Uczenie maszynowe *end-to-end* na przykładzie programu *DeepChess*

Stanisław Kaźmierczak

Agenda

- Wprowadzenie
- Dane treningowe
- Trening i struktura sieci
- Redukcja rozmiaru sieci
- Zmodyfikowane alfa – beta
- Wyniki
- Wnioski

Wprowadzenie

- Najsilniejsze programy szachowe nie używają lub używają w bardzo ograniczonym zakresie uczenia maszynowego
- O sile programu stanowi w głównej mierze szlifowana latami funkcja oceny pozycji
- Zastosowanie metod uczenia maszynowego w szachach przynosiło dobre, aczkolwiek nie rewelacyjne wyniki
- *Reinforcement learning*:
  - Poziom mistrza szachowego [2, 3]
  - Dobrze radził sobie w mniej złożonych grach jak warcaby czy backgammon
  - Formalne argumenty przemawiające za tym, że *Reinfocement learning* słabo radzi sobie w bardziej złożonych grach jak szachy (*branching factor* = 35) [4]

Wprowadzenie

- *Monte Carlo*:
  - Stanowiło znaczne wzmocnienei w Go [5]
  - Nie przynosi rewelacyjnych rezultatów w szachach ze względu na ich zbyt taktyczny charakter, np. tylko jeden ruch prowadzi do zwycięstwa, a wszystkie pozostałe do porażki
- Do tej pory nie istniał program wykorzystujący uczenie maszynowe od początku do końca grający na poziomie arcymistrzowskim
- 3 fazy nauki *DeepChess*:
  - Głębokie nienadzorowane uczenie w celu ekstrakcji cech wysokiego poziomu
  - Nauka sieci, aby umiała wybierać lepszą z dwóch pozycji
  - Kompresja sieci w celu przyspieszenia obliczeń

Funkcja oceny pozycji

- Standardowowe funkcje oceny pozycji w szachach:
  - Wejściem jest pozycja, wyjściem liczba
  - Wartość funkcji reprezentuje jak dobra jest pozycja, na ogół z punktu widzenia białego (przewaga białych – wartość dodatnia, pozycja równa – 0, przewaga czarnych – wartość ujemna)
  - kombinacja liniowa kilkuset (lub więcej) parametrów (*Deep Blue* – 8k), np. suma umownej wartości bierek, bezpieczeństwo króla, zaawansowanie pionów, zdublowane piony, kontrola centrum, itp.
  - Tworzone ręcznie na podstawie wiedzy eksperckiej
  - Szlifowane latami
- Funkcja oceny pozycji w *DeepChess*:
  - Tworzona od zera, brak jakiejkolwiek wiedzy eksperckiej
  - Do nauki wykorzystywane jest jedynie zbiór partii wraz z rezultatem danej rozgrywki

Funkcja oceny pozycji w *DeepChess*

- Idea: porównywanie dwóch pozycji, brak numerycznej oceny
- Wejście: dwie pozycje
- Sieć neuronowa uczy się określać, która pozycja jest lepsza
- Wybór par pozycji w procesie uczenia:
  - Jedna pozycja wybierana jest losowo z partii, w której białe wygrały
  - Druga wybierana jest z partii, w której zwyciężyły czarne
  - Założenie: w zdecydowanej większości przypadków pozycje, w których białe wygrały są lepsze (z punktu widzenia białych) od tych, w których białe przegrały
- Powyższe podejście pozwala stworzyć duży zbiór treningowy:
  - np. mając 10^6 pozycji, w których białe wygrały i 10^6 pozycji, w których białe przegrały możemy stworzyć 2×10^{12} par uczących (x2, bo każda para pozycji może być użyta dwa razy: wygrana – przegrana, przegrana – wygrana)

