Zaawansowane Metody Inteligencji Obliczeniowej

Wykład 9: AlphaGo i AlphaZero

Michał Kempka Marek Wydmuch

9 maja 2022

Fundusze Europejskie
Polska Cyfrowa

Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Rozwoju Regionalnego

"Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)",
projekt finansowany ze środków Programu Operacyjnego Polska Cyfrowa POPC.03.02.00-00-0001/20

Plan wykładu

1. Granie w gry na ludzkim poziomie
   - TD-Gammon i self-play
   - Deep Q-Learning i głębokie sieci nieuronowe
2. AlphaGo i AlphaZero
   - AlphaGo
   - AlphaGo Zero/Alpha Zero

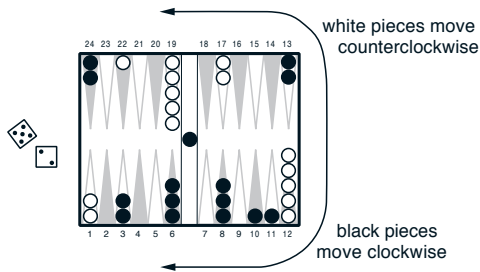
Granie w gry lepiej od ludzi

Przykłady gier i algorytmów, opartych o techniki, które znamy i które osiągnęły mistrzowski poziom:

- Szachy — Deep Blue (wariant Minimax z heurystykami)
- Tryktrak — TD-Gammon (TD(λ) z siecią neuronowa jako estymatorem wartości i gra ze samym sobą (ang. self-play))
- Gry Atari — Deep Q-Learning (DQN) (usprawniony Q-Learning z siecią konwolucyjną jako estymatorem wartości)

Tryktrak

- Popularna gra na całym świecie.
- Każdy gracz ma 15 pionów, a plansza ma 24 pola.
- Cel: Przemieścić wszystkie swoje piony za planszę po drugiej stronie.
- W swojej turze gracz rzuca dwiema kostkami, każda kostka pozwala ruszyć jeden pion o wyznaczoną liczbę pól do przodu.
- Gracze mogą zbijać samotne piony przeciwnika stając na nich swoimi pionami.
- Podsumowując: niedeterministyczna gra z duża ilość możliwych ruchów.



Rysunek: Schemat gry w tryktrak

Neurogammon (Tesauro, 1989)

- Sieć neuronowa z ręcznie zaprojektowanymi cechami specjalnie dla gry w tryktrak.
- Uczona w sposób nadzorowany na przykładach od ekspertów.
- Wygrał World Backgammon Olympiad w 1989 roku.

TD-Gammon 0.0 (Tesauro, 1992)

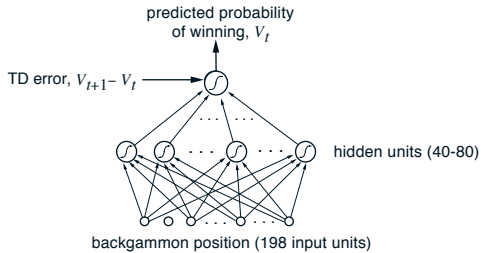
:

- Sieć neuronowa wyłącznie z prostą reprezentacją planszy (bez wiedzy dziedzinowej), estymująca prawdopodobieństwo wygrania dla danego stanu.
- Uczona za pomocą TD(λ), z nagrodą tylko na końcu gry (0 albo 1, bez dyskontowania = estymacja prawdopodobieństwa wygrania).
- **Na podstawie 300000 gier rozegranych sam ze sobą (ang. self-play)**
- Lepszy niż Neurogammon.

Architektura TD-Gammon 0.0

- cechy (198 wejść):
  - dla każdego pola i koloru 4 wejścia $(2 \times 4 \times 24 = 192)$:
    - 0 pionów $\rightarrow \{0, 0, 0, 0\}$;
    - 1 pion $\rightarrow \{1, 0, 0, 0\}$;
    - 2 piony $\rightarrow \{0, 1, 0, 0\}$;
    - 3 piony $\rightarrow \{0, 0, 1, 0\}$;
    - n>3 pionów $\rightarrow 0, 0, 0, (n-3)/2$;
  - 2 wejścia: liczba pionów danego koloru na bandzie,
  - 2 wejścia: liczba pionów w domu,
  - 2 wejścia: czyj ruch (0,1 i 1,0).
- Jedna warstwa ukryta z 40 neuronami (więcej w kolejnych wersjach).
- Wyjście określające prawdopodobieństwo wygranej.

