Akcje są bardziej ogólne - każdą akcję można przełożyć na sekwencję działań poszczególnych jednostek + ustawienie widoku.
- Istotnym elementem gry jest budowa budynków i rozwój technologii w odpowiedniej kolejności i z zachowaniem płynności gotówkowo-materiałowej (warstwa ekonomiczna gry).
- Stan jest tylko częściowo obserwowalny (niespełniona własność Markowa), gdyż gracz widzi świat tylko w zasięgu wzroku własnych jednostek (mgła wojny), nie zna też postępów w rozwoju technologii innych graczy.
- Gra może być wieloosobowa (wieloagentowa) ale w profesjonalnych rozgrywkach ograniczana jest zwykle do gry „jeden na jeden”.

- Dwa etapy uczenia: uczenie nadzorowane (*supervised learning*) na podstawie ruchów profesjonalnych graczy oraz uczenie ze wzmocnieniem ze strategią mieszaną przeciwnika - (*Fictitious Self-Play - FSP*).
- W przypadku uczenia nadzorowanego uczona jest tylko funkcja strategii na podstawie miary Kulbacka-Leibnera (KL). W przypadku uczenia ze wzmocnieniem stosowana jest metoda *actor-critic* z dodatkową warstwą wyjściową reprezentującą wartość stanu.
- Zastosowanie sieci rekurencyjnej LSTM (*Long short-term memory*) do analizy wektorów obserwacji z poszczególnych kroków. Wektor obserwacji złożony jest z danych przestrzennych i ogólnych informacji, co wymaga stosowania połączeń rozproszonych *scatter connections*.
- Zastosowanie sieci typu *Pointer Network* do generowania sekwencji czynności na podstawie obserwacji. Jest to sieć z mechanizmem uwagi ale wektory wejściowe i wyjściowe mogą mieć dowolną długość.
- Zastosowanie ligi strategii przeciwnika z algorytmem wyboru strategii oponenta oraz techniką *Prioritized FPS* do tworzenia mieszanin strategii oponentów.
- Algorytmy *V-trace* oraz UPGO - *Ongoing Policy Update* pozwalające na uczenie strategii w złożonej hierarchicznej przestrzeni możliwych akcji.
- Stosowanie techniki *curriculum learning* poprzez odpowiedni dobór oponentów i stosowanie sztucznych nagród (np. za odpowiedni rozwój ekonomiczny).

# Algorytm AlphaTensor - optymalizacja mnożenia macierzy



Algorytm AlphaTensor służy do szukania algorytmów (sekwencji działań) mnożenia macierzy o jak najmniejszej liczbie mnożeń par wartości i został zainspirowany algorytmem Strassen'a mnożenia macierzy  $2 \times 2$ . Literatura:

Fawzi, A., Balog, M., Huang, A. et al. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature 610, 47–53 (2022).  
<https://doi.org/10.1038/s41586-022-05172-4>

- Celem jest dekompozycja (faktoryzacja) trójwymiarowego tensora mnożenia macierzy:

$T = \sum_{r=1}^R \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$ , gdzie  $\mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$  to iloczyn kartezjański wektorów,  $R$  - minimalizowana liczba mnożeń.

- Problem jest sformułowany w postaci wieloetapowego procesu decyzyjnego (gry ze środowiskiem) polegającej na odejmowaniu kolejnych iloczynów kartezjańskich od początkowego tensora aż do uzyskania tensora zerowego lub osiągnięcia maksymalnej liczby kroków:  $S_t \leftarrow S_{t-1} - \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$ , gdzie  $S_0 = T$ .
- Nagrody to -1 za każdy krok i specjalna ujemna wartość zależna od rzędu tensora końcowego (im mniejszy, tym lepiej), gdy wyczerpana zostaje maksymalna liczba kroków.
- Uczenie odbywa się z użyciem głębokiej sieci neuronowej  $f_\theta(s) = (\pi, z)$  reprezentującej funkcję użyteczności  $Q$  (w artykule  $z(\cdot|s)$ ) oraz funkcję strategii  $\pi(s, a)$  typu transformer podzielonej na 3 części: korpus (*torso*), fragment reprezentujący strategię (*policy head*) i fragment reprezentujący funkcję wartości (*value head*).
- W trakcie podejmowania decyzji (wyboru wektorów  $\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}$ ) uruchamiane są symulacje MCTS. Akcje są wyznaczone ze wzoru:

$a = \operatorname{argmax}_a Q(s, a) + c(s) \hat{\pi}(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)}$ , gdzie  $N(s, a)$ -liczba wykonań akcji  $a$  w stanie  $s$ ,  $c(s)$  - współczynnik eksploracji,  $\hat{\pi}(s, a) = \frac{N(s, a)}{N(s)}$ -strategia empiryczna.

- Funkcją straty dla *policy head* jest odległość KL pomiędzy wyjściem sieci, a strategią empiryczną lub rozkładem prawdop.akcji w syntetycznych demonstracjach.

