Stanisław Kaźmierczak

- Wprowadzenie
- Podejście standardowe
- Sieć ewaluująca pozycję
- Probability-depth search
- Podsumowanie

- Selektywność człowieka vs moc obliczeniowa maszyny
- Większa efektywność obliczeniowa u człowieka
  - Kasparov: 3-5 pozycji na sekundę, Deep Blue: 200 milionów
- Zwiększenie selektywności maszyny stanowi duże wyzwanie
  - Jak określić warunki ucinania/dalszej eksploracji gałęzi?
- Silny gracz nabywa takiej wiedzy dzięki tysiącom rozegranych partii
  - Problem z przełożeniem wiedzy ludzkiej na konkretne reguły komputera
- Założenie: minimalna wiedza wstępna związana z szachami
- *Giraffe* tworzy własne reguły na podstawie partii rozegranych sam ze sobą

Wprowadzenie

- Nacisk położony na:
  - Statystyczną ocenę pozycji (bez patrzenia wprzód)
  - Decyzja o ucięciu/eksploracji gałęzi
  - Sortowanie ruchów i dalsze przeszukiwanie zaczynając od potencjalnie najlepszych
- Sieć neuronowa jako substytut intuicji

Podejście standardowe

Sortowanie ruchów

- Optymalne sortowanie w algorytmie *alfa-beta* pozwala zwiększyć głębokość przeszukiwania dwukrotnie
- Klasyczne sposoby sortowania:
  - Na podstawie wcześniejszych przeszukiwań (mimo, że były płytsze)
  - Na podstawie kolejności ruchów węzła „brata”
  - Bicie figur o wyższej wartości zwiększa ocenę ruchu, bronienie figur o niższej wartości zmniejsza
  - Promocje pionka wysoko oceniane

Funkcja oceny pozycji

- Statyczna ocena pozycji bez przeszukiwania w głąb
- Zdecydowana większość ulepszeń czołowych silników szachowych opiera się na wzmocnieniu funkcji oceny pozycji
- Analiza funkcji ewaluacyjnej stanowi podstawę efektywnej reprezentacji cech dla uczenia maszynowego
- Elementy oceny pozycji (*Stockfish*):
  - Materiał
    - Efekt synergii, np. para gońców
    - Regresja wielomianowa dla bardziej złożonych zależności
  - Tablice figura-pole (*Piece-Square*)

Tablica dla pionków

```
 0,  0,  0,  0,  0,  0,  0,  0,
50, 50, 50, 50, 50, 50, 50, 50,
10, 10, 20, 30, 30, 20, 10, 10,
 5,  5, 10, 27, 27, 10,  5,  5,
 0,  0,  0, 25, 25,  0,  0,  0,
 5, -5,-10,  0,  0,-10, -5,  5,
 5, 10, 10,-25,-25, 10, 10,  5,
 0,  0,  0,  0,  0,  0,  0,  0
```

Tablica dla skoczków

```
-50,-40,-30,-30,-30,-30,-40,-50,
-40,-20,  0,  0,  0,  0,-20,-40,
-30,  0, 10, 15, 15, 10,  0,-30,
-30,  5, 15, 20, 20, 15,  5,-30,
-30,  0, 15, 20, 20, 15,  0,-30,
-30,  5, 10, 15, 15, 10,  5,-30,
-40,-20,  0,  5,  5,  0,-20,-40,
-50,-40,-20,-30,-30,-20,-40,-50,
```

Funkcja oceny pozycji

- Elementy oceny pozycji (*Stockfish*) cd.:
  - Struktura pionów (zaawansowanie, izolacja, wsparcie innych, zdublowanie, itp.)
  - Ocena specyficzna dla rodzaju figury, np. bonus dla skoczka/gońca za bycie na „przyczółku", dla wieży za otwartą linię, itd.
  - Mobilność figur
    - Liczba możliwych ruchów do wykonania przez figurę
    - Pola kontrolowane przez figury przeciwnika o niższej wartości nie są brane pod uwagę
  - Zagrożenia (bonus dla bronionych figur, kara dla niebronionych)
  - Bezpieczeństwo króla (bliskość figur atakujących, osłona pionkowa, możliwość roszady
  - Przestrzeń (bonus za kontrolowane pola)
  - *Drawishness* – pewne kombinacje figur dają dusze szanse na remis

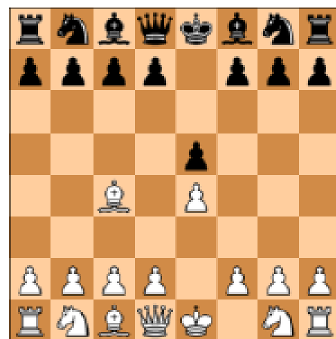
Giraffe

Założenia

- Cel
  - system zdolny przeprowadzać automatyczną wysokopoziomową ekstrakcję cech, bez ograniczeń związanych z ludzką kreatywnością (TDLeaf(λ))
  - Selektywne *probability-based search* z użyciem sieci neuronowych
- Szczegóły
  - Fundament: silnik z liniową funkcją oceny uwzględniającą materiał, tablice figura-pole, mobilność, bezpieczeństwo króla gra na poziomie nieco poniżej Kandydata na Mistrza FIDE (Elo = 2200)
  - Heurystyczne decyzje podejmowane głównie przez nauczony system (agresywne heurystyki obcinające zostały celowo pominięte)
  - Algorytmy optymalizacji: *mini-batch stochastic gradient descent with Nesterov's accelerated momentum*, *AdaGrad adaptive subradient method*, *AdaDelta*