Dane treningowe

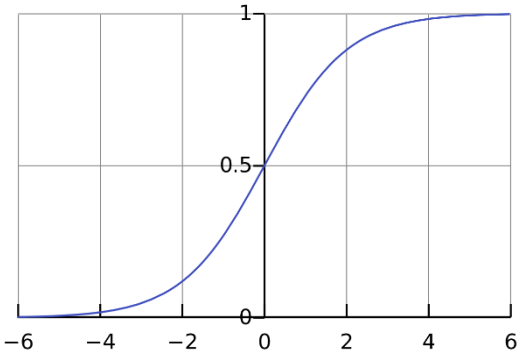
- Źródło: CCRL (www.computerchess.org.uk/ccrl)
- 640 000 partii (221 695 wygranych przez białe, 164 387 – przez czarne, reszta to remisy)
- Eksperymenty pokazały, że partie remisowe nie wnoszą nic w proces nauki
- Z każdej partii wybieranych jest losowo 10 pozycji, przy czym:
  - Nie bierzemy pozycji z pierwszych pięciu ruchów
  - Ruch wykonany z wybranej pozycji nie jest biciem (bicia są mylące, ponieważ skutkują chwilową „przewagą”)
- Ostatecznie zbiór treningowy zawiera 3 860 820 pozycji (2 216 950, w których białe wygrały i 1 643 870 wygranych przez czarne)

Trening i struktura sieci

- Faza 1: trening autoenkodera
  - Autoenkoder działa jak nieliniowy ekstraktor cech wysokiego poziomu
- Faza 2: nauka porównywania pozycji
  - Tworzymy dwie kopie autoenkodera (odpowiadające dwum pozycjom)
  - Na ich wyjściu dodajemy w pełni połączoną kilkuwarstwową sieć
  - Dwa neurony na wyjściu,

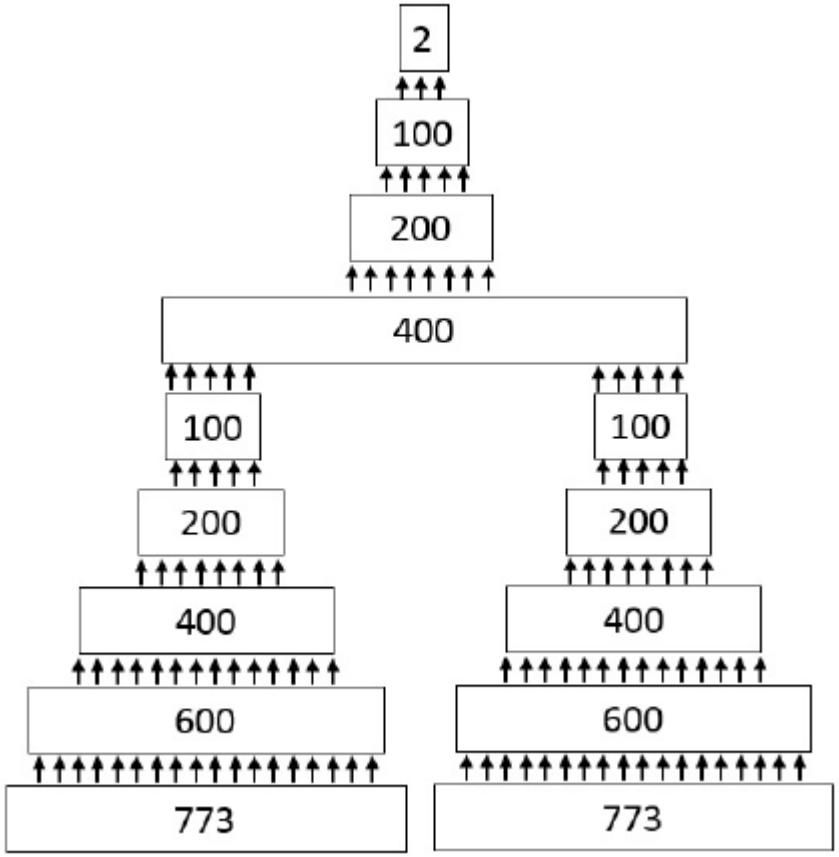
    funkcja aktywacji – softmax
  - Wyjście wskazuje, która z

    pozycji pochodzi jest lepsza



  - Okazuje się, że dwuwartościowe wyjście daje lepsze rezultaty niż jednowartościowe wyjście binarne