# Algorytm MuZero - z uczeniem modelu środowiska (*model-based*)



Algorytm MuZero jest podobny do [AlphaZero](#) i w pewnym stopniu jego uogólnieniem dla zadań, w przypadku których nie jest znany dokładny model środowiska (reguły gry) w postaci kolejnych stanów (i prawdopodobieństw przejść pomiędzy stanami), możliwych akcji czy nagród. Nie można więc przeprowadzić symulacji MCTS bez wcześniejszego zbudowania (wytrenowania) takiego modelu.

## Literatura:

1. Schrittwieser, J., Antonoglou, I., Hubert, T. et al. Mastering Atari, Go, chess and shogi by planning with a learned model. Nature 588, 604–609 (2020). <https://doi.org/10.1038/s41586-020-03051-4>
2. <https://medium.com/applied-data-science/how-to-build-your-own-muzero-in-python-f77d5718061a>

- Trzy sieci neuronowe do aproksymacji:
  - funkcji predykcji  $f : s \rightarrow p, v$ ,
  - funkcji dynamiki (modelu)  $g : s, a \rightarrow r, s$ ,
  - funkcji reprezentacji  $h : o \rightarrow s$ ,

gdzie  $s$  oznacza stan aproksymowany,  $o$  jest wektorem obserwacji.

- Symulacje MCTS przebiegają podobnie jak w AlphaZero, poza tym, że stan początkowy  $s_0$  jest aproksymowany za pomocą funkcji  $h$  na podstawie obserwacji, a stany następne i nagrody są wyznaczane za pomocą aproksymatora  $g$ . W ten sposób system tworzy wewnętrzną reprezentację stanów i nagród sprzyjającą uzyskiwaniu dobrych wyników. Nie jest sprawdzana legalność akcji podczas symulacji MCTS.

- Akcja w MCTS wybierana jest według wzoru:

$$a^k = \operatorname{argmax}_a \left[ Q(s, a) + P(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} C(s) \right], \text{ gdzie}$$

$C(s) = c_1 + \log \frac{\sum_b N(s, b) + c_2 + 1}{c_2}$ ,  $Q(s, a)$ -użyteczność akcji obliczana w trakcie symulacji jak średni zwrot,  $P(s, a)$ -prawdopodobieństwo wyboru akcji z modelu neuronowego,  $N(s, a)$ -liczba wykonań akcji  $a$  w symulacjach MCTS.

- Po wykonaniu wielu symulacji MCTS, akcja  $\alpha$  wykonywana w prawdziwym środowisku jest wybierana z prawdopodobieństwem  $p_\alpha = \frac{N(\alpha)^{1/T}}{\sum_b N(b)^{1/T}}$ .

- Uczenie wszystkich trzech funkcji (sieci neuronowych) odbywa się równolegle na podstawie wcześniej zapisanych gier *self-play* w pamięci powtórek (*replay buffer*).
- Funkcja straty  $l_t(\theta) = \sum_{k=0}^K l^r(u_{t+k}, r_t^k) + l^v(z_t + k, v_t^k) + l^p(\pi_{t+k}, \mathbf{p}_t^k) + c \|\theta\|^2$ , gdzie  $K$ -liczba ruchów (kroków epizodu) od kroku  $t$ ,  $u, z, \pi$  - odpowiednio otrzymana bieżąca nagroda, bieżący zwrot oraz rozkład prawdopodobieństwa akcji otrzymany w korzeniu drzewa MCTS po symulacjach,  $r, v, \mathbf{p}$  - wartości zwracane przez sieć.
- Architektura sieci - z warstwami splotowymi i połączeniami rezyduальnymi w przypadku funkcji reprezentacji  $h$  i dynamiki  $g$  jest prawie taka sama jak architektura sieci w AlphaZero. Architektura funkcji predykcji  $f$  jest dokładnie taka sama jak w AlphaZero (wewnętrzny stan  $s$  ma taką samą strukturę co rzeczywisty  $o$ ).

- Algorytm MuZero osiągnął lepsze wyniki w grze GO od AlphaZero pomimo rezygnacji ze znajomości zasad gry w trakcie symulacji MCTS. Jest to obiektem badań, ale wydaje się, że wewnętrzna reprezentacja stanów pozwala na wyodrębnienie i użycie cech stanu gry bardziej istotnych dla jego oceny, a wewnętrzny system pośrednich nagród przyspiesza proces uczenia.
- Algorytm MuZero, w przeciwieństwie do AlphaZero, można zastosować w przypadku interakcji uczącego się agenta z rzeczywistym środowiskiem w sytuacji braku prostego symulatora środowiska (np. sterowanie autonomicznym samochodem, zarządzanie rzeczywistą firmą).
- Algorytm MuZero jest bardziej zbliżony do ludzkiego sposobu uczenia się, gdyż uczy się jednocześnie strategii i zasad gry oraz posiada wewnętrzny model świata.





POLITECHNIKA  
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,  
TELEKOMUNIKACJI  
I INFORMATYKI

# Dziękuję za uwagę.

Jerzy Dembski



Fundusze  
Europejskie  
Polska Cyfrowa



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz  
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

„Technologia” – Akademia Innowacyjnych Technologii (AI Tech) w