Funkcja oceny pozycji

- Realizowana przy pomocy sieci neuronowej
- Reprezentacja cech:
  - Założenie: *gładkie* mapowanie wejścia na wyjście (pozycje posiadające podobną reprezentację cech powinny mieć podobną ocenę)
  - Intuicyjne i naiwne reprezentacje nie zadziałają dobrze, np. *bitboard*:
    - 64 pola, 12 bitów na pole (bo 12 rodzajów figur)
    - 768-wymiarowa przestrzeń
    - Bliskie siebie pozycje (w sensie reprezentacji) niekoniecznie mają podobną ocenę


(a) Similar Position

(b) Original

(c) Different Position

Reprezentacja szachownicy

- Reprezentacja cech cd.:
  - Lepsze rezultaty daje kodowanie pozycji jako listy figur i ich współrzędnych
    - Zbliżone pod względem reprezentacji pozycje będą miały podobną ocenę
  - 32-elementowe wektory niezależne od liczby figur na szachownicy
    - 1-2: króle, 3-4: hetmany, 5-8: skoczki, 9-12: gońce, 13-16: wieże, 17-32: pionki
  - Oprócz współrzędnych każdy element wektora zawiera informację o:
    - Obecności/nieobecności danej figury
    - Broniona/niebroniona
    - Jak daleko może ruszyć się w każdym kierunku (hetman, goniec, wieża)

Ocena reprezentacja cech

- Ewaluacja reprezentacji szachownicy
  - Nadzorowany trening sieci neuronowej, aby zwracała rezultat jak najbardziej zbliżony do wyniku funkcji oceniającej *Stockfish*'a
  - 5 milionów losowo wybranych pozycji z bazy wysokiej jakości partii rozegranych pomiędzy arcymistrzami lub maszynami
  - Różnorodne pozycje pochodzące ze wszystkich faz gry
- Wymagania względem reprezentacji szachownicy:
  - Sieć jest w stanie nauczyć się (zwracać bardzo zbliżone wartości) funkcji oceniającej *Stockfish*'a przy ustalonej reprezentacji
  - Reprezentacja powinna być jak najbardziej ogólna, żeby nie ograniczać jej zdolności do nauki nowych (nieodkrytych obecnie) rzeczy, np. cech pozycji

Reprezentacja szachownicy

- Ostateczna reprezentacja cech (szachownicy)
  - Po czyjej stronie ruch – białych lub czarnych
  - Prawdo do roszady – informacja o każdej z czterech możliwości
  - Rozkład materiału – liczba figur każdego rodzaju
  - Lista figur – obecność, współrzędne, „najsłabsza" figura atakująca i broniąca
    - Konkretne miejsca na liście odpowiadają na stałe konkretnym figurom
  - Mobilność figur „dalekodystansowych" - jak daleko figura może ruszyć się w każdym kierunku
  - Mapa ataku i obrony – każdemu polu przypisywana jest „najsłabsza' figura broniąca i atakująca
- 363 cechy

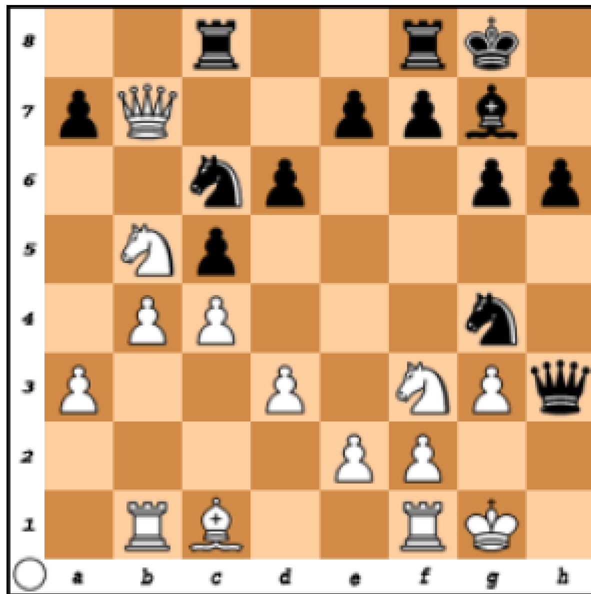
- Lista figur
  - Trzon reprezentacji cech
  - Reprezentacja przy pomocy samej listy daje sensowne rezultaty
  - Pozwala sieci na efektywną naukę:
    - Centralizacji skoczków/gońców
    - Ustawianie wieży na linii pionków
    - Przesuwania pionów do przodu (generalna zasada)
    - Centralizacji króla w końcówkach
    - Mobilności różnych figur w różnych fazach gry
    - Bezpieczeństwo króla bazujące na odległości od atakujących figur i króla przeciwnika

- Po czyjej stronie ruch
  - Pomaga sieci nauczyć się konceptu tempa – prawie w każdej pozycji istnieje ruch, który jest lepszy niż niewykonanie jakiegokolwiek ruchu
    - Zugzwang
  - Istotność cechy zależna od fazy gry, konfiguracji materiału czy struktury pionków
- Rozkład materiału
  - W zdecydowanej większość przypadków jest zbędne (te same informacje można wyciągnąć z listy figur)
  - Przydatne w przypadku promocji pionków, gdy nie ma wystarczającego miejsca na liście figur
  - Ułatwia sieci naukę koncepcji synergii figur, jak również powiązać materiał z innymi cechami
    - Król w centrum w grze środkowej, a król w centrum w końcówce