:



Rysunek: Schemat architektury TD-Gammon 0.0

Self-play w TD-Gammon 0.0

:

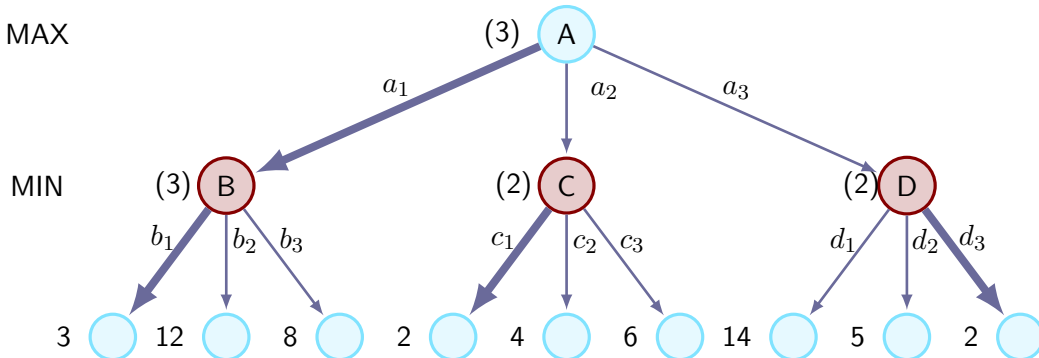
- Zamiast uczyć się z przykładów dostarczonych przez ekspertów (ograniczony zbiór) lub grania z ludzkim graczem (czasochłonne), TD-Gammon uczył się wyłącznie poprzez granie ze samym sobą.
- Początkowo wagi modelu były inizjalizowane losowo.
- Model grał sam ze sobą (wybierając na zmianę ruch dla jednego i drugiego gracza) i dokonując na bieżąco aktualizacji wag zgodnie z algorytmem TD(λ).

TD-Gammon 1.0, 2.0+ (Tesauro, 1994, 1995, 2002)

:

- 1.0: +dodatkowo ręcznie zaprojektowane cechy.
- 1.0, 2.1+: +zwiększono liczbę neuronów z w warstwie ukrytej z 40 do 80 (w 1.0 i 2.1) i do 160 (w 3.0+).
- 2.0: zamiast wybierać akcję z największą wartością dodany został algorytm przeszukiwania drzewa stanów (Expected Minimax z beam-searchem).
- 3.0+: zwiększono głębokość przeszukiwania drzewa.

Minimax



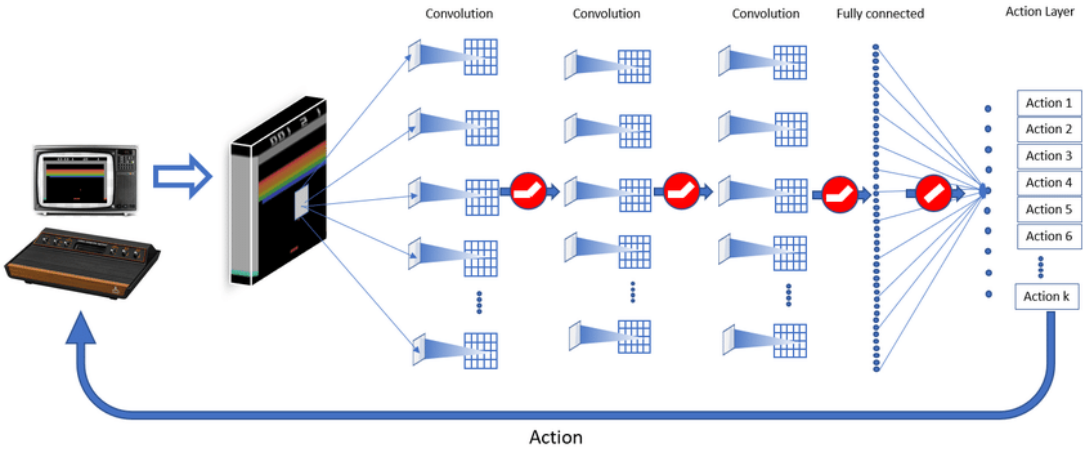
Rysunek: Przykład algorytmu Minimax.

