IMPLEMENTACJA ALGORYTMU SAMOUCZĄCEGO SIĘ GIER W ZASTOSOWANIU DO ULTIMATE TIC-TAC-TOE

Olga Krużyńska

Aleksandr Tsekhanovskii

Promotor:
dr hab. Inż. Wojciech Kotłowski, prof. PP

CEL

Stworzenie systemu samouczącego się, na podstawie algorytmu Descent w grze Ultimate Tic Tac Toe i porównanie tego systemu z programem SaltZero (koncepcja AlphaZero), a także z klasycznym podejściem bez sieci neuronowej – Negamax α-β z interative deepening.

MOTYWACJA

- Projekt, który da dobre podstawy do zaglębienia się w sztuczną inteligencję i da wiedzę o samouczeniu,
- fascynacja osiągnięciami AlphaZero (uczenie przez rozgrywki samego z sobą danych zewnętrznych i heurystyk, stworzonych przez człowieka,
- czemu wybraliśmy algorytm Descent a nie MCTS, skoro to AlphaZero nas interesowało:
  - Istnieją warianty AlphaZero do wybranej przez nas gry UTTT (SaltZero).
  - w artykułach naukowych znaleźliśmy mało znany algorytm Descent i stwierdziliśmy, że wykorzystamy coś nowego niż MCTS.
  - nowe rozwiązania - nie było jeszcze zastosowania Descent do UTTT i przetestowaliśmy jak to podejście wypada

- Idea
- oznaczenia
- kroki działania

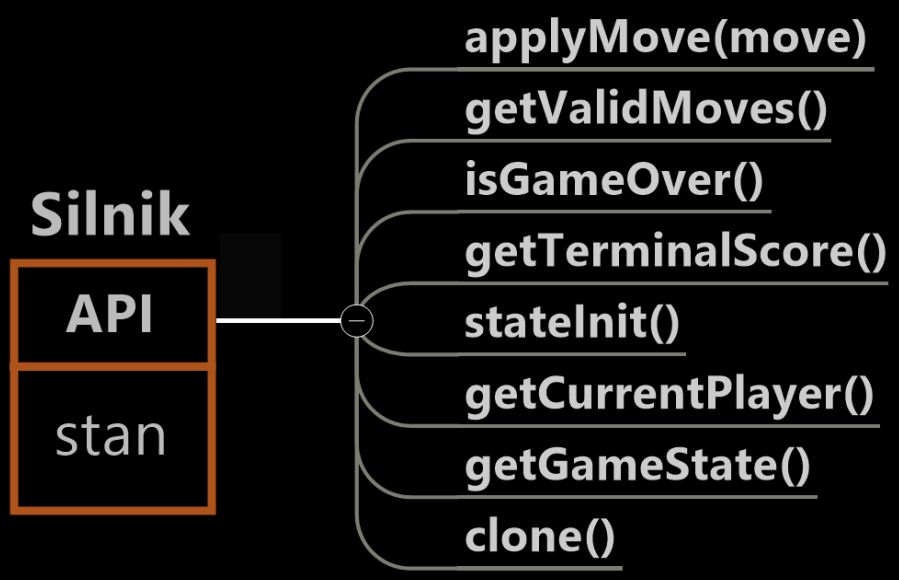
PSEUDOKOD ALGORYTMU DESCENT

```
Function descent(s, S, T, fθ, ft, τ)
    t = time()
    while (time() - t) < τ do
        descent_iteration(s, S, T, fθ, ft)
    return S, T
```

```
Function descent_iteration(s, S, T, fθ, ft)
    if terminal(s) then
        S ← S ∪ {s}
        v(s) ← ft(s)
    else
        if s ∉ S then
            S ← S ∪ {s}
            foreach a ∈ actions(s) do
                if terminal(a(s)) then
                    S ← S ∪ {a(s)}
                    v'(s, a) ← ft(a(s))
                    v(a(s)) ← v'(s, a)
                else
                    v'(s, a) ← fθ(a(s))
            ab ← best_action(s)
            v'(s, ab) ← descent_iteration(ab(s), S, T, fθ, ft)
            ab ← best_action(s)
            v(s) ← v'(s, ab)
    return v(s)
```

```
Function best_action(s)
    if first_player(s) then
        return arg_max_{a ∈ actions(s)} v'(s, a)
    else
        return arg_min_{a ∈ actions(s)} v'(s, a)
```

SYSTEM SAMOUCZĄCY SIĘ - ETAPY TWORZENIA



Silnik:

- odpowiada za stan gry
- zawiera funkcje obsługujące ten stan (zgodnie z logiką gry)

Implementacja silnika wymagania:

- minimalny rozmiar pamięci
- wysoka wydajność oraz niezawodność

Wykonane w języku C++ poprzez:

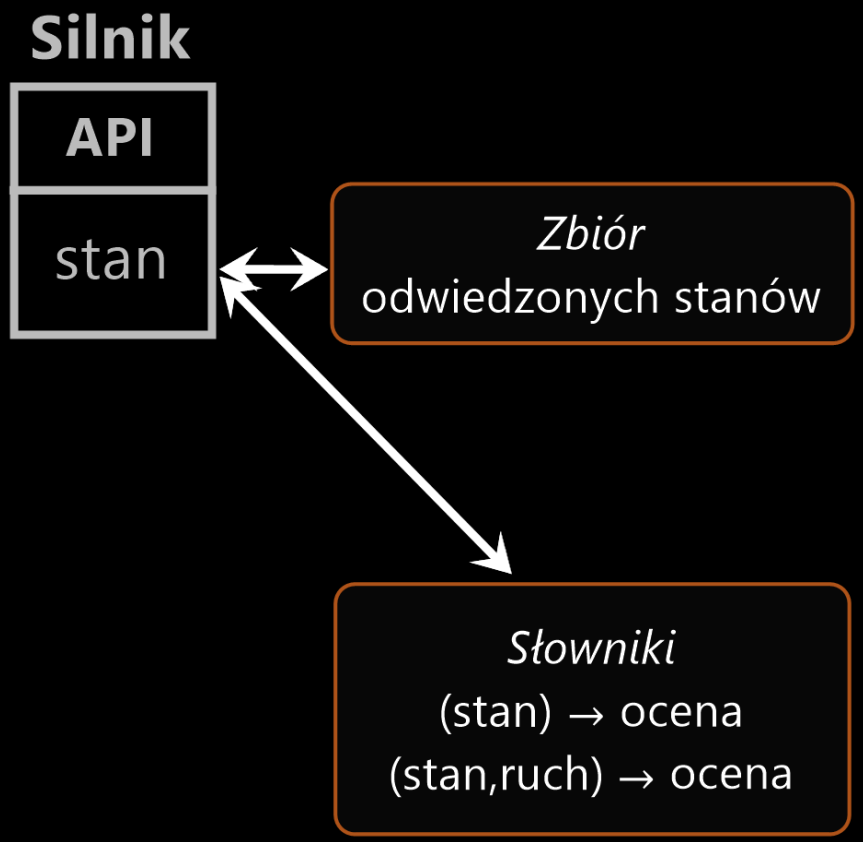
- kompaktowy sposób kodowania bitowego stanu gry (zajmuje 12 liczb)
- realizację za pomocą operacji bitowych
- prekalkulację małych plansz

SYSTEM SAMOUCZĄCY SIĘ - ETAPY TWORZENIA

struktury pomocnicze:

- przechowanie odwiedzonych stanów
- przypisywanie ocen stanom i ruchom ze stanów
- Klucze haszujące służą jako identyfikatory