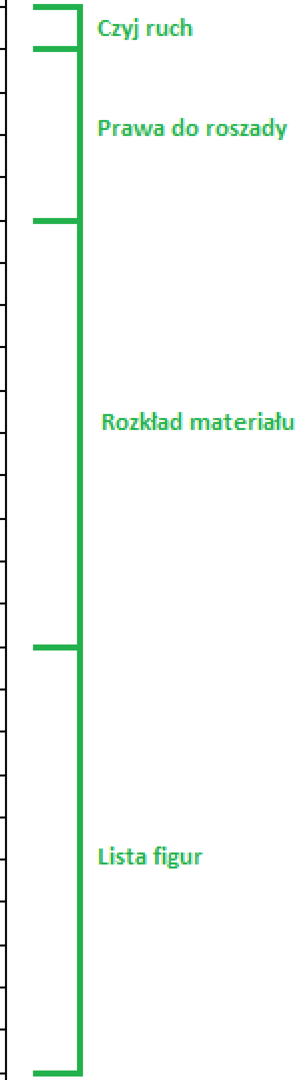
Reprezentacja szachownicy

- Mapa ataku i obrony
  - Umożliwia sieci naukę aspektu kontroli szachownicy
    - Trudne do nauczenia z listy figur (*piece-centric*, a nie *square-centric*)
    - Informacje zawarte w mapie możliwe są do wyciągnięcia z pozostałych cech, mając jednak na uwadze powyższe, uznano za stosowne włączyć mapę do reprezentacji cech

Przykład



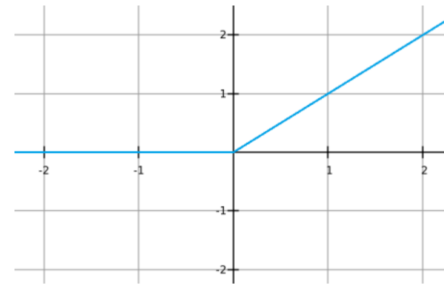
Feature	Value
Side to Move	White
White Long Castle	No
White Short Castle	No
Black Long Castle	No
Black Short Castle	No
White Queens	1
White Rooks	2
White Bishops	1
White Knights	2
White Pawns	7
Black Queens	1
Black Rooks	2
Black Bishops	1
Black Knights	2
Black Pawns	7
White Queen 1 Exists	Yes
White Queen 1 Position	b7
White Rook 1 Exists	Yes
White Rook 1 Position	b1
White Rook 2 Exists	Yes
White Rook 2 Position	f1
White Bishop 1 Exists	Yes
White Bishop 1 Position	c1
White Bishop 2 Exists	No
White Bishop 2 Position	N/A
...	



Architektura sieci

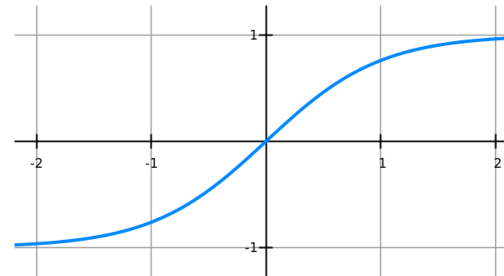
- 4-warstwowa sieć (warstwa wejścia, 2 warstwy ukryte, warstwa wyjścia)
- Funkcja aktywacji
  - Węzły ukryte – ReLU (*Rectified Linear Unit*)

$$f(x) = \max(0, x)$$



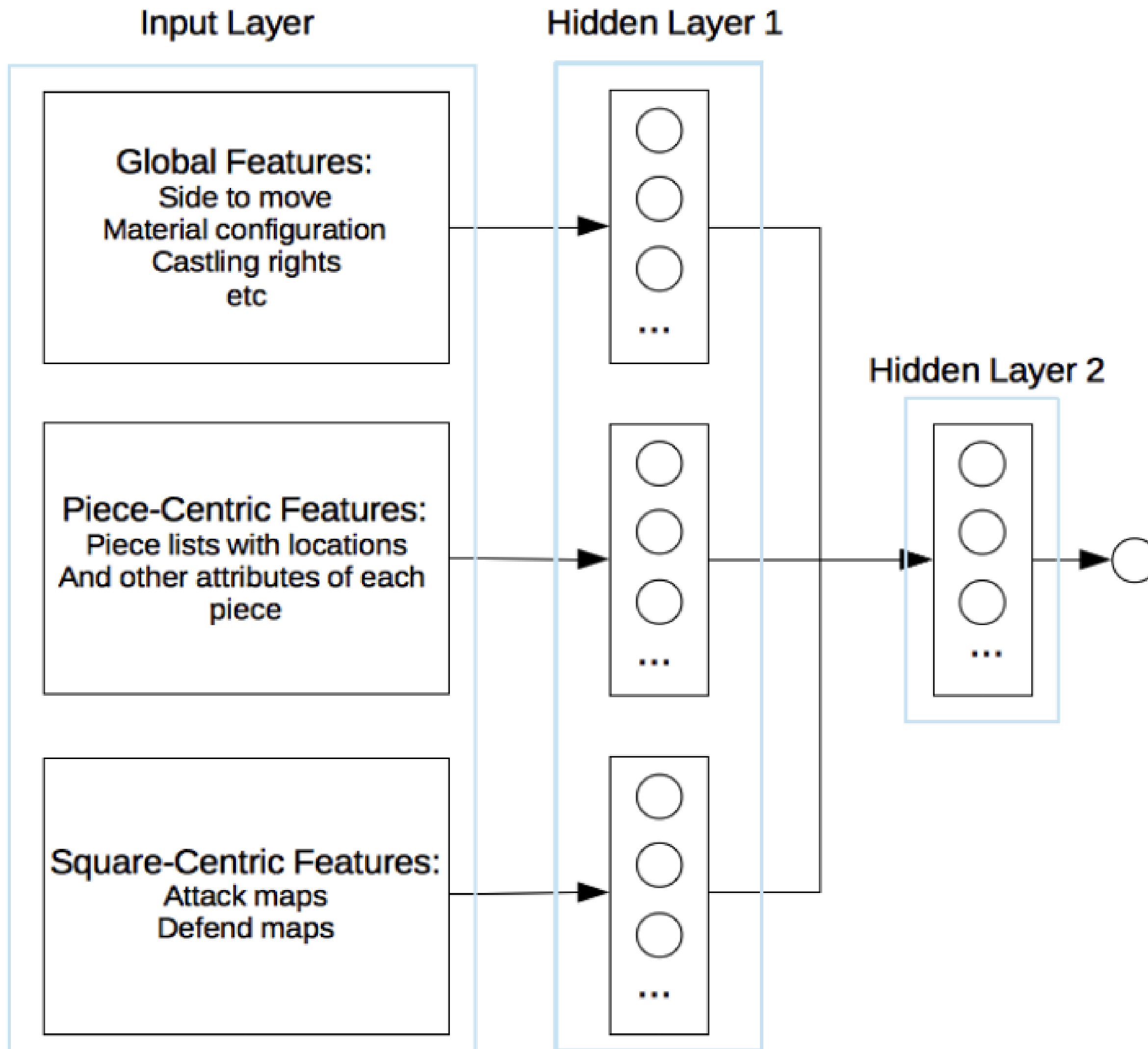
  - Węzeł wyjściowy – tangens hiperboliczny w celu ograniczenia wyjścia do przedziału (-1, 1)