Expected (Stochastic) Minimax/Alpha-beta pruning

- Algortym Minimax można także zastosować do stochastycznych środowisk poprzez wprowadzenie dodatkowych wierzchołków tam gdzie następują zdarzenia losowe i wyliczaniu wartości oczekiwanej wyniku (Expected-Minimax):

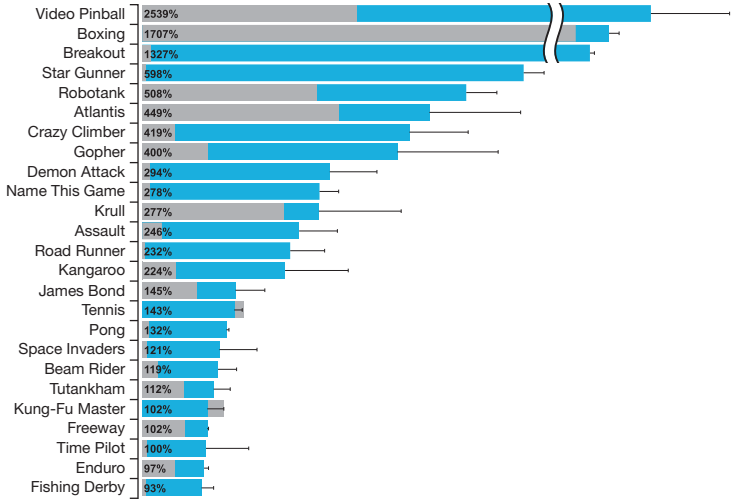
$$\text{E-Minimax}(s) = \begin{cases} \text{CałkowitaNagroda}(s) & \text{if StanTerminalny}(s) \\ \max_{a \in \mathcal{A}_s} \text{E-Minimax}(\text{Nast.Stan}(s, a)) & \text{if Gracz}(s) = \text{MAX} \\ \min_{a \in \mathcal{A}_s} \text{E-Minimax}(\text{Nast.Stan}(s, a)) & \text{if Gracz}(s) = \text{MIN} \\ \sum p(s', r|a, s) \times \text{E-Minimax}(s') & \text{if Gracz}(s) = \text{CHANCE} \end{cases}$$

DQN – granie w gry wideo lepiej od ludzi? (Mnih et al., 2015)

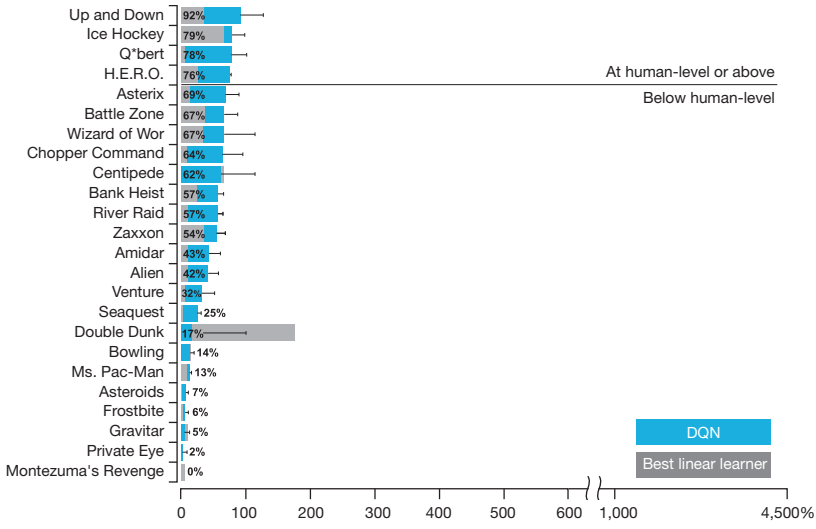


Rysunek: Schemat architektury użytej przez Mnih et al., 2015.