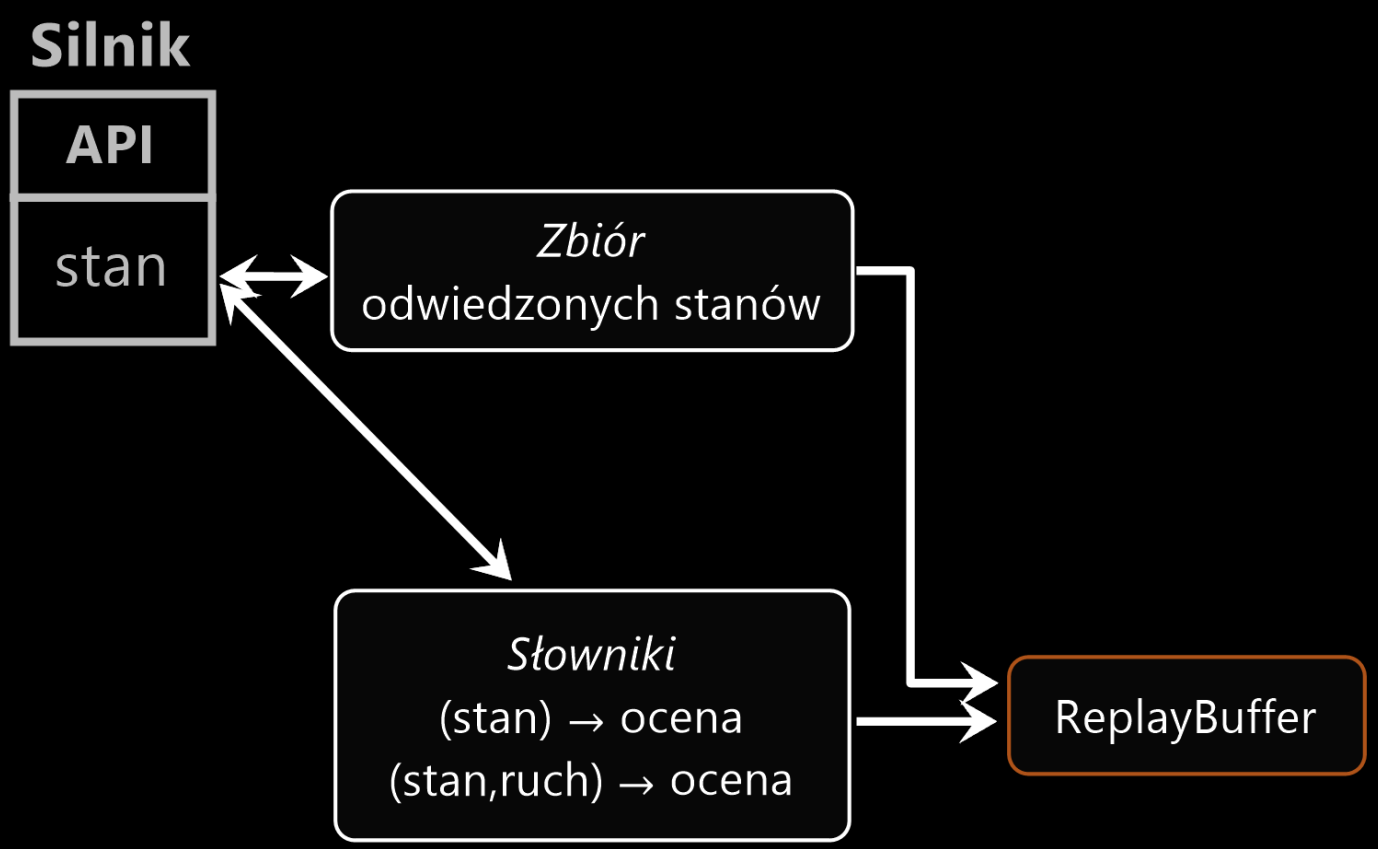
Zrealizowalizowane z wykorzystaniem kontenerów robin-set i robin-map

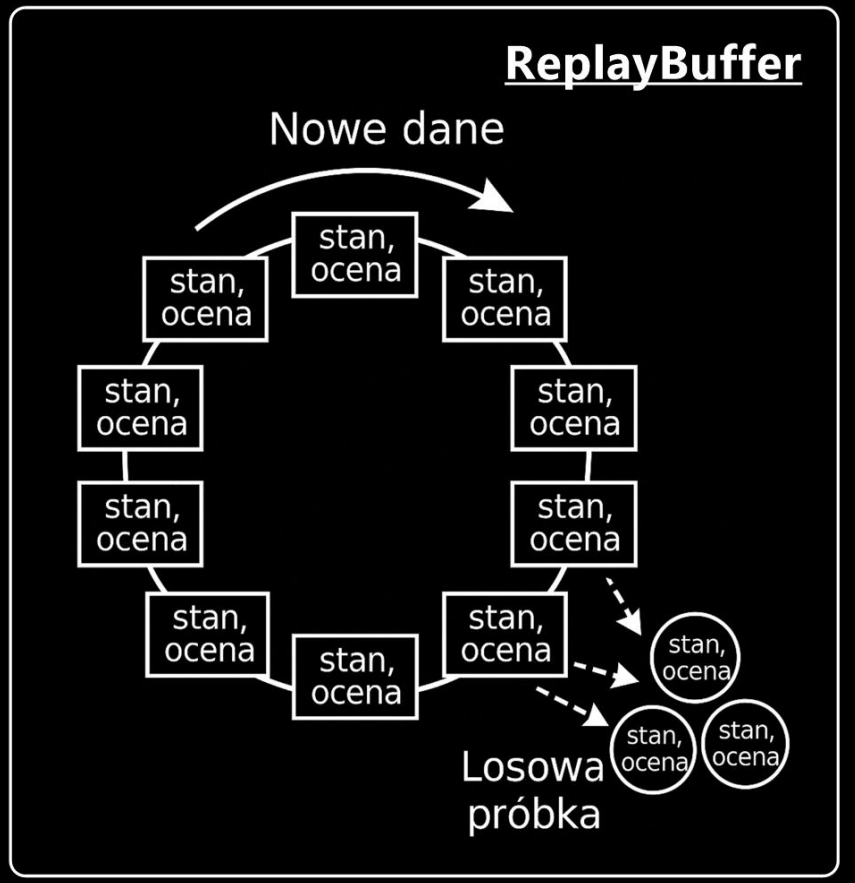


SYSTEM SAMOUCZĄCY SIĘ - ETAPY TWORZENIA

Cykliczny bufor:

- do składowania odwiedzonych i ocenionych Descentem stanów należało stworzyć cykliczny bufor.
- zwraca losową próbkę z zebranych danych.