$$f(x) = \tanh x = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



- Zasada: mieszanie cech należących do różnych grup powinno następować w wyższych, bardziej abstrakcyjnych warstwach
- Tworzenie połączeń między cechami różnych grup w niższych warstwach sieci nie przynosi pożytku i zwiększa podatność sieci na przeuczenie
- Wyróżniono 3 grupy cech: *piece-centric*, *square-centric*, *position-centric*
- W pierwszych dwóch warstwach cechy z różnych grup trzymane są oddzielnie
- Dopiero 2 najwyższe warstwy są ze sobą w pełni połączone, żeby wekstrahować wysokopoziomowe zależności pochodzące z cech z różnych grup

Architektura sieci



Generowanie zbioru uczącego

- Założenia (potencjalnie mogą być ze sobą w konflikcie)
  - Poprawny rozkład – pozycje powinny mieć w przybliżeniu ten sam rozkład, jak te pochodzące z realnych gier, np. nie ma sensu trenować sieci pozycją z trzema hetmanami na stronę
  - Różnorodność – system powinien być w stanie grać (poprawnie oceniać) np. w mocno nierównych pozycjach (co prawda rzadko występują w realnych grach, to jednak program napotyka je w wewnętrznych węzłach podczas przeszukiwania cały czas)
  - Odpowiednio liczny zbiór uczący (warunek wyklucza ręczne generowanie pozycji przez człowieka)

Generowanie zbioru uczącego

- Realizacja
  - Wybranie 5 milionów losowych pozycji z bazy danych gier rozgrywanych pomiędzy komputerami
  - Dla każdej z wybranych pozycji, w celu zaburzenia równowagi, wykonany został losowy dozwolony ruch
  - Powstałe pozycje zostały użyte jako punkt startowy do gry z samym sobą
  - Powyższa metoda spełnia postawione wcześniej założenia
  - Otrzymano w ten sposób ok. 175 milionów pozycji treningowych

- TD-Leaf zdolny jest nauczyć sieć z losowych wartości, jednak w przypadku tak złożonego problemu trening zająłby bardzo dużo czasu
- Do zainicjalizowania procesu treningu użyta została bardzo prosta funkcja ewaluacyjna oceniająca na podstawie samego materiału
- W ciągu kilku sekund otrzymano dobry punkt startowy do dalszej nauki
- Użycie silnej funkcji ewaluacyjnej w celu inicjalizacji sieci stoi w sprzeczności z założeniem projektu (ograniczenie dostarczonej z zewnątrz wiedzy do minimum)

Nauka sieci

- TD-Leaf(λ)
- Algorytm
  - Dla każdej iteracji treningu wybierz 256 pozycji ze zbioru treningowego (pozycje z rzeczywistych partii po wykonaniu jednego legalnego losowego ruchu)
  - Dla każdej pozycji zagraj 12 ruchów (dla każdego ruchu zapisz odpowiadające mu wyniki wyszukiwania)
  - Po wykonaniu 12 ruchów patrzymy jak zmieniała się ocena pozycji (szansa, że strona będąca przy ruchu wygra) na przestrzeni 12 ruchów i liczymy błąd dla pozycji startowej sumując zmiany ważone odległością od pozycji startowej
    - Ocena pozycji przybiera wartości z przedziału [-50, 50]
  - Błędy skalowane są przez λ^m
    - λ ma ustaloną wartość 0,7
    - m – odległość od pozycji startowej