DQN – Wyniki (Mnih et al. 2015)



DQN – Wyniki (Mnih et al. 2015)



DQN – Granie w gry wideo lepiej od ludzi? (Mnih et al., 2015)

- DQN osiągnął lub przewyższył poziom ludzkiego gracza w 29 z 49 gier ze zbioru Atari.
- W grach takich jak Montezuma's Revenge, która wymaga planowania w przód z dużym wyprzedzeniem radzi on sobie jednak porównywalnie do losowego agenta.

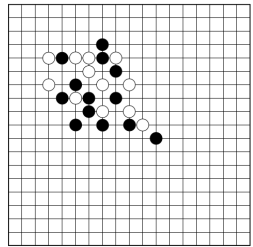
Plan wykładu

1. Granie w gry na ludzkim poziomie
   - TD-Gammon i self-play
   - Deep Q-Learning i głębokie sieci nieuronowe

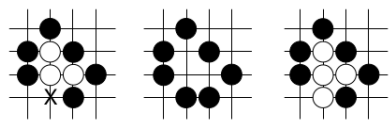
2. AlphaGo i AlphaZero
   - AlphaGo
   - AlphaGo Zero/Alpha Zero

Go

- Starożytna chińska gra stanowiła wyzwanie dla sztucznej inteligencji of wielu dekad.
- Gra na planszy o rozmiarze 19x19 (gra się też na mniejszych planszach 9x9 i 13x13).
- Proste zasady: gracze na zmianą kładą kamienie na plansz, kamienie zostają przejęte jeśli zostaną otoczone przez grupę kamieni przeciwnika.
- Cel gry: kontrolowa (otoczyć) jak największy obszar planszy/przejąć jak najwięcej kamieni przeciwnika.



Rysunek: Schemat gry w go

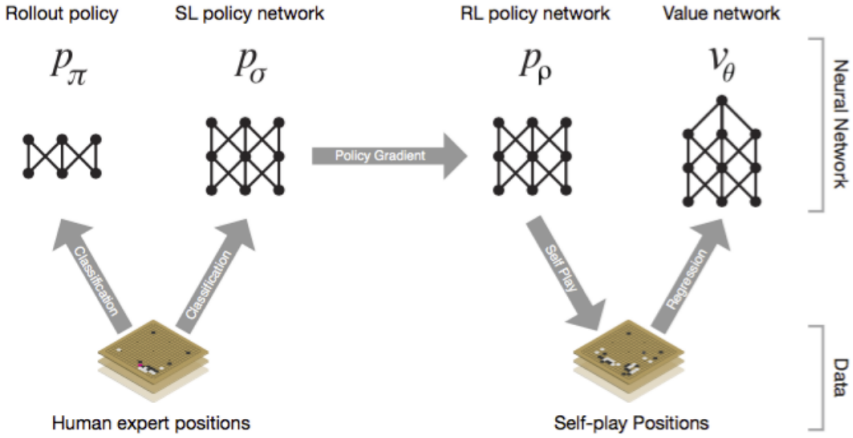


Rysunek: Przejmowania obszaru w go

AlphaGo

- Metody, które pozwalały osiągnąć poziom ludzkiego gracza w innych grach nie sprawdziły się w przypadku Go (ogromna przestrzeń stanów, ogromny ilość możliwych ruchów (ang. branching factor), konieczność planowania wiele ruchów do przodu).
- Arcymistrzowski poziom gry w go udało się osiągnąć grupie z DeepMind (Silver et al., 2016) za pomocą programu AlphaGo.
- AlphaGo pokonał w 2016 roku 18-krotnego mistrza świata w Go — Lee Sedola 4:1.
- AlphaGo wykorzystuje kombinację uczenia nadzorowanego, Monte Carlo Tree Search, uczenia ze wzmocnieniem i granie ze samym sobą.