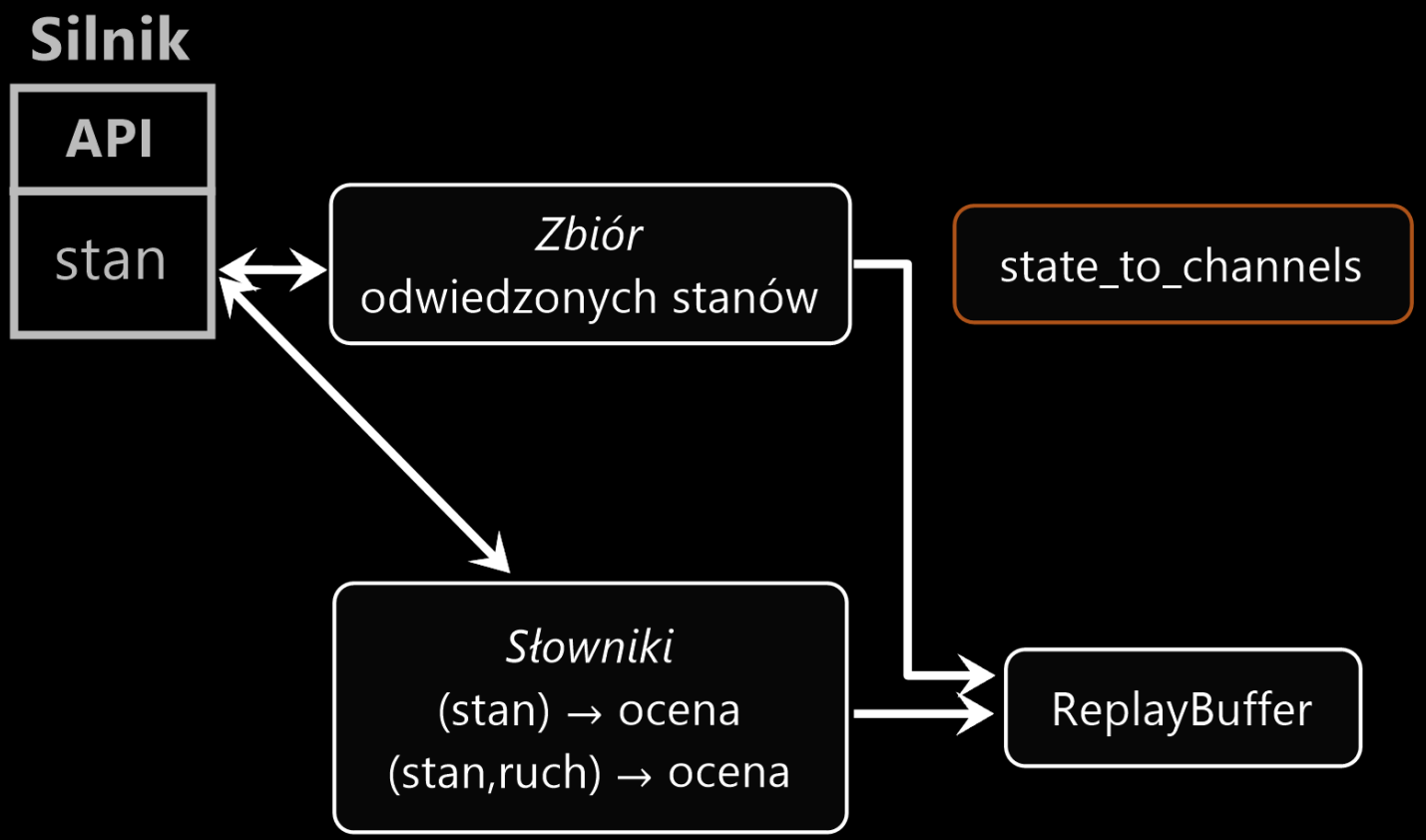




SYSTEM SAMOUCZĄCY SIĘ - ETAPY TWORZENIA

moduł pomocniczy

- konwertuje stan gry do postaci kanałów wejściowych dla sieci neuronowej.



SYSTEM SAMOUCZĄCY SIĘ - ETAPY TWORZENIA

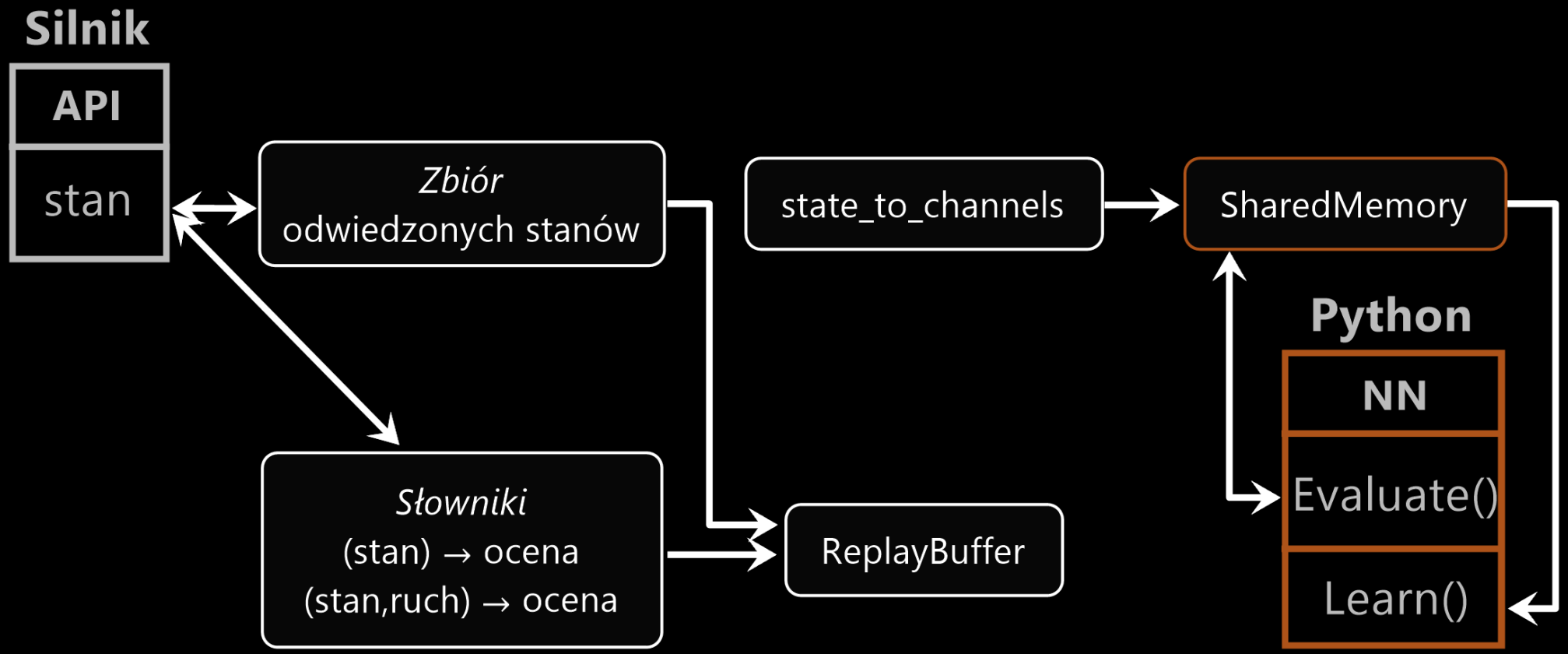
Część związana z siecią neuronową realizowana w Pythonie

Python oferuje więcej możliwości w zastosowaniach związanych z sieciami neuronowymi

Interpreter Pythona pracuje wewnątrz procesu C++

To pozwala na:

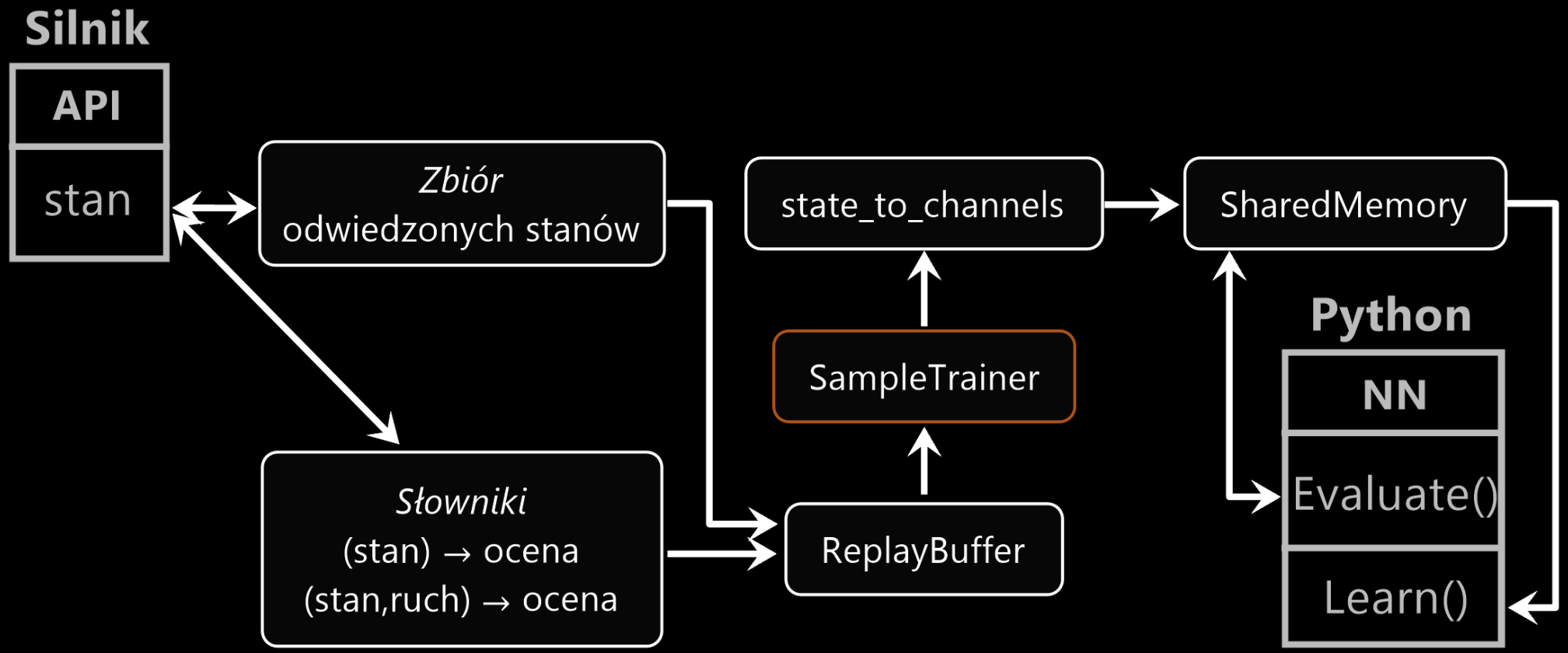
- użycie wspólnej pamięci do wzajemnej wymiany danych
- wywoływania funkcji Pythona z minimalnymi opóźnieniami



SYSTEM SAMOUCZĄCY SIĘ - ETAPY TWORZENIA

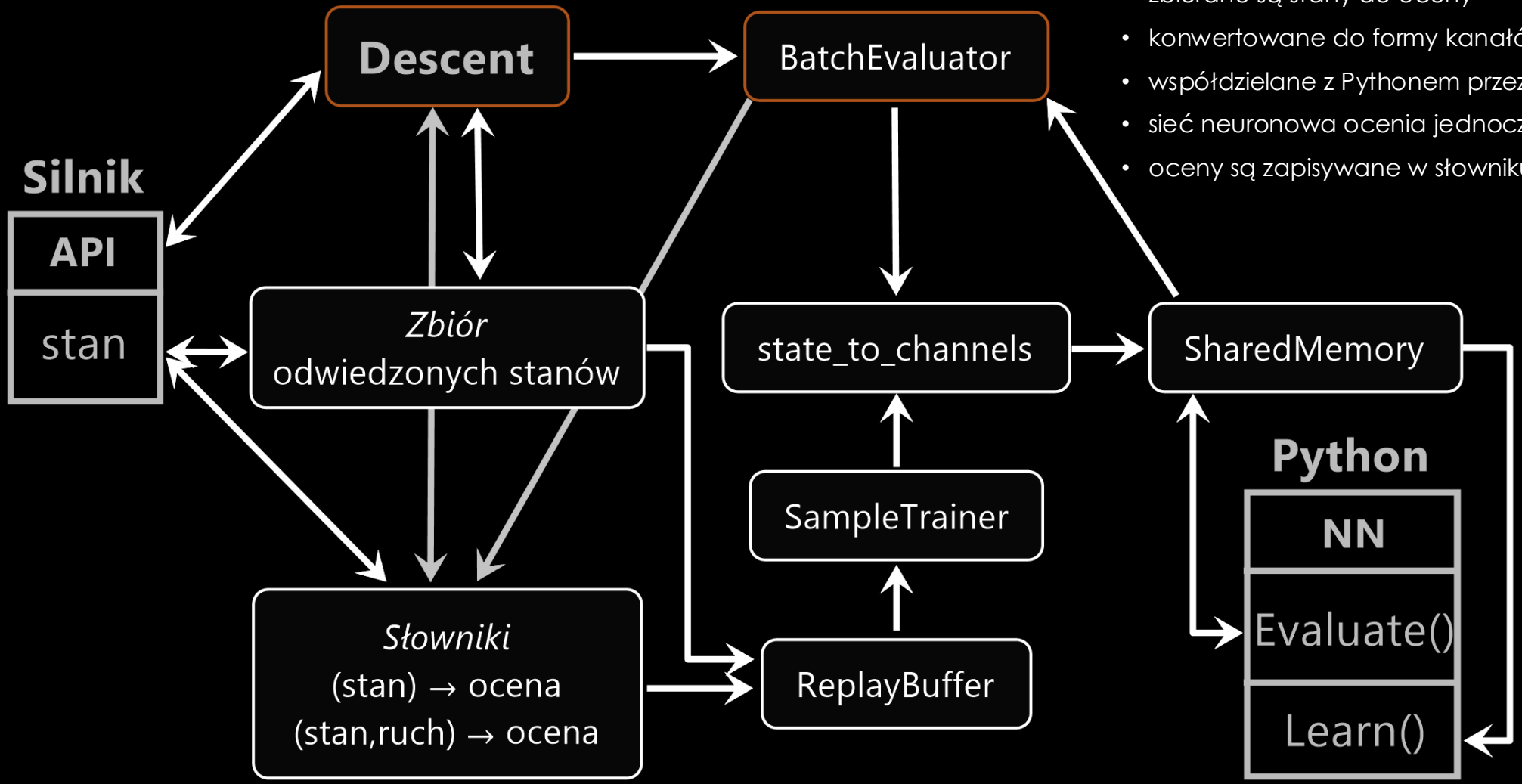
SampleTrainer:

- pobiera dane z ReplayBuffera
- konwertuje do reprezentacji kanałowej
- zapisuje do wspólnej pamięci
- wywołuje funkcji treningu sieci po stronie Pythona



SYSTEM SAMOUCZĄCY SIĘ - ETAPY TWORZENIA

Descent wybiera gałąź do analizy na podstawie ocen sieci neuronowej.