Trening i struktura sieci

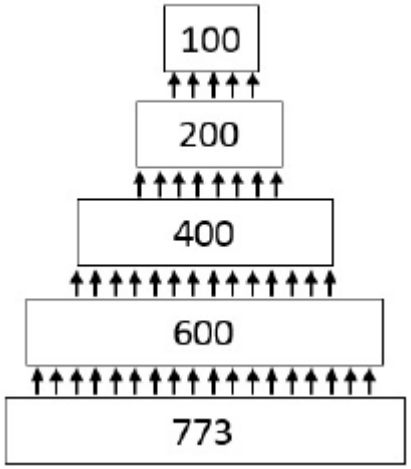


Wejście sieci

- 773-bitowe wejście
  - Na każdym z 64 pól może stać jedna z 6 figur koloru białego lub czarnego (64 x 6 x 2 = 768)
  - Czyj ruch (1 bit)
  - Prawo do roszady (4 bity)

Trening autoenkodera

- Każde dwie sąsiadujące warstwy traktowane są jako autoenkoder
- Kolejno trenujemy autoenkodery z coraz niższych warstw (773-600-773, 600-400-600, itd.)
- Dopóki w pełni zostanie nauczona dana warstwa nie przechodzimy do kolejnej
- Zbiór treningowy: milion pozycji wygranych przez białe i milion wygranych przez czarne
- Funkcja aktywacji ReLU: $f(x) = \max(0, x)$
- 200 iteracji



Trening *DeepChess*

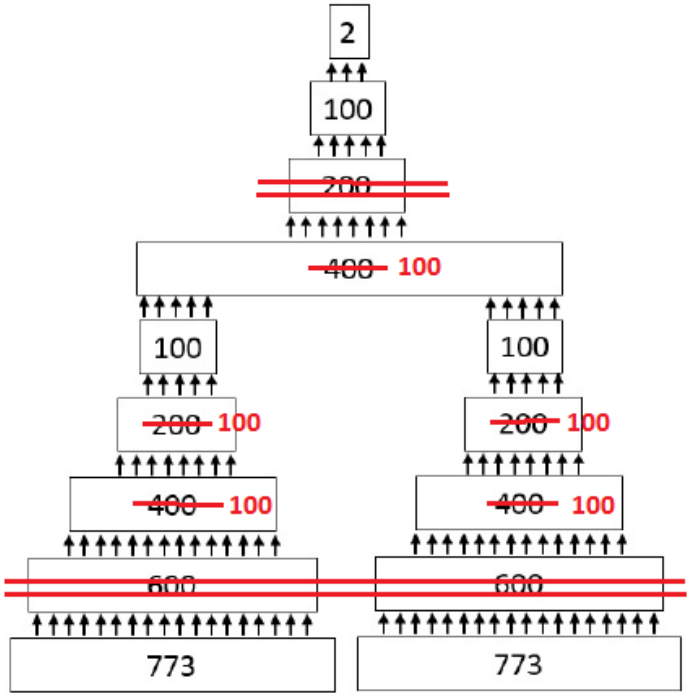
- Trening nadzorowany, podczas którego modyfikowana jest cała sieć, również autoenkoder
- 1000 iteracji
- W każdej iteracji generowanych jest losowo milion par
  - Jedna pozycja wybierana jest losowo ze zbioru 2 116 950 pozycji wygranych przez białe
  - Jedna pozycja wybierana jest losowo ze zbioru 1 543 870 pozycji wygranych przez czarne
  - Każda para pozycji jest następnie ustawiana losowo jako wygrana-przegrana lub przegrana-wygrana
  - 100k pozycji każdego typu jest przeznaczonych na walidację

Trening *DeepChess*

- Zbiór potencjalnych par treningowych ma liczność 6.5×10^{12}, więc pary wejściowe praktycznie się nie powtarzają (i tym samym nie nastąpi przeuczenie)
- Funkcja aktywacji ReLU: $f(x) = \max(0, x)$
- Współczynnik nauki wynosi 0.01 na starcie i wymnażany jest przez 0.99 w każdej iteracji
- Błąd na zbiorze treningowym i walidacyjnym wyniósł odpowiednio 1.8% oraz 2.0%