AlphaGo — schemat uczenia i architektura



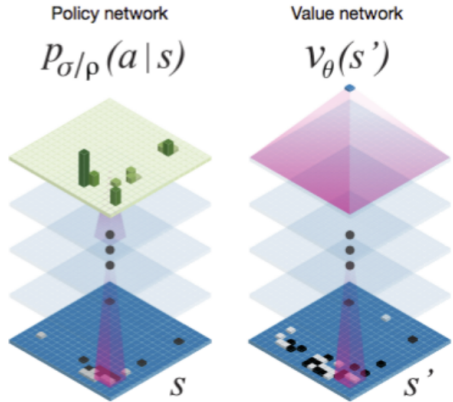
Rysunek: Schemat uczenia i architektura AlphaGo

AlphaGo — wykorzystane sieci i ich uczenie

- Rollout policy network (p_π) — prosta sieć polityki rolloutu, nauczona poprzez uczenie nadzorowane na zbiorze 30 mln. pozycji (z 160 tyś. gier graczy od 6 do 9 dana) i ruchów wykonanych przez ludzkich graczy (celem sieci było przewidywanie ruchu ludzkiego gracza). Trenowana na 50 GPU przez 1 dzień.
- SL (select) policy network (p_σ) — sieć polityki wyboru ruchu, uczona tak samo jak sieć polityki rolloutu. Trenowana na 50 GPU przez 3 tygodnie.
- RL policy network (p_ρ) — sieć uczenia ze wzmocnieniem, uczona uczona poprzez granie sama ze sobą. Trenowana na 50 GPU przez 1 dzień.
- Value network v_θ — sieć uczona na zbiorze zebranym podczas grania ze samym sobą najlepszej wersji RL policy network (30 mln. pozycji). Trenowana na 50 GPU przez 1 tydzień.

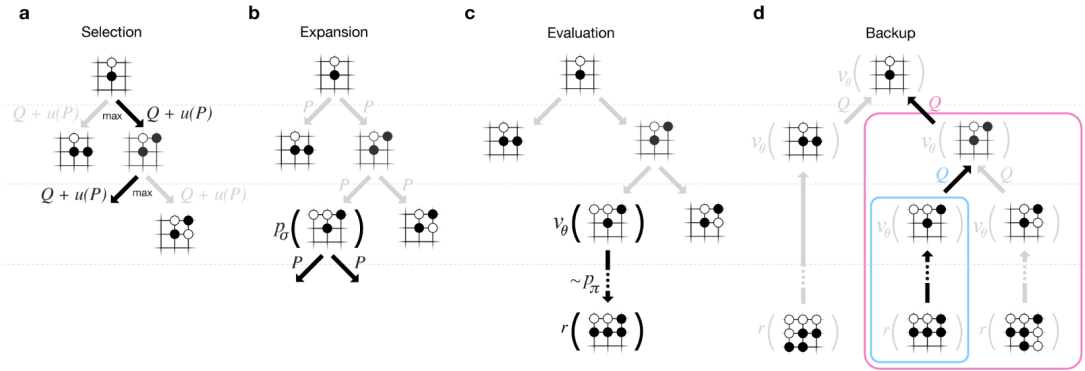
AlphaGo — architektura sieci

- Jako wejście sieci był zbiór macierzy (tensor) $19 \times 19 \times 48$ cech binarnych lub liczb całkowitych (inaczej mówiąc, było 48 cech na dla każdego pola).
- Pierwsze kilka macierzy reprezentowało planszę — cechy typu czy pole jest zajęte przez czarny kamień, czy przez biały, czy puste.
- Kolejne cechy stworzone przez ekspertów — np. ile kamieni zostanie przejętych po wykonaniu tego ruchu.