BatchEvaluator

ocenia dostępne ruchy za pomocą jednego wywołania sieci neuronowej:

- zbierane są stany do oceny
- konwertowane do formy kanałów
- współdzielane z Pythonem przez moduł SharedMemory
- sieć neuronowa ocenia jednocześnie wszystkie przekazane stany
- oceny są zapisywane w słowniku jako oceny ruchów ze stanu

Moduł SelfPlay

- przeprowodza rozgrywki
- gromadzi dane treningowe

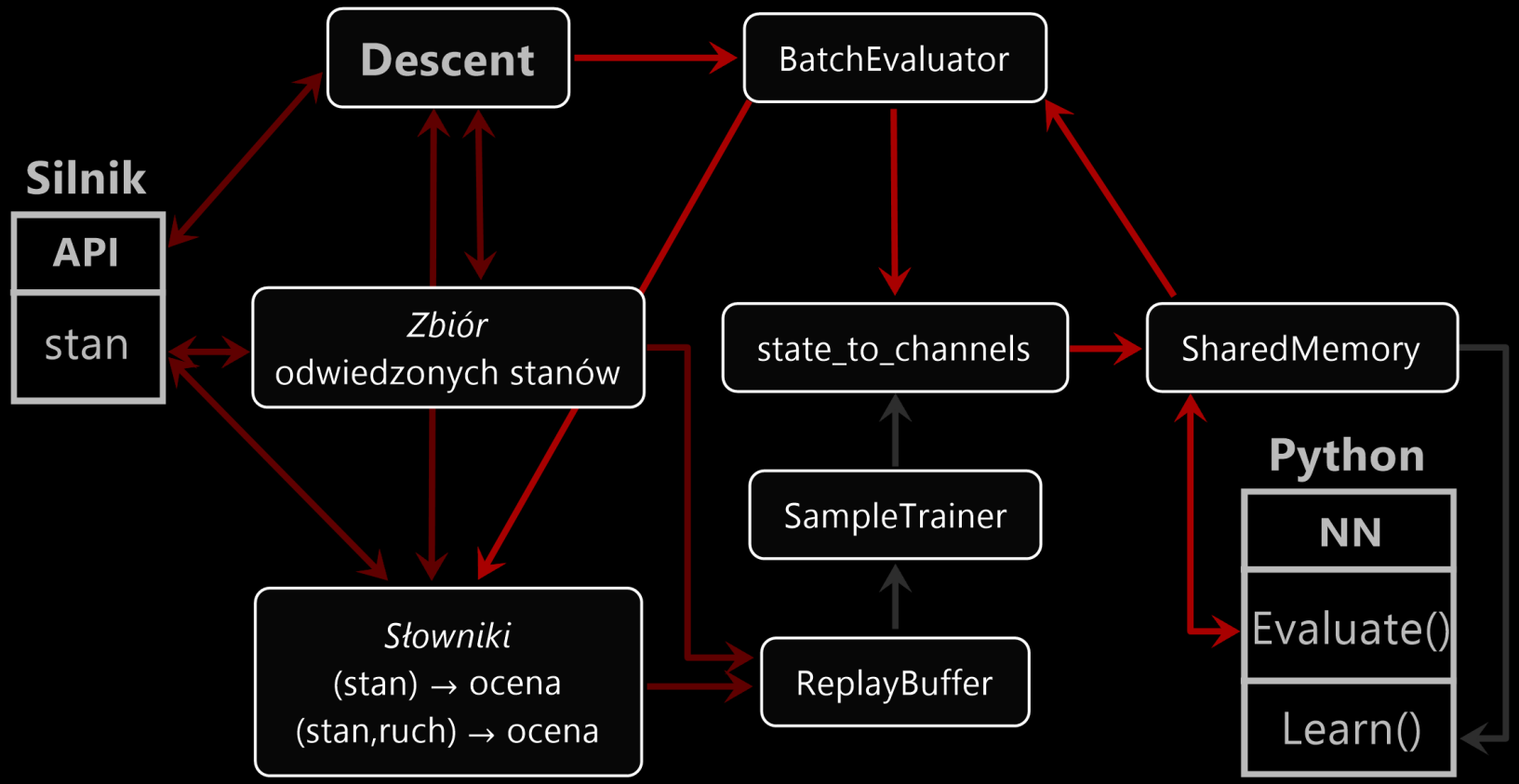
SYSTEM SAMOUCZĄCY SIĘ - PRZEBIEG DZIAŁANIA

- Proces uczenia systemu samouczącego się stanowią 2 etapy (jedna epoka uczenia)
- Pierwszy etap to zbieranie danych treningowych
- Drugi etap stanowi trening sieci

SYSTEM SAMOUCZĄCY SIĘ - PRZEBIEG DZIAŁANIA 1

Etap 1: Zbieranie danych

- realizowanie gier, dopóki nie zostanie zebrana liczba danych o rozmiarze próbki
- w pojedynczej grze, dopóki gra nie jest w stanie terminalnym:
  *algorytm Descent ocenia ruchy z bieżącego stanu gry poprzez sieć neuronową (w tym konwesja do postaci kanałów i zapis do współdzielonej pamięci)
  *wybór ruchu
  *aktualizacja stanu gry
- dodanie danych do RelpayBuffer

