Podsumowanie Rozwoju i Ewaluacji Projektu ChessAI

Data: 24.05.2024
Autor: Tomasz Madeja

---

1. Wprowadzenie

Niniejszy dokument przedstawia szczegółowe podsumowanie ewolucji projektu **ChessAI**, silnika szachowego opartego na uczeniu maszynowym. Celem raportu jest analiza porównawcza dwóch kluczowych etapów rozwoju projektu: początkowej wersji implementującej podstawowe założenia architektury AlphaZero oraz wersji znacznie rozbudowanej, która wprowadza zaawansowane techniki optymalizacyjne, hybrydowe podejście do generowania ruchów oraz w pełni zautomatyzowany i zrównoleglony potok treningowy.

Dokument analizuje fundamentalne zmiany w architekturze oprogramowania, algorytmach sztucznej inteligencji oraz metodologii treningu. W końcowej części przedstawiono wyniki testów wydajnościowych (benchmarków) nowej wersji silnika w konfrontacji z renomowanym silnikiem Stockfish, wraz z wnikliwą analizą przyczynową uzyskanych rezultatów.

2. Streszczenie Wyników i Kluczowych Zmian

Projekt ChessAI przeszedł fundamentalną transformację, ewoluując z akademickiego dowodu słuszności koncepcji (Proof-of-Concept) w zaawansowaną, modułową platformę do budowy i testowania silników szachowych.

Kluczowe wprowadzone modyfikacje to:

1. **Transformacja Architektoniczna:** Przejście od monolitycznego modelu "czystego" uczenia przez wzmacnianie (Reinforcement Learning) do **systemu hybrydowego**, który integruje sieć neuronową z klasyczną wiedzą szachową (księga otwarć, bazy końcówek).
2. **Optymalizacja Rdzenia AI:** Przepisanie kluczowych algorytmów, w tym implementacja **równoległego przeszukiwania MCTS z batchowaniem zapytań** i mechanizmem *Virtual Loss*, co drastycznie zwiększyło wydajność i przepustowość analizy.
3. **Profesjonalizacja Potoku Treningowego:** Wprowadzenie dwuetapowego procesu uczenia:
    - **Etap 1: Trening Nadzorowany (Supervised Learning)** na ogromnym zbiorze partii ludzkich w celu "wstępnego rozgrzania" sieci.
    - **Etap 2: Zrównoleglony Trening przez Wzmacnianie (RL)** z wykorzystaniem wielu rdzeni procesora, co skróciło czas generowania danych treningowych z dni do godzin.

4. **Wdrożenie Rygorystycznej Ewaluacji:** Stworzenie dedykowanego modułu do benchmarkingu, który mierzy nie tylko wyniki meczów (Wygrana/Przegrana/Remis), ale również zaawansowane metryki jakości gry, takie jak **ACPL (Average Centipawn Loss)** i **Top-1/Top-3 Accuracy**, używając silnika Stockfish jako wyroczni.

Należy jednak podkreślić, że **wstępne wyniki benchmarków są znacząco poniżej oczekiwań**. Przyczyną tego stanu rzeczy jest **krytyczny błąd w implementacji modułu move_mapping.py**, zidentyfikowany na końcowym etapie prac, tuż przed przeprowadzeniem testów. Błąd ten powoduje nieprawidłowe mapowanie pomiędzy akcjami przewidywanymi przez sieć neuronową a faktycznymi ruchami wykonywanymi na szachownicy, co w praktyce uniemożliwia AI skuteczne wykorzystanie swojej "inteligencji". Mimo to, postęp inżynieryjny i architektoniczny stanowi solidny fundament pod dalszy, skokowy rozwój projektu po naprawieniu wspomnianej usterki.

---

3. Ewolucja Architektury: Od "Czystego" RL do Modelu Hybrydowego

Najważniejszą zmianą koncepcyjną w projekcie jest odejście od dogmatycznego podejścia, w którym sieć neuronowa jest jedynym źródłem wiedzy, na rzecz pragmatycznego modelu hybrydowego. Stara wersja opierała się wyłącznie na MCTS kierowanym przez sieć w każdej fazie gry. Nowa wersja wprowadza hierarchiczny proces decyzyjny, który znacząco podnosi jakość gry i efektywność obliczeniową.

Nowe komponenty wiedzy:

- **Księga Otworć (opening_book.json):** Wprowadzono plik JSON zawierający popularne i teoretycznie solidne linie debiutowe. W początkowej fazie partii AI, zamiast przeprowadzać kosztowne obliczenia, wybiera jeden z rekomendowanych ruchów, zapewniając sobie dobrą pozycję bez ryzyka popełnienia wczesnego błędu.
- **Bazy Końcówek Syzygy (tablebases/):** Zintegrowano obsługę 7-figurowych baz końcówek. Gdy liczba figur na szachownicy spadnie poniżej tego progu, AI przełącza się w tryb "perfekcyjny", odpytując bazy danych, które zawierają optymalne ruchy dla wszystkich możliwych pozycji. Eliminuje to błędy w kluczowej, końcowej fazie gry.
- **Moduł knowledge.py:** Stworzono dedykowany moduł, który hermetyzuje logikę dostępu do powyższych źródeł wiedzy, czyniąc kod czystszym i bardziej modułowym.