Nauka sieci

- Reguła aktualizująca wagi

$$w = w + \alpha \sum_{t=1}^{N-1} \nabla J(x_t, w)\Big[\sum_{j=t}^{N-1} \lambda^{j-t} d_j\Big]$$

w - zestaw wag w modelu

α - szybkość uczenia, ustalona wartość 1

$\nabla J(x_t, w)$ - wektor pochodnych cząstkowych $J(x_t, w)$ względem kolejnych składowych wektora w

$$d_j = \nabla J(x_{j+1}, w) - \nabla J(x_j, w)$$

Przykład

#	Search Score	Score Change	λ	Total Error
1	10	0	0.7^0	0
2	20	10	0.7^1	7
3	20	0	0.7^2	0
4	20	0	0.7^3	0
5	-10	-30	0.7^4	-7.2
6	-10	0	0.7^5	0
7	-10	0	0.7^6	0
8	40	50	0.7^7	4.12
9	40	0	0.7^8	0
10	40	0	0.7^9	0
11	40	0	0.7^{10}	0
12	40	0	0.7^{11}	0

- Każda zmiana oceny pozycji przyczynia się do błędu całkowitego dla pozycji startowej
- Całkowity błąd po 12 ruchach:

$$10 \cdot 0.7^1 - 30 \cdot 0.7^4 + 50 \cdot 0.7^7 = 3.92$$

Nauka sieci

- Po uzyskaniu wszystkich błędów dla 256 pozycji, użyty został algorytm propagacji wstecznej w celu obliczenia gradientu funkcji kosztu w odniesieniu do wag sieci neuronowej
- Następnie gradienty są użyte do trenowania sieci z użyciem metody *stochastic gradient descent* (*AdaDelta*, *AdaGrad*, *SGD with momentum*, *SGD with Nesterov's accelerated gradient*)
- *AdaDelta* – różnice względem pozostałych (oprócz *AdaGrad*)
  - Tempa uczenia zmieniane są po każdej iteracji
  - Dla każdej wagi zachowuje oddzielne tempo uczenia
    - Dzięki temu wagi neuronów, które są rzadko aktywowane mogą posiadać wysokie tempo nauki

Wyniki

- Testowanie: Strategic Test Suite
  - 1500 pozycji podzielonych tematycznie na 15 grup
  - Pozycje bez jednoznacznych rozwiązań taktycznych
  - Ukierunkowane na ocenę strategicznego „czucia” pozycji
  - Najlepszy ruch ⟶ 10, pozostałe ⟶ [0, 10]
  - Uczenie samym materiałem ⟶ 6000/15000
  - Docelowo ⟶ 9500/15000

1. Undermining
2. Open Files and Diagonals
3. Knight Outposts
4. Square Vacancy
5. Bishop vs Knight
6. Re-Capturing
7. Offer of Simplification
8. Advancement of f/g/h pawns
9. Advancement of a/b/c Pawns
10. Simplification
11. Activity of the King
12. Center Control
13. Pawn Play in the Center
14. Queens and Rooks to the 7th Rank

Wyniki

Engine	Approx. Rating [3]	Average Nodes Searched	STS Score
Giraffe (1s)	2400	258570	9641
Giraffe (0.5s)	2400	119843	9211
Giraffe (0.1s)	2400	24134	8526
Stockfish 5	3387	108540	10505
Senpai 1.0	3096	86711	9414
Texel 1.04	2995	119455	8494
Arasan 17.5	2847	79442	7961
Scorpio 2.7.6	2821	139143	8795
Crafty 24.0	2801	296918	8541
GNU Chess 6 / Fruit 2.1	2685	58552	8307
Sungorus 1.4	2309	145069	7729

- Pozycyjne rozumienie porównywalne z „wyśrubowanymi” funkcjami oceny najsilniejszych silników szachowych

```
function minimax(position, depth)
{
    if depth == 0:
        return evaluate(position)

    bestScore = -∞

    for each possible move mv:
        subScore = -minimax(position.apply(mv), depth - 1)
        if subScore > bestScore:
            bestScore = subScore

    return bestScore
}
```

Przeszukiwanie probabilistyczne

- Teoretyczny wariant główny (PV – *prinicipal variation*) – sekwencja teoretycznie optymalnych ruchów, które prowadzą od rozważanej pozycji do końca gry (mat, pat, powtórzenie pozycji, itd.)
- Każda pozycja ma co najmniej jeden teoretyczny PV
- Przyjmijmy założenie, że każda pozycja ma dokładnie jeden PV (rzeczywistej liczby nie da się stwierdzić bez przeszukania całego drzewa, co jest niewykonalne)
- Zadanie: „Przeszukaj wszystkie węzły, których prawdopodobieństwo należenia do teoretycznego PV jest większe niż 0.0000001"
- $P(child_1|parent) + P(child_2|parent) + ... + P(child_n|parent) = 1$