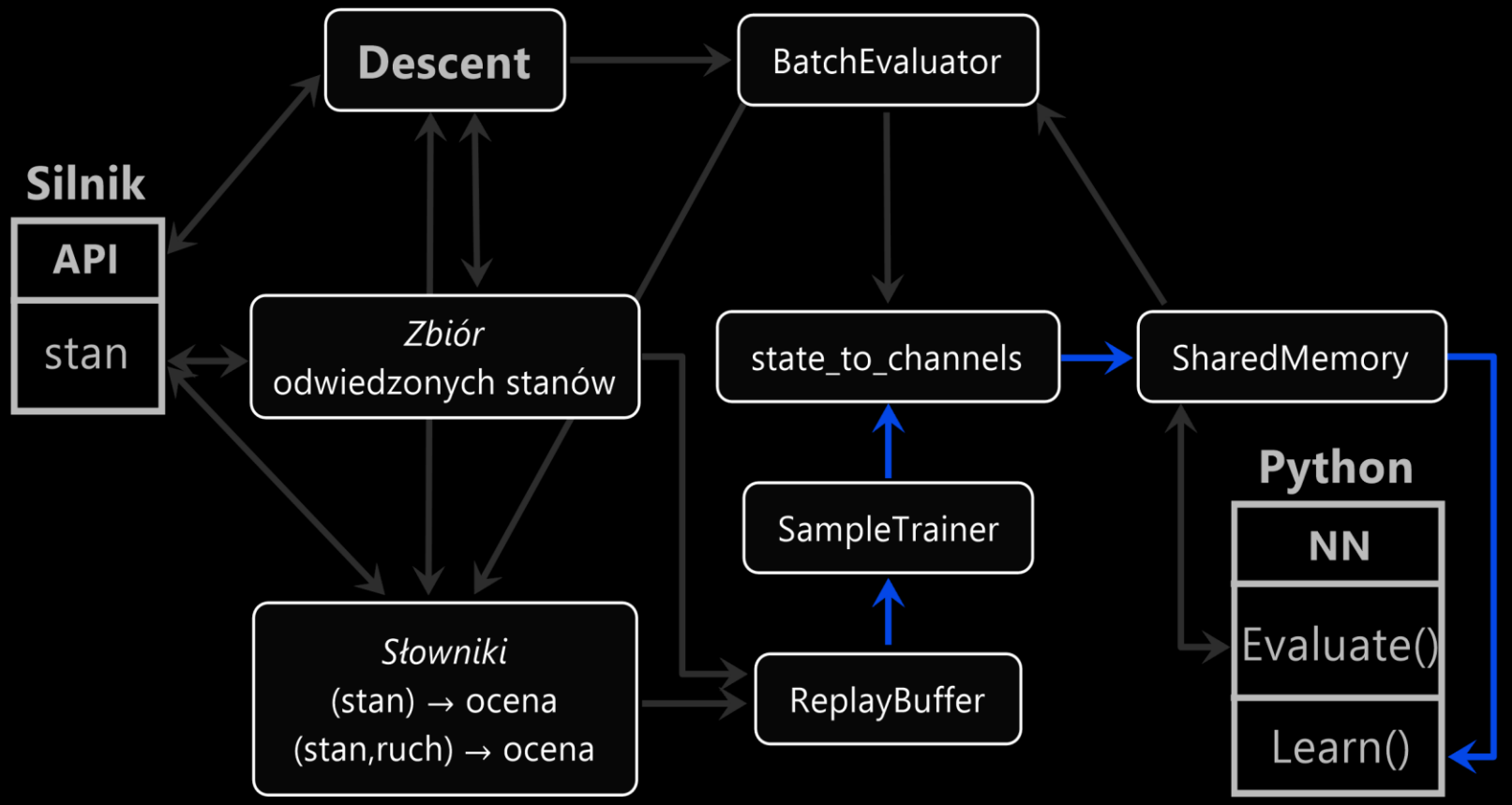


Schemat 1: przebieg działania systemu samouczącego się podczas etapu 1.

SYSTEM SAMOUCZĄCY SIĘ – PRZEBIEG DZIAŁANIA 2

Etap 2: Trening sieci:

- pobranie losowej próbki danych o określonym rozmiarze z ReplayBuffer
- konwersja danych do postaci kanałów i zapis do pamięci współdzielonej
- wykonywanie funkcji Learn() po stronie Pythona
- dane są augumentowane (obroty i odbicie lustrzane)
- uczenie sieci (dane przechodzą raz)
- zapisanie zaaktualizowanej wersji sieci (checkpointa)



Dalej Descent używa nowej wersji sieci, co pozwala na:

- Lepszą analizę stanów
- Zbieranie dokładniej ocenionych danych do kolejnego treningu - sieć uczy się na coraz lepiej ocenionych danych

Schemat 2: przebieg działania systemu samouczącego się podczas etapu 2.

SYSTEM TESTOWY - TWORZENIE

- testowanie zarówno checkpointów jak i graczy innego typu – Minimax i SaltZero (TCP+protokół komunikacyjny, klasa abstrakcyjna i opracowanie struktury pliku konfiguracyjnego).
- Opracowywanie sposobu testowania:

  - brak możliwości realizacji gier między każdą parą (wiele par, po kilka gier na parę dla dokładności) ze względu na zasoby. Należało wybrać część par do gier bezpośrednich.

  - nie można było wybrać dowolnych par. Gracze powinni być połączeni jak najściślej przez wspólnych przeciwników, pary dobrane tak, że można oszacować siłę każdego.

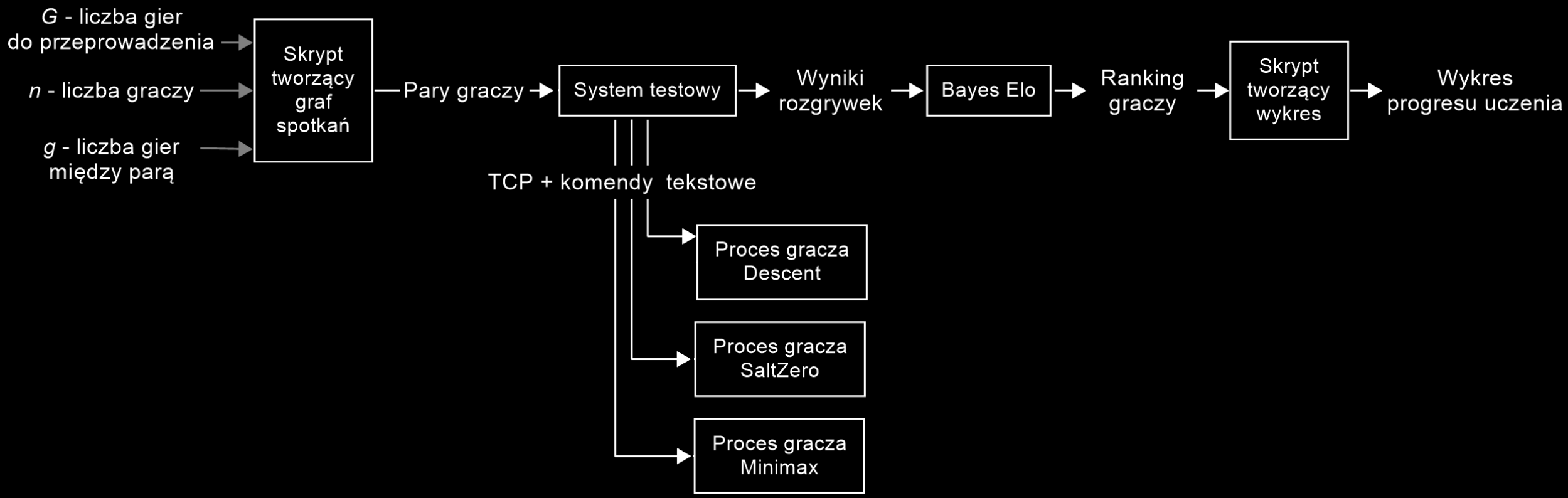
  - Rozwiązanie: informacje o grafie spotkań i maksymalnej łączności algebraicznej

- Przedstawienie wyników gier - ranking Bayes ELO (bezpośrednie i pośrednie uwzględnianie połączenia)

SYSTEM TESTOWY - DZIAŁANIE

Po zakończonym uczeniu, następuje proces testowania:

1. Zgromadzone checkpointy (gracze) dobierane są w pary na podstawie grafu spotkań.
2. System testowy przeprowadza określoną liczbę gier między dobranymi parami. Dla każdej pary:
   - łączy się poprzez TCP z każdym graczem z pary
   - dla każdej gry wybiera ruch początkowy ze zdefiniowanych ruchów
   - za pomocą komend tekstowych przeprowadza dalszą grę (przenoszenie ruchów między graczami, kontrola wyniku gry)
   - zapisuje wynik gry do pliku wynikowego
3. Program Bayes Elo tworzy ranking na podstawie wyników z systemu testowego.
4. Tworzony jest wykres postępu uczenia.