Nowa hierarchia decyzyjna:

Proces wyboru ruchu w nowej wersji przebiega następująco:

1. **Czy aktualna pozycja jest w księdze otwarć?** Jeśli tak, wykonaj ruch z księgi.
2. **Czy pozycja kwalifikuje się do użycia baz końcówek?** Jeśli tak, wykonaj perfekcyjny ruch z bazy.
3. **W przeciwnym wypadku, uruchom przeszukiwanie MCTS** z użyciem sieci neuronowej, aby znaleźć najlepszy ruch w grze środkowej lub złożonej końcówce.

Takie podejście drastycznie poprawia jakość gry w skrajnych fazach partii i pozwala zarezerwować moc obliczeniową na najbardziej skomplikowane pozycje gry środkowej.

---

4. Optymalizacje Rdzenia AI i Procesu Treningowego

Rozbudowa projektu objęła fundamentalne usprawnienia w samym sercu algorytmów AI oraz w procesie ich trenowania.

4.1. Usprawnienia Algorytmiczne i Architektoniczne

- **Równoległy MCTS z Batchowaniem:** Algorytm Monte Carlo Tree Search został przepisany od podstaw. Wersja pierwotna wykonywała sekwencyjnie pętlę selekcji, ekspansji, symulacji i propagacji, generując jedno zapytanie do modelu na każdą symulację. Nowa implementacja jest **zrównoleglona i zbatchowana**:
    - W jednej iteracji algorytm zbiera zdefiniowaną liczbę (mcts_batch_size) nierozwiniętych liści drzewa.
    - Wszystkie te pozycje są przetwarzane w **jednym, dużym batchu** przez sieć neuronową, co maksymalizuje wykorzystanie mocy obliczeniowej GPU.
    - Wprowadzono mechanizm **"Virtual Loss"**, który tymczasowo obniża ocenę węzłów na ścieżkach aktualnie eksplorowanych, zniechęcając inne "wątki" w ramach jednego batcha do podążania tą samą drogą. Zwiększa to różnorodność przeszukiwania.

- **Udoskonalona Architektura Sieci Neuronowej (model.py):**
    - Stary model używał prostego spłaszczenia (Flatten) po bloku rezydualnym. Nowa architektura, wzorowana ściślej na AlphaZero, posiada **oddzielne głowice konwolucyjne** dla polityki (wybór ruchu) i wartości (ocena pozycji), co pozwala na ekstrakcję bardziej wyspecjalizowanych cech.
    - Do każdej warstwy konwolucyjnej dodano **normalizację batchową (BatchNormalization)**, co stabilizuje proces treningu, przyspiesza konwergencję i poprawia ogólną wydajność modelu.

- **Standaryzacja Reprezentacji Ruchów (move_mapping.py):**
    - Autorski, nieefektywny system mapowania ruchów (ponad 8000 akcji) został zastąpiony **standardową 73-płaszczyznową reprezentacją z AlphaZero (4672 akcje)**. Uczyniło to projekt zgodnym z literaturą naukową i uprościło architekturę sieci.

4.2. Rewolucja w Potoku Treningowym

Całkowicie zmieniono podejście do uczenia modelu.

- **Etap 1: Wstępny Trening Nadzorowany:**
    - Dodano skrypty (prepare_data.py, train_supervised.py) do przetwarzania milionów partii szachowych z publicznych baz danych (Lichess).

    - Wprowadzono mechanizm **"chunkingu"**, który dzieli ogromne zbiory danych na mniejsze, zarządzalne kawałki, co pozwala na ich przetwarzanie bez wyczerpania pamięci RAM.
    - Model jest najpierw trenowany w trybie nadzorowanym, ucząc się naśladować ruchy silnych ludzkich graczy. Daje mu to solidne podstawy i intuicję szachową, zanim przejdzie do znacznie wolniejszego uczenia przez wzmacnianie.

- **Etap 2: Zrównoleglony Trening przez Wzmacnianie (train_alphazero_style.py):**
    - Cała pętla treningowa (generowanie gier, trening, ewaluacja) została **zrównoleglona przy użyciu modułu multiprocessing**. Pozwala to na jednoczesne wykorzystanie wielu rdzeni CPU do generowania partii i prowadzenia meczów ewaluacyjnych, co skraca cykl treningowy o rząd wielkości.

---

5. Ewaluacja i Analiza Wyników Benchmarkowych

Aby obiektywnie zmierzyć siłę gry i jakość podejmowanych decyzji przez nowy silnik, stworzono zaawansowany skrypt benchmark.py.

