Marcin Zięba

Inteligencja obliczeniowa

Raport 5

Temat: **Implementacja algorytmów ewolucyjnych na GPU z użyciem TensorFlow**

TL;DR; Przygotowałem i sprawdziłem podstawową implementację algorytmu EMAS dla problemu minimalizacji funkcji Rastrigina. Aktualna wersja implementacji nie wykorzystuje kompilacji XLA z uwagi na zmienną długość tensorów (można spróbować to zmienić w przyszłej wersji). Testy zostały wykonane na CPU, zwykłym GPU oraz lepszym GPU, pokazując poprawność implementacji oraz wzrost wydajności na GPU dla wystarczająco dużych problemów. Testy na TPU nie zostały wykonane przez niemożliwość kompilacji.

Colab: Podstawowy EMAS dla funkcji Rastrigina

Q&A

Co zostało zaimplementowane?

Zaimplementowałem podstawową wersję algorytmu EMAS (Evolutionary Multi-Agent System). Pod względem funkcjonalnym od pełnej wersji różni się głównie brakiem wysp. Pod względem technicznym jest to rozwiązanie wstępne głównie przez to, że problem oraz operatory ewolucyjne są na sztywno osadzone w implementacji.

Jakie problemy napotkałem?

Przyjęte rozwiązania sprawiają, że nie muszę zakładać żadnego górnego ograniczenia na liczebność populacji, ponieważ wszystkie operacje zostały napisane tak, aby pierwszy wymiar tensorów był zawsze nieokreślony. Przez większość czasu wydawało się to dobrym rozwiązaniem. I prawdopodobnie faktycznie jest to czystsze rozwiązanie, ale jednocześnie uniemożliwia kompilację XLA, rzucając wyjątek w pętli głównej. Prawdopodobnie jest to jednak łatwe do obejścia, więc można to sprawdzić w przyszłej wersji. Uniemożliwiło to także uruchomienie testów na TPU (najwyraźniej przed uruchomieniem kodu na TPU dokonywana jest niejawna kompilacja do XLA - dobrze wiedzieć).

Jaką funkcję optymalizowałem?

Funkcja Rastrigina dla n-wymiarów. Testowałem w wersji lekkiej 10-D, oraz cięższej 10000-D.

Jakie operatory genetyczne testowałem?

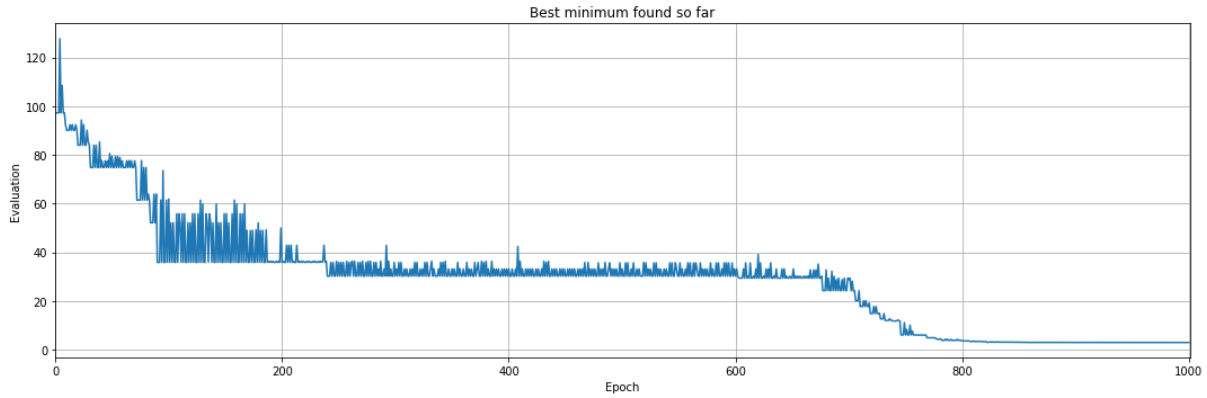
Użyłem operatora krzyżującego, biorącego losowy punkt z hiperkuli, której średnica przechodzi przez oba punkty z przestrzeni rozwiązań biorące udział w reprodukcji.

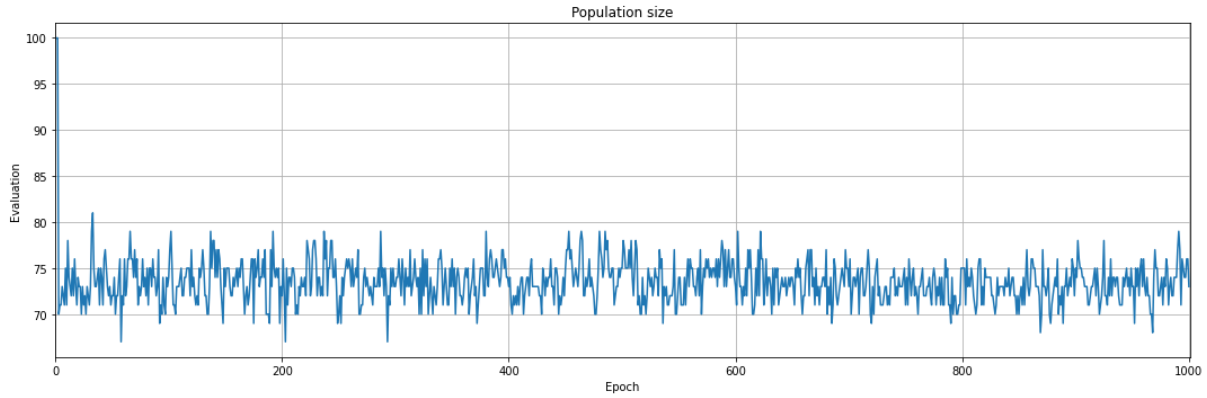
Jakie hiper-parametry symulacji przyjąłem?