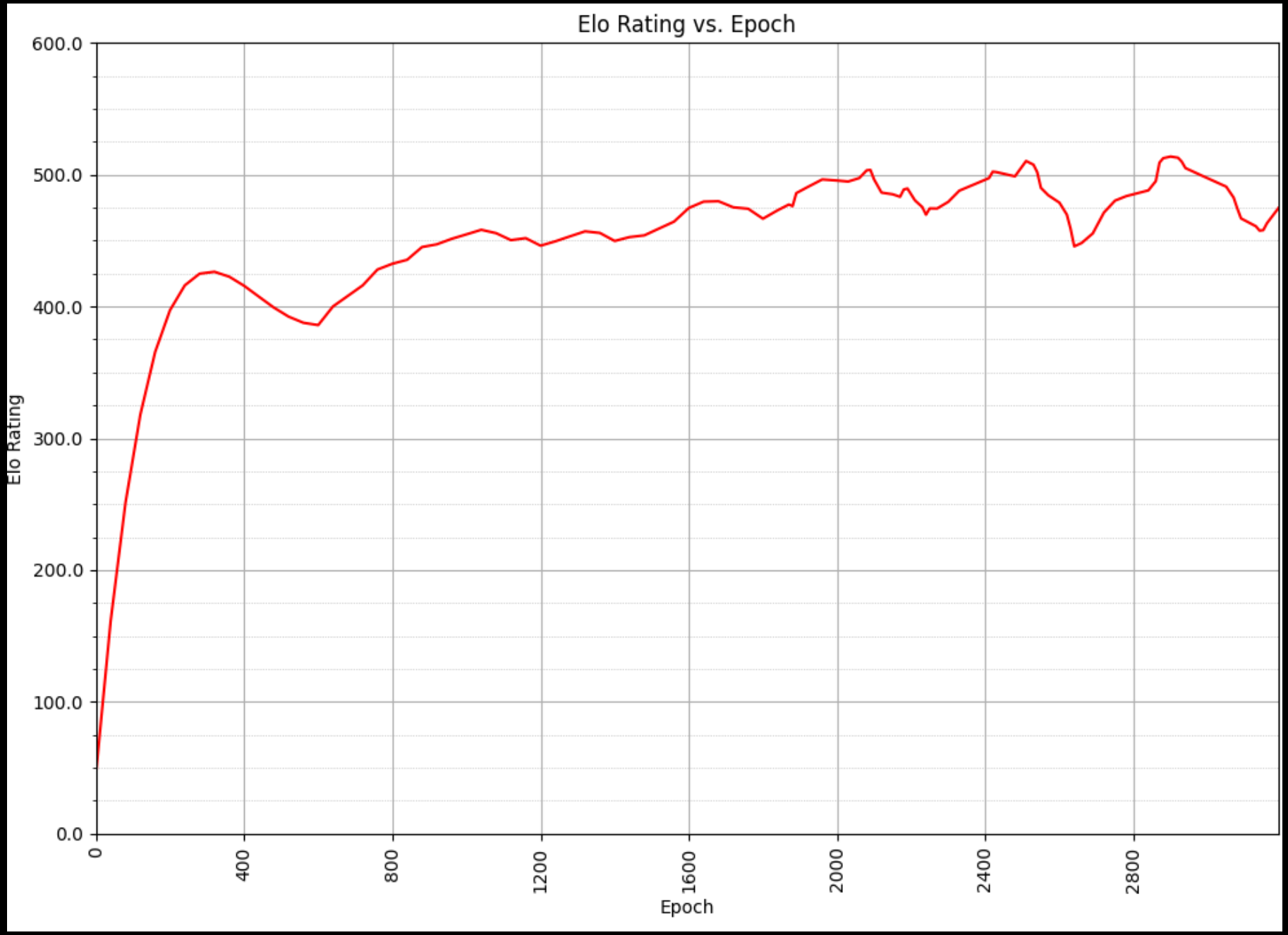


PRZEBIEG UCZENIA SYSTEMU – PIERWSZE UCZENIE

Pierwsze uczenie:

- Uczenie bez ekspertów - do zbierania danych i treningu używana jest wersja (checkpoint) bieżąca sieci
- Uczenie przez 3000 epok, zebrano 100 checkpointów
- Zebrane checkpointy przetestowano i zestawiono w rankingu Bayes ELO
- System rozwija się, ale uczenie jest niestabilne i wolne (wykres na kolejnym slajdzie)

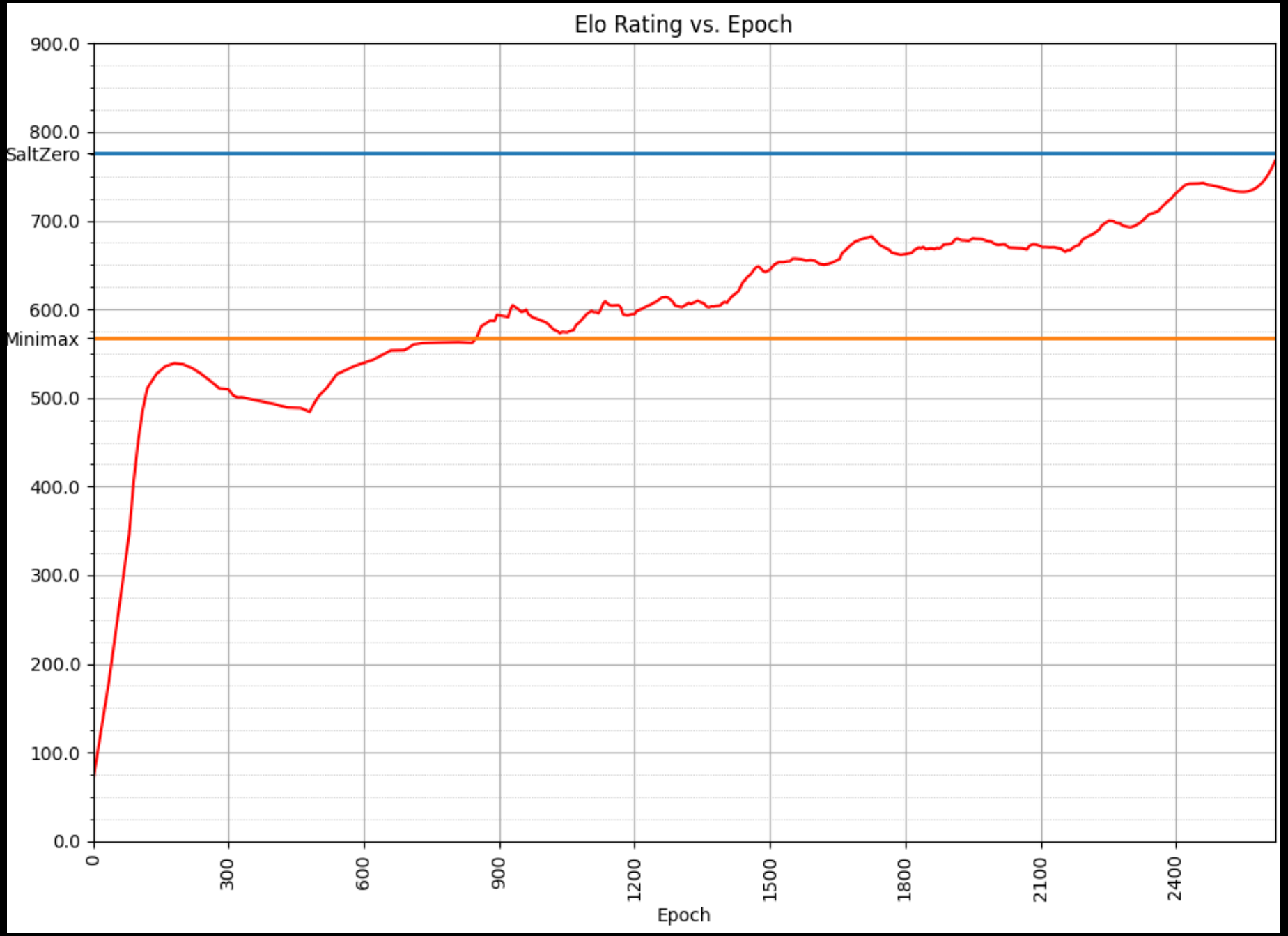
WYNIKI (PIERWSZE UCZENIE SYSTEMU)