Przeszukiwanie probabilistyczne

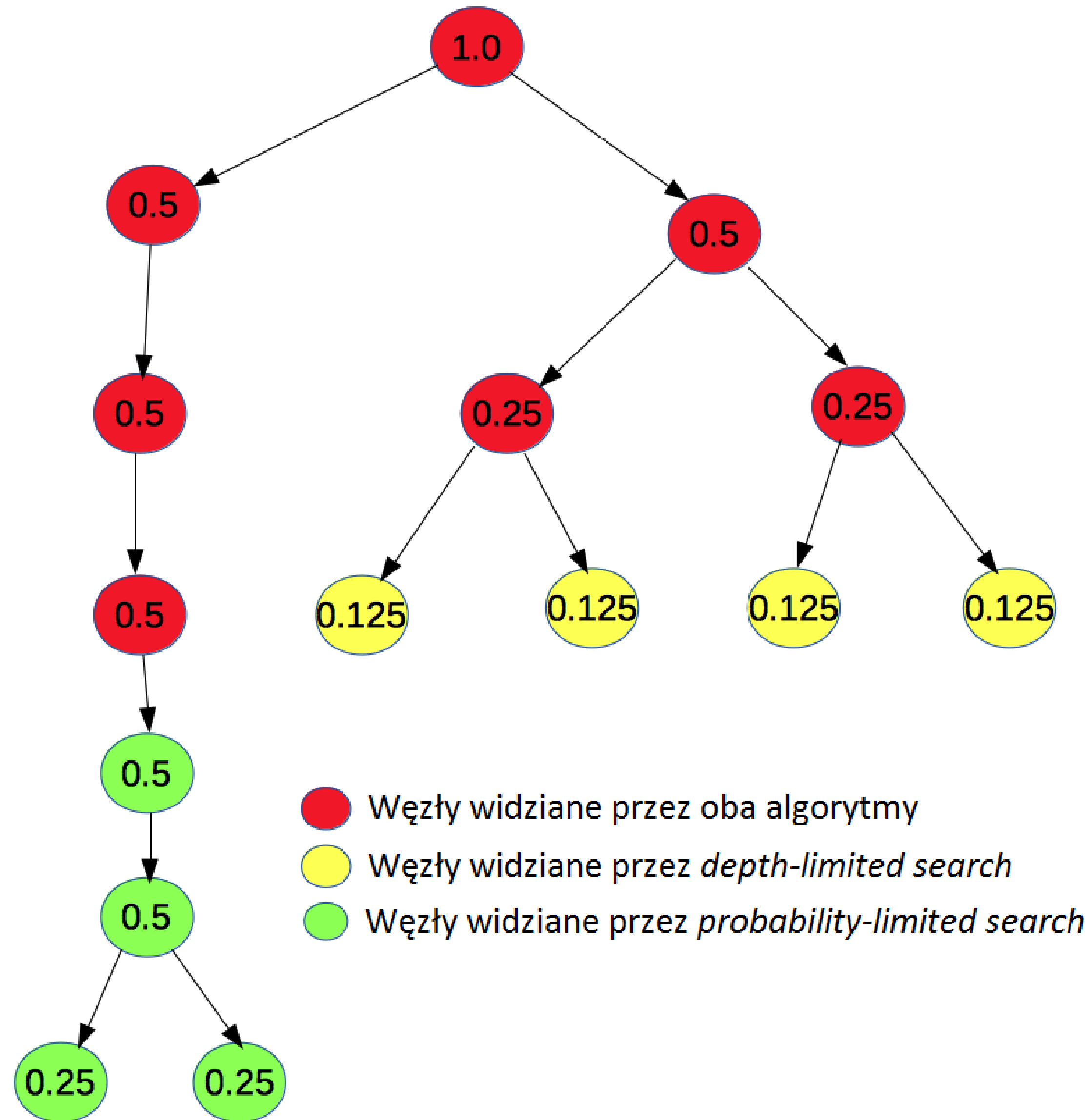
- Załóżmy brak wiedzy na temat pozycji oraz ruchów i przyjmijmy, że prawdopodobieństwo rodzica rozkłada się równo na potomków

```
function minimax(position, currentProbability)
{
    if currentProbability < ProbabilityThreshold:
        return evaluate(position)

    bestScore = -∞

    numMoves = number of legal moves from this position

    for each possible move mv:
        subScore = -minimax(position.apply(mv), currentProbability / numMoves)
        if subScore > bestScore:
            bestScore = subScore

    return bestScore
}
```

Porównanie przeszukiwań

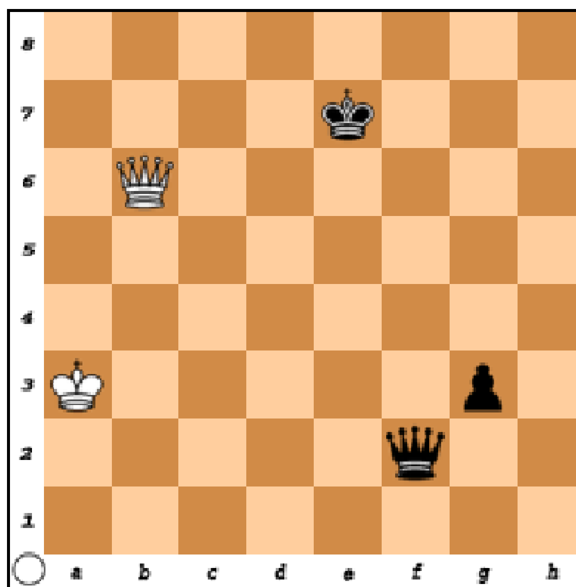
- W większości przypadków oba sposoby przeszukiwań przeanalizują w przybliżeniu podobne poddrzewa, ponieważ:
  - *Branching factor* jest w przybliżeniu stały w całym procesie wyszukiwania
  - Potomkowie dziedziczą prawdopodobieństwo rodzica po równo
  - Przeszukiwanie zakończy się w obu przypadkach w przybliżeniu na tej samej głębokości
- Wyjątek – pozycje z poddrzewami z różnymi *branching factor*, np.
  - Ruch szachujący
  - Gdy gałąź zawiera wariant z wymianą figur o wysokiej mobilności (np. hetmany)

