SPRAWOZDANIE

z projektu wykonywanego w ramach przedmiotu

Algorytmy w inżynierii danych

Wykonane przez:

- Aliaksandr Karolik, nr indeksu 295138
- Piotr Jeleniewicz, nr indeksu 291072

Temat:

W ramach projektu zdecydowaliśmy się na realizację tematu nr 1 – **Automatyczne Różniczkowanie.** Porównaniu zostały poddane dwa algorytmy różniczkowania automatycznego:

- różniczkowanie w przód,
- różniczkowanie w tył.

Obecny postęp prac:

Oba algorytmy różniczkowania zostały zaimplementowane w języku Julia, przy wykorzystaniu przeciążania operatorów. Do chwili pisania tego sprawozdania zaimplementowane zostały oba algorytmy różniczkowania dla następujących funkcji matematycznych:

- podstawowe funkcje matematyczne takie jak: dodawanie, odejmowanie, mnożenie, dzielenie, sinus, cosinus, funkcja wykładnicza,
- funkcja ReLU,
- funkcja Softmax – obecnie funkcja ta została w pełni zaimplementowana tylko dla algorytmu różniczkowania w przód. W przypadku różniczkowania w tył funkcja ta niepoprawnie oblicza wartości leżące na diagonali macierzy.

Dodatkowo w przypadku obu algorytmów zaimplementowana została funkcja licząca Jakobian.

Przedstawienie działania:

W celu przedstawienia działania naszego kodu, przeprowadziliśmy porównanie czasów liczenia oraz ilości używanej pamięci podczas obliczania gradientu funkcji Rosenbrocka za pomocą algorytmu różniczkowania w przód i w tył. Pomiary zostały przeprowadzone przy pomocy pakietu ***BenchmarkTools.***