Rysunek: Schemat architektur SL/RL policy network i Value network w AlphaGo

AlphaGo — Monte Carlo Tree Search



Rysunek: Schemat MCTS zastosowany w AlphaGo

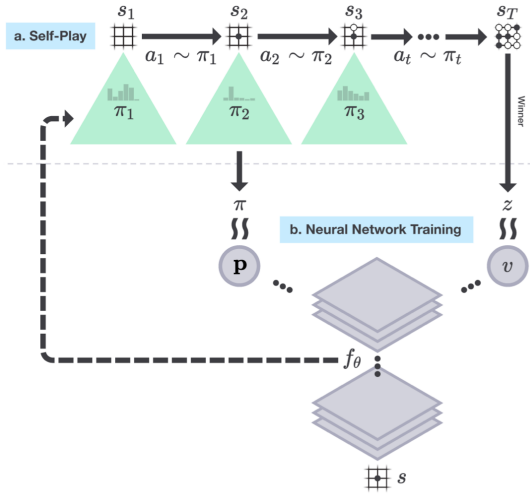
AlphaGo — Monte Carlo Tree Search

- Wykorzystuje sieci p_π, p_σ oraz v_θ
- Selekcja odbywa się na podstawie estyma wartości z v_θ powiększoną o dodatkową losową wartość ($u(P)$) by zapewnić eksploracje. Wartość ta zależy od prawdopodobieństwa akcji zgodnie z p_π oraz ilości odwiedzin stanu i akcji.
- Ekspansja zostaje dokonana na podstawie p_σ (nie wszystkie akcje jesteśmy wstanie odwiedzić, więc lepiej wybierać te, które mają wysoką szanse zostać wybrane przez ludzkiego gracza).
- Rollout zostaje wykonany zgodnie z polityką p_π (szybka polityka)
- Ostateczna wersja AlphaGo używała przeszukiwania na 48 CPU i 8 GPU.

AlphaGo Zero (Silver et al., 2017)

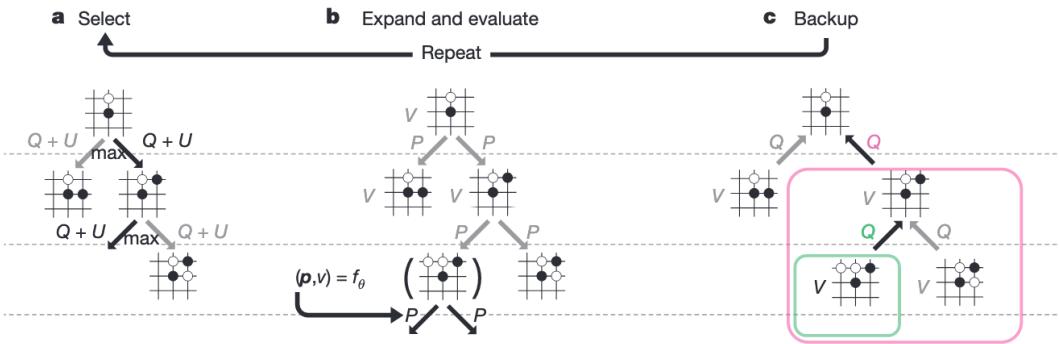
- Ewolucja AlphaGo, która uczy się bez początkowego zbioru danych od ekspertów — ludzki graczy oraz skonstruowanych przez nich pomocniczych cech.
- Jako reprezentacje używany był tensor $9 \times 19 \times 17$ gdzie cześć cech była reprezentacją planszy tak jak w AlphaGo, a reszta stanowiła historię poprzednich ruchów (stan gry w Go nie jest stanem Markova — nie zawiera wszystkich informacji).
- Zamiast wielu sieci wykorzystywana była jedna z dwoma „głowami" — zestawami wyjść, jednym estymującymi prawdopodobieństwa akcji (policy output), a jedno wartość stanu (value output).

AlphaZero — schemat uczenia i architektura



Rysunek: Schemat uczenia Alpha Zero

AlphaZero — MCTS



Rysunek: Schemat MCTS w Alpha Zero

AlphaGo Zero (Silver et al., 2017)

- Po zakończeniu gry i otrzymaniu nagrody następuje uaktualnienie wag sieci.
- Sieć jest uczona za pomocą błędu, który jest sumą MSE estymat wartości stanu (value output) v a nagrody z i cross entropy pomiędzy estymatami polityki (policy output) $\boldsymbol{p}$ a wartościami otrzymanymi z przeszukiwania drzewa stanów $\boldsymbol{\pi}$:

$$l = (z - v)^2 - \boldsymbol{\pi} \log \boldsymbol{p} + c||\theta||^2$$

- $c||\theta||^2$ to dodatkowe wyrażenie regularyzujące wagi sieci.

Bibliografia

[1] Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition.

[2] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Fundusze Europejskie
Polska Cyfrowa

Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Rozwoju Regionalnego

"Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)",
projekt finansowany ze środków Programu Operacyjnego Polska Cyfrowa POPC.03.02.00-00-0001/20