Porównanie przeszukiwań



Porównanie przeszukiwań

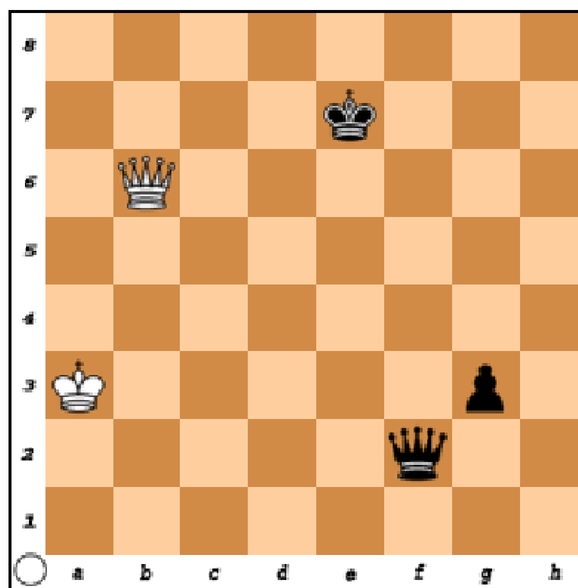
- Przeszukiwanie *depth-limited* spędza dużo więcej czasu na poddrzewach z wysokim *branching factor*



- *Depth-limited* spędzi prawie cały czas dla wariantów, gdzie hetmany pozostają wciąż na szachownicy
- Gałąź z wymianą hetmanów mogłaby być przeszukana dużo głębiej kosztem małego kawałka czasu użytego dla gałęzi bez wymiany

Porównanie przeszukiwań

- Przeszukiwanie *probability-limited* spędza w przybliżeniu tyle samo czasu na każdej gałęzi i zejdzie dużo głębiej w wariancie z wymianą hetmanów



- Z punktu widzenia prawdopodobieństwa jest to bardziej korzystne
  - Prawdopodobieństwo, że dany węzeł na głębokości 5 gałęzi bez wymiany należy do teoretycznego PV jest dużo mniejsze niż dla węzła na głębokości 7 w gałęzi z wymianą (*branching factor* jest tu znacznie niższy)

Porównanie przeszukiwań

- Naiwna implementacja przeszukiwania *probability-limited* jest nieznacznie silniejsza (26 +/-12 punktów ELO) od naiwnej implementacji *depth-limited*
- Brak porównań między zaawansowanymi implementacjami obu algorytmów
- Większość zaawansowanych optymalizacji użytych w przeszukiwaniu *depth-limited* można również zastosować w *probability-limited*
- Badania pokazały, że w *depth-limited* wydłużanie przeszukiwania w przypadku ruchów szachujących i sytuacji z pojedynczą możliwą odpowiedzią przynosi korzyści
  - *depth-limited* zachowuje się wtedy podobnie do *probability-limited* (*probability-limited* staje się uogólnieniem powyższych technik)
- *Giraffe* jest pierwszym silnikiem implementującym przeszukiwanie *probability-limited*

Szacowanie prawdopodobieństwa

- Do tej pory zakładaliśmy, że $P(child_i|parent) = \frac{1}{n}$
- $P(child_i \& parent) = P(child_i|parent)P(parent)$
- Cel: nauczyć się, aby dla danej pozycji wejściowej i ruchu prowadzącego do potomka zwracała $P(child_i|parent)$