Początkowa energia: 5 dla każdego agenta
Zmiana energii po walce: 2 (przegrany -2, wygrany +2)
Koszt reprodukcji: 2.5 (obu agentów -2.5, nowy agent startuje z 5)

Czy implementacja okazała się poprawna i zbiegała?

Wygląda na to, że tak. Poniżej przedstawione są statystyki uruchomienia dla 100 agentów w populacji, dla 10-wymiarowej funkcji Rastrigina w obrębie 1000 epok.





Pierwszy wykres przedstawia zmianę najlepszego rozwiązania w czasie. Ewaluacje są wykonywane tylko w obrębie areny walk. Tłumaczy to widoczne wahania, wynikające z tego, że aktualnie najlepszy kandydat znalazł się akurat na arenie reprodukcji, a wśród pozostałych były tylko gorsze.

Drugi wykres przedstawia zmianę rozmiaru populacji w czasie. Jak widać, dla zadanych hiper-parametrów populacja bardzo szybko osiąga stabilny rozmiar około 70-75 osobników.

Pierwszy wykres sugeruje zbieżność do pewnego (całkiem rozsądnego) rozwiązania. Analiza zmienności kandydatów pokazuje, że po 1000 epok każdy z agentów znalazł się w mniej więcej tym samym miejscu (odchylenie standardowe rzędu 0.0003), tzn. całkiem ustała eksploracja.

```
1   np.std(solution, axis=0)
```

```
array([0.00037241, 0.00034522, 0.00035206, 0.0003624 , 0.00037756,
       0.00032986, 0.00035547, 0.00037317, 0.00036911, 0.00038035],
      dtype=float32)
```

Sprawdziłem także sumę energii wszystkich agentów na koniec symulacji i faktycznie jest ona zachowywana.

```
1   solution, energy, best_scores, pop_size = emas_rastrigin(k=100, epochs=1000, n=10)
```

```
1   np.sum(energy)
```

```
500.0
```

Początkowa energia wynosi 5 dla każdego agenta, więc całkowita pula energii w systemie z setką agentów wynosi 500. Jak widać, po tysiącu epok nadal wynosi 500.

Jak wygląda kwestia czasu obliczeń?

Rozmiar problemu: niewielki

- liczba agentów: 100
- liczba wymiarów: 10
- liczba epok: 1000

	Czas
CPU, czyste funkcje Pythonowe	20.3378s
CPU, funkcje użytkowe i operatory udekorowane przez tf.function	15.6573s
CPU, całość udekorowana z flagą reduce_retracing=True	0.2664s
GPU, całość udekorowana z flagą reduce_retracing=True	1.6891s

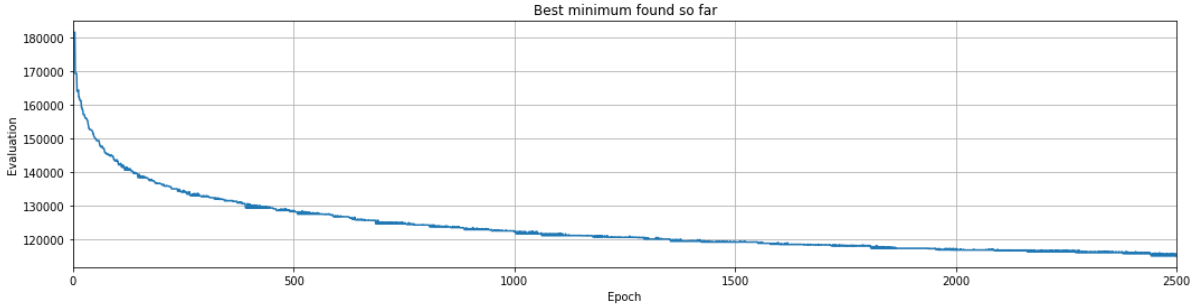
Rozmiar problemu: raczej duży

- liczba agentów: 2000
- liczba wymiarów: 10000
- liczba epok: 2500

Wszystkie wpisy w tabelce zakładają pełną dostępną optymalizację, tj. całość udekorowana z flagą reduce_retracing=True (kompilacja XLA aktualnie jest niedostępna - potrzebne są drobne zmiany w kodzie).

	Czas
CPU	396.0635s
GPU - Standardowe	15.5300s
GPU - Premium	4.3887s
GPU - Premium (50,000 epok! - 20x więcej)	88.4901s
TPU	-

Algorytm nadal jest zbieżny pomimo znacznego zwiększenia wymiarowości.



Zbiega jednak bardzo wolno. Poniżej jest wykres najlepszych rozwiązań dla 50,000 epok.