PRZEBIEG UCZENIA SYSTEMU Z EKSPERTAMI

Nowa architektura systemu samouczącego się z ekspertami

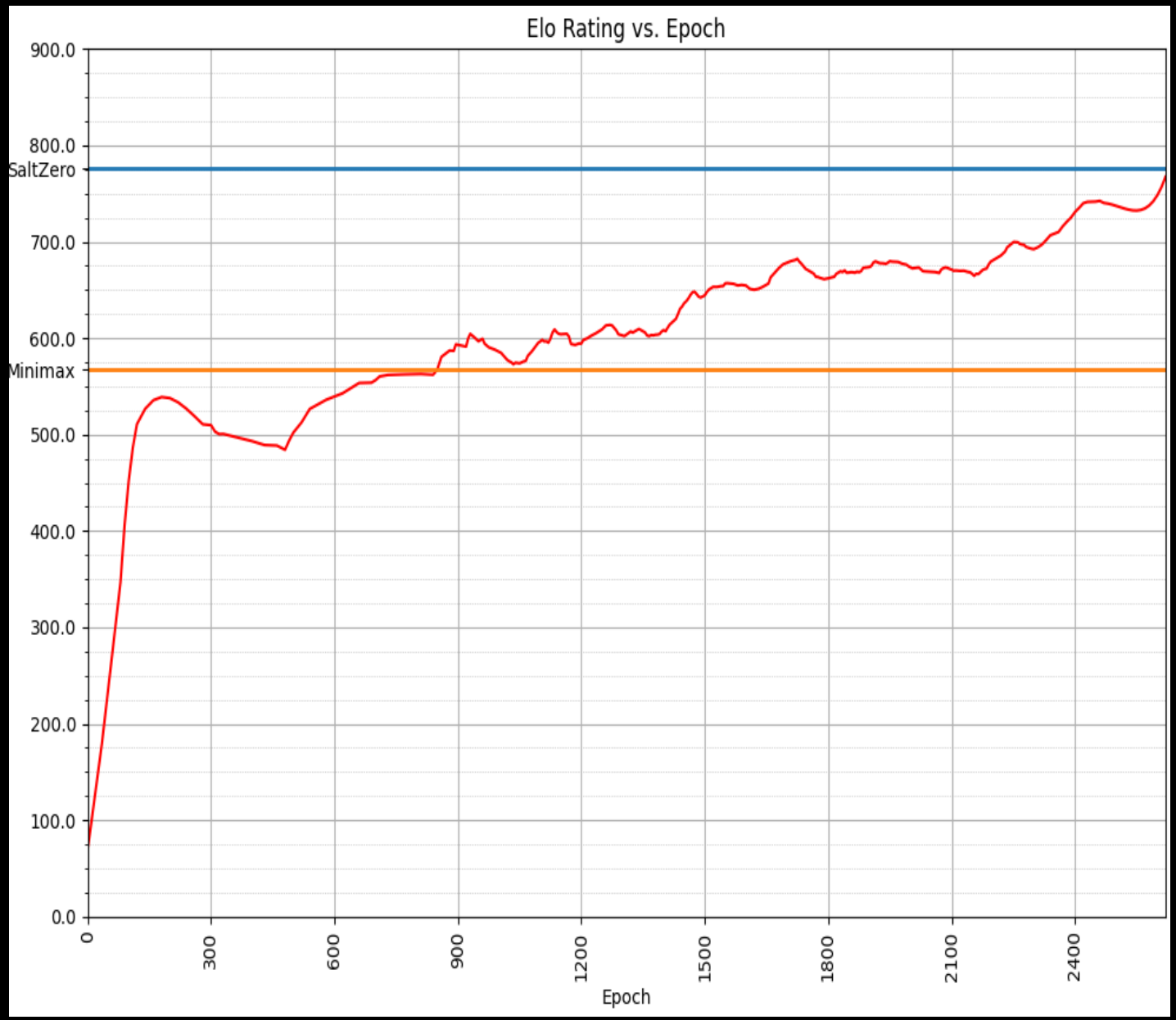
- Dodano pomocniczą sieć do ładowania eksperckiech checkpointów
- Descent korzysta z bieżącej sieci **i/lub** ekspertów do oceny i generowania danych (Możliwość regulowania udziału ekspertów podczas uczenia)
- Szybka nauka nowej sieci do poziomu eksperta
- Stopniowe zmniejszanie udziału eksperta, pojawianie się nowych ekspertów
- Stabilniejszy i szybszy proces nauki
- ~30 000 testowych gier (dokładniejsze oceny ELO)



WYNIKI (DRUGIE UCZENIE SYSTEMU)

Wykres przedstawia wyniki drugiego uczenia systemu, wykorzystując ekspertów.

ZALETY SYSTEMU Z EKSPERTAMI



Możliwości badawcze systemu z ekspertami:

- Szybkie uczenie nowej sieci przez eksperta od zera do wysokiego poziomu
- Powtarzalne warunki do porównywania wpływu hiperparametrów (stały ekspert, ten sam checkpoint początkowy)
- Możliwość trenowania sieci o innej architekturze przez eksperta
- Szerokie zastosowania do badań oraz optymalizacji architektury sieci neuronowych

ULEPSZENIA

1. **Głębsze ocenianie stanów nieterminalnych w algorytmie Descent niż w klasycznym Best-First Search, pozwalające na precyzyjniejszą ocenę pozycji przy zachowaniu wydajności dzięki wsadowemu przetwarzaniu na GPU.**

   Czyli dla każdego dostępnego ruchu zbieramy wszystkie możliwe odpowiedzi przeciwnika.
   Ruch jest oceniany na podstawie najlepszego ruchu odpowiedzi, czyli zwiększamy głębokość oceny sieci neuronowej o jeden krok.

   zysk: dokładność ocen, algorytm generuje bardziej jakościowe dane uczące, co ulepsza proces samouczenia bez dodatkowych kosztów.

2. **Wykorzystanie podczas samouczenia dwóch wersji sieci neuronowej, które grają ze sobą. Do bufora treningowego przekazywane są dane zebrane przy użyciu sieci, która wygrała.**
3. Trenowanie sieci na różnych próbkach, testowanie każdej aktualizacji w meczach przeciwko poprzedniej wersji oraz uwzględnianie w finalnej aktualizacji jedynie tych przykładów, które nie obniżają jej win-rate.
4. Użycie terminalnego algorytmu Minimax do generowania dodatkowych danych treningowych.
5. Znalezienie lepszej architektury sieci poprzez zastosowanie przyspieszonego uczenia przy użyciu sieci eksperckiej.

WYNIKI PO ULEPSZENIACH

Wersja Descent	Liczba gier	Stosunek punktów (Descent:SaltZero)	Wynik Descent	Remisy
Bot v1 (przed ulepszeniami)	300	135,0 : 165,0	45 %	45 %
Bot v2 (po ulepszeniach)	300	161,0 : 139,0	54 %	38 %

PODSUMOWANIE

Nasza praca może również **służyć jako przewodnik** dla osób zainteresowanych tworzeniem systemów samo-uczących się.

Zawiera:

- podstawy teoretyczne
- kompletną ścieżkę realizacji:
  od ogólnej idei do implementacji i integracji poszczególnych modułów w system samouczący się.
  Realizację frameworka do testowanie różnych gier
- przeprowadzanie całego procesu testowania
- opracowanie wyników testowania w postacie ByesELO
- rozwiązania techniczne i optymalizacyjne

DZIĘKUJEMY ZA UWAGĘ!