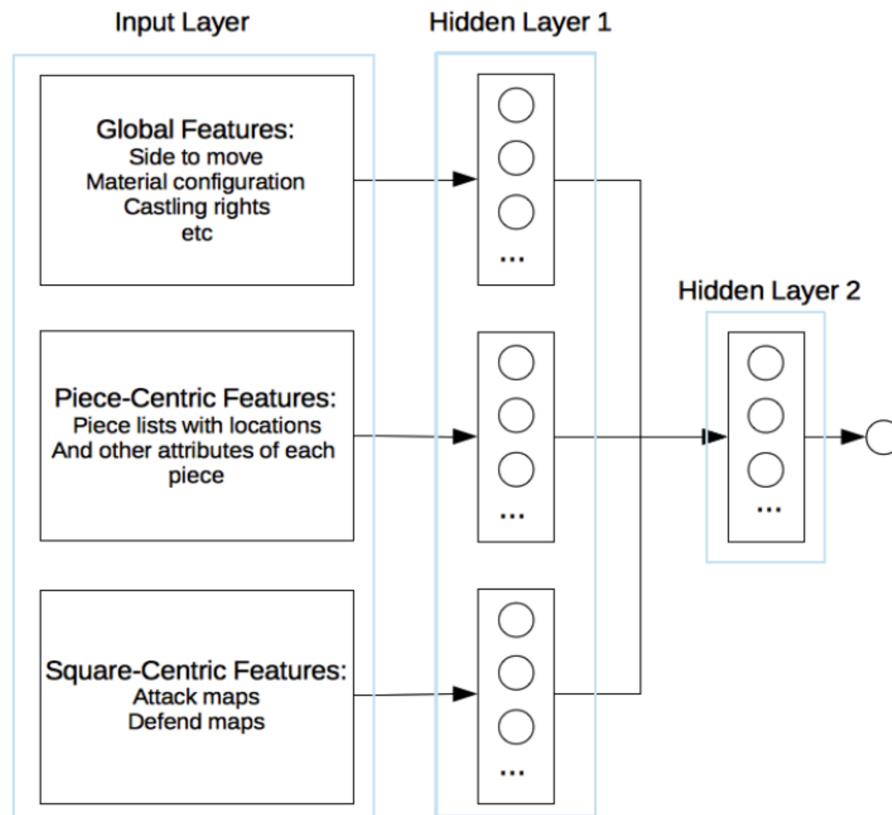
Reprezentacja cech

- Część reprezentująca pozycję rodzica – taka sama jak w przypadku reprezentacji cech dla oceny pozycji
- Część reprezentująca ruch:
  - Rodzaj figury
  - Pole „z”
  - Pole „do”
  - Rodzaj promocji (w przypadku, gdy pionek osiąga ostatnią linię)
  - Ranking ruchu – jak dany ruch ma się względem pozostałych
- Prawdopodobieństwo, że dany ruch jest najlepszy zależy od pozostałych możliwych ruchów dla danej pozycji
  - Podział prawdopodobieństwa między wyróżniające się ruchy
- Jajko czy kura? Jak użyć rankingu ruchów do obliczenia prawdopodobieństwa, gdy chcemy użyć prawdopodobieństwo do uzyskania rankingu?

- Rozwiązanie – dwa przebiegi
  - W pierwszym przebiegu wszystkie ruchy oceniane są tak jakby były najlepsze (ta sama wartość rankingu dla każdego ruchu)
  - Następnie na podstawie otrzymanych prawdopodobieństw tworzony jest ranking ruchów
  - W drugim przebiegu ruchy oceniane są ponownie z uwzględnieniem rankingu

Architektura sieci

- Bardzo zbliżona do sieci ewaluującej pozycje
- Różnica: logistyczna funkcja aktywacji neuronu wyjściowego – zwracane wartości z przedziału (0, 1)



Generowanie pozycji treningowych

- W przypadku trenowania sieci ewaluującej pozycje, bardzo mocno ograniczona została liczba sytuacji, w których jedna ze stron ma bezwzględną przewagę, ponieważ pozycje takie rzadko występują w rzeczywistych partiach
- Pozycje takie pojawiają się natomiast często w wewnętrznych węzłach drzewa gry
- Jako że sieć estymująca prawdopodobieństwo działa na węzłach wewnętrznych, zbiór treningowy powinien mieć dostosowany do tego rozkład pozycji
- Generowanie zbioru treningowego:
    - Stwórz drzewa gry, którego korzeniem jest pozycja ze zbioru uczącego dla sieci ewaluującej pozycje (głębokość drzewa ograniczona czasem na tworzenie)
    - Losowo wybierz pozycje z węzłów wewnętrznych drzewa
- Zbiór uczący składa się z 5 milionów pozycji

Uczenie sieci

- Oznacz każdą pozycję zbioru treningowego najlepszym znalezionym ruchem (przeszukiwanie drzewa ograniczone czasem)
- Wygeneruj zbiór uczący dla metody *Gradient descent* łącząc każdą pozycję ze wszystkimi dozwolonymi dla niej ruchami (każdy ruch oznacz binarnie czy jest/nie jest najlepszym ruchem)
- Wykonaj *Stochastic gradient descent* (*AdaDelta*) do zaktualizowania wag sieci

Wyniki

Predicted Rank of Actual Best Move	Frequency (%)	Cumulative Frequency (%)
0	45.73	45.73
1	15.95	61.68
2	7.91	69.59
3	5.18	74.77
4	3.71	78.48
5	3.00	81.48
6	2.38	83.86
7	1.77	85.63
8	1.86	87.49
9	1.35	88.84
10	1.54	90.38
Below 10th	9.62	100.00

- Estymator probabilistyczny z użyciem sieci neuronowej zwiększa siłę gry o 48 +/- 12 punktów ELO (57% do 43% przy 3000 partii)

- Nauczony system może grać na podobnym poziomie co system z bogatą wiedzą ekspercką
- Jest mocno prawdopodobne, że zaprezentowane podejście może być łatwo zaadaptowane do innych gier planszowych o sumie zerowej osiągając siłę gry na poziomie najmocniejszych programów
- Wyszukiwanie *probability-based* jest co najmniej porównywalne, a może i mocniejsze niż klasyczne *depth-based*
- Relatywnie lepsze wyczucie strategiczne (charakterystyczne dla silnego gracza ludzkiego) niż taktyczne
- Siła gry na poziomie Mistrza Międzynarodowego (ok. 2.2% zawodników z oficjalnym rankingiem) przy użyciu PCta

Bibliografia

- Matthew Lai *Giraffe: Using Deep Reinforcement Learing to Play Chess*. M.Sc. Thesis, Imperial College London, 2015.
- Hsu Feng-Hsiung *Behind Deep Blue: Building the computer that defeated the world chess champion*, Princeton University Press, 2002.
- Sacha Droste, Johannes Fürnkranz *Learning of Piece Values for Chess Variants*, Technische Universität Darmstadt, 2008.
- Jonathan Baxter, Andrew Tridgell, Lex Weaver *Learning To Play Chess Using Temporal Differences*, Australian National University, 2001.
- Daniel Osman, Jacek Mańdziuk *Comparison of TDLeaf and TD learning in game playing domain*, Lecture Notes in Computer Science, vol. 3316, 549-554, Springer-Verlag
- Paweł Stawarz *Zastosowanie algorytmu uczenia się ze wzmocnieniem na przykładzie konkretnego systemu TD-GAMMON*, Politechnika Wrocławska, 2007.

Dziękuję za uwagę