5.1. Metodologia Testów

- **Przeciwnik:** Modyfikowalna wersja silnika Stockfish.
- **Wyrocznia (Oracle):** Druga, znacznie silniejsza instancja Stockfisha, używana do oceny ruchów wykonanych przez ChessAI.
- **Metryki:**
    - **Wynik meczu:** Procent wygranych, przegranych i remisów.
    - **Accuracy (Top-1/Top-3):** Odsetek ruchów ChessAI, które były zgodne z najlepszym (lub jednym z trzech najlepszych) ruchem wskazanym przez wyrocznię.
    - **Average Centipawn Loss (ACPL):** Średnia utrata oceny pozycji (w centypionach) po wykonaniu ruchu przez ChessAI w porównaniu do najlepszego możliwego ruchu. Niższa wartość oznacza lepszą grę.

- **Warunki:** Mecze rozgrywano ze zdefiniowanego zestawu 5 popularnych otwarć, aby zapewnić powtarzalność i zróżnicowanie testów.

5.2. Prezentacja Wyników

Parametry Testu	Liczba Gier	Wynik (W-L-D)	Win Rate	ACPL	Top-1 Acc.	Top-3 Acc.
vs. Stockfish 2.3.1 (głębokość 2)	10	0 - 10 - 0	0.0%	98.63	30.78%	54.75%
vs. Stockfish 17.1 (głęb. 1, 1 węzeł)	10	1 - 7 - 2	10.0%	*brak*	*brak*	*brak*
vs. Stockfish 17.1(głębokość 1)	50	0 - 48 - 2	0.0%	*brak*	*brak*	*brak*

5.3. Interpretacja i Identyfikacja Problemu

Wyniki są jednoznacznie słabe. Nawet przeciwko drastycznie osłabionemu Stockfishowi, który patrzy zaledwie jeden lub dwa ruchy w przód, ChessAI notuje niemal wyłącznie porażki. Wysoka wartość ACPL (98.63) wskazuje, że AI regularnie popełnia poważne błędy pozycyjne.

Główna przyczyna tak słabych wyników została zidentyfikowana i jest nią krytyczny błąd w implementacji modułu ai/move_mapping.py.

Podczas gdy architektura sieci i schemat mapowania zostały zaktualizowane do standardu 73-płaszczyznowego, logika mapująca indeksy z wektora wyjściowego sieci neuronowej na konkretne obiekty chess.Move została zaimplementowana nieprawidłowo. W praktyce oznacza to, że:

1. **Sieć neuronowa uczy się poprawnie** – na podstawie tysięcy partii kojarzy daną pozycję z optymalnym rozkładem prawdopodobieństw na wektorze wyjściowym o długości 4672.
2. Jednak w fazie wykonawczej, gdy MCTS prosi sieć o ocenę, a ta zwraca wysokie prawdopodobieństwo dla indeksu X (który *powinien* odpowiadać np. ruchowi e2e4), **system błędnie tłumaczy ten indeks na zupełnie inny, często bezsensowny ruch Y**.

W efekcie cała "inteligencja" i "wiedza" zgromadzona przez sieć neuronową jest bezużyteczna, ponieważ jej rekomendacje są źle interpretowane. Silnik gra niemal losowo, a nieliczne zwycięstwa lub remisy wynikają prawdopodobnie z wykorzystania księgi otwarć/baz końcówek lub rażących błędów przeciwnika. Z uwagi na późne wykrycie błędu, nie było możliwości przeprowadzenia ponownego, czasochłonnego treningu i ewaluacji przed sporządzeniem niniejszego raportu.

---

6. Wnioski i Dalsze Kroki

Pomimo niezadowalających wyników benchmarków, rozbudowa projektu ChessAI była ogromnym sukcesem z perspektywy inżynierii oprogramowania i architektury systemów AI. Stworzono solidną, wydajną i skalowalną platformę, która jest gotowa do generowania silnych modeli po naprawieniu zidentyfikowanego błędu.

Główne wnioski:

- Wprowadzenie **modelu hybrydowego** oraz **zaawansowanych optymalizacji algorytmicznych** (batching MCTS) jest kluczowe dla osiągnięcia konkurencyjnej siły gry.
- **Dwuetapowy, zrównoleglony potok treningowy** jest skutecznym i efektywnym czasowo podejściem do uczenia złożonych modeli szachowych.
- Krytyczne znaczenie ma **rygorystyczne testowanie jednostkowe i integracyjne** każdego komponentu, zwłaszcza modułów odpowiedzialnych za transformację danych (jak move_mapping.py).

Planowane dalsze kroki:

1. **Priorytet nr 1: Naprawa błędu w move_mapping.py** i weryfikacja poprawności mapowania we wszystkich kierunkach (ruch -> indeks oraz indeks -> ruch).
2. **Przeprowadzenie pełnego cyklu treningowego od nowa:**
    - Ponowne uruchomienie skryptu train_supervised.py na zbiorze danych Lichess w celu stworzenia nowej, solidnej bazy.
    - Uruchomienie długiej sesji treningowej train_alphazero_style.py w celu dostrojenia modelu przez samodoskonalenie.
3. **Ponowna ewaluacja benchmarkowa** w celu uzyskania miarodajnych wyników siły gry nowego, poprawnie wytrenowanego modelu.
4. Dalsza rozbudowa księgi otwarć i eksperymenty z hiperparametrami treningu (wielkość sieci, tempo uczenia, parametry MCTS).