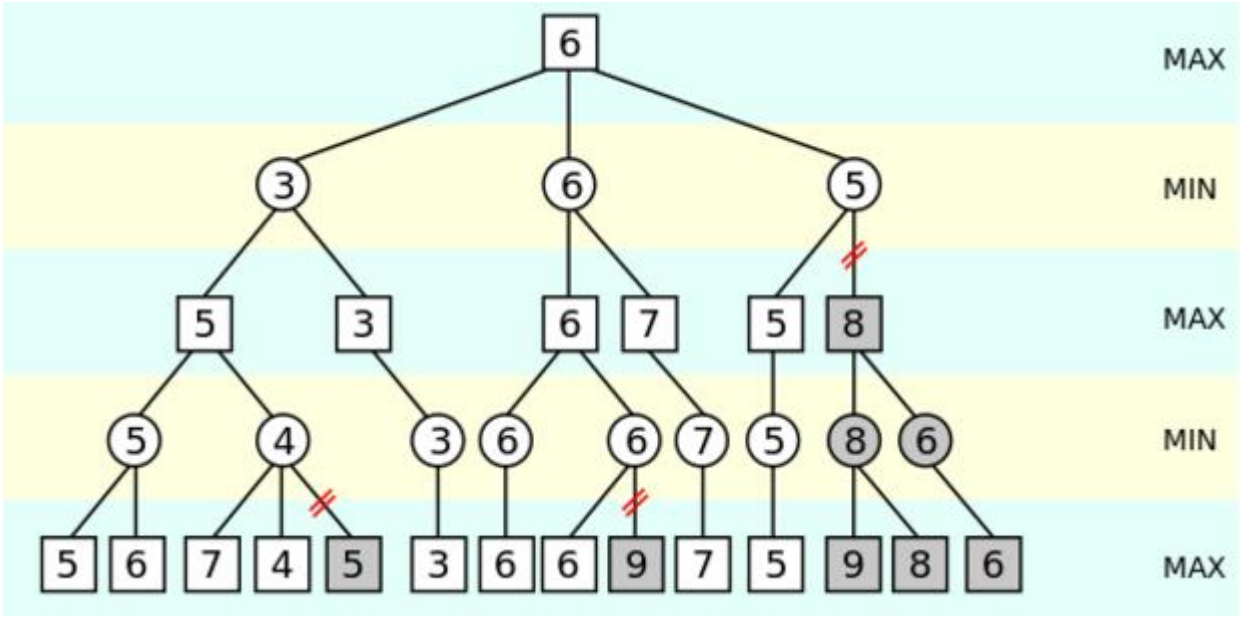
Redukcja rozmiaru sieci

- Sieć w zaprezentowanej postaci działa znacząco wolniej niż standardowe funkcje ewaluacyjne
- Należy stworzyć mniejszą sieć, która w dobry sposób będzie naśladować oryginalną sieć
- Na początku trenowana jest sieć 773-100-100-100 w celu naśladowania autoenkodera z oryginalnej sieci
- Następnie dodawane są 3 warstwy: 100-100-2 i trenowana jest cała sieć



Method	Accuracy
Uncompressed	98.0%
Compressed	97.1%
Small	95.4%

Zmodyfikowane alfa-beta



- Wartości alfa i beta są zastępowane pozycjami $alfa_{pos}$ i $beta_{pos}$
- Przypadek, gdy my mamy zagrać w węźle: jeśli nowa pozycja jest lepsza niż $alfa_{pos}$ wówczas staje się ona $alfa_{pos}$
- Przypadek, gdy przeciwnik ma zagrać w węźle: jeśli nowa pozycja jest gorsza niż $beta_{pos}$ wówczas staje się ona $beta_{pos}$
- Jeśli $alfa_{pos}$ jest lepsza w danym węźle niż $beta_{pos}$ wówczas przerwij dalsze przeszukiwania z tego węzła

Haszowanie pozycji

- Wiele pozycji powtarza się w różnych częściach drzewa gry
- W celu uniknięcia zbędnych obliczeń zastosowana została tablica haszująca przechowująca pozycje i odpowiadające im wartości wyeksrahowanych cech
- Dla każdej nowej pozycji najpierw sprawdzana jest tablica haszująca.
- Jeśli okaże się, że odpowiedni hasz już istnieje, wówczas wykorzystywane są wcześniej policzone wartości

Czy program uczy się reguł gry?

- *Without any a priori knowledge,* ***in particular without any knowledge regarding the rules of chess***
- *We employ deep neural networks to learn an evaluation function from scratch****, without incorporating the rules of the game*** *and using no manually extracted features at all*
- *We do* ***not provide*** *our evaluation function with any features,* ***including any knowledge about the rules of chess***
- *This is remarkable, considering that* ***no*** *a priori knowledge of chess,* ***including the very rules of the games*** *are provided*
- *without any chess knowledge whatsoever (****not even basic knowledge as the rules of chess****)*

Czy program uczy się reguł gry?

Moim zdaniem te wszystkie zdania, które Pan przytoczył odnoszą się wyłącznie do funkcji ewaluacyjnej, czyli w tym przypadku do tej sieci neuronowej, a nie do całego programu. Rzeczywiście autorzy trenują sieć neuronową bez żadnych dodatkowych informacji o zasadach gry, wyłącznie na podstawie porównywania dwóch różnych pozycji.

Logika gry była zaimplementowana już w alogorytmie alpha-beta, który po prostu wykorzystywał DeepChees jako funkcje ewaluacyjną, ale równie dobrze mogłaby być to funkcja FALCON czy CRAFTY, z którymi właśnie porównywany jest DeepChees.

Statyczne rozumienie pozycji

- Program dobrze porównuje pozycje pod względem materiału (zarówno proste przypadki, gdy po jednej stronie brakuje jakiejś bierki, jak i bardziej wyszukane np. wieża vs skoczek i goniec)
- *DeepChess* nauczył się dobrze rozpoznawać również bardziej subtelne aspekty pozycji jak np. bezpieczeństwo króla, para gońców, mobilność bierek, zaawansowanie pionków, prawo do roszady, itp.
- Preferuje dynamiczne pozycje z możliwościami ataku nawet kosztem materiału
- W wielu przypadkach wybiera pozycji z pionkiem lub dwoma mniej, ale z przewagą pozycyjną (charakterystyczna cecha silnych graczy ludzkich)
- Na podobieństwo do zachowania mocnych szachistów ma wpływ niewątpliwie nieliniowość ewaluatora *DeepChess*

Statyczne rozumienie pozycji

Tal - Larsen
Move: **Nd5**

Aronian - Leko
Move: **Re5**

Alekhine -Golombek
Move: **d5**

Seirawan - Kozul
Move: **c5**

Wyniki

Match	Result	RD
DeepChess 30min - CRAFTY	59.0 - 41.0	+63.2
DeepChess 30min - FALCON	51.5 - 48.5	+10.4
DeepChess 120min - FALCON	63.5 - 36.5	+96.2

- *Falcon*
  - Poziom arcymistrzowski
  - Funkcja oceny pozycji rozwijana prawie 10 lat zawiera ponad 100 parametrów
- *Crafty*
  - Ranking ELO: 3033 (10.05.2017)
  - Standardowy program używany jako referencja, rozwijany od 30 lat
- *DeepChess* działa 4 razy wolniej niż funkcja ewaluacyjna w *Falcon*
  - *DeepChess* jest istotnie lepszy niż funkcja ewaluacyjna w *Falcon*

Wnioski

- Gdzie szukać dalszych możliwości poprawy siły gry programu?
  - Poprzez przyspieszenie działania sieci neuronowej DeepChess bez pogarszania jej wyników
- Czym *DeepChess* różni się od gracza ludzkiego?

Wnioski

- Gdzie szukać dalszych możliwości poprawy siły gry programu?
  - Poprzez przyspieszenie działania sieci neuronowej DeepChess bez pogarszania jej wyników
- Czym *DeepChess* różni się od gracza ludzkiego?

SELEKTYWNOŚĆ

Bibliografia

1. D. E. Omid, N. S. Netanyahu, L. Wolf. *DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess*. ICANN (2) 2016: 88-96
2. J. Baxter, A. Tridgell, L. Weaver. Learning to play chess using temporal-dierences. *Machine Learning*, 40(3):243-263, 2000.
3. M. Lai. Girae. *Using deep reinforcement learning to play chess*. Master's Thesis, Imperial College London, 2015.
4. M.A. Wiering. *TD learning of game evaluation functions with hierarchical neural architectures*. Master's Thesis, University of Amsterdam, 1995.
5. D. Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484-489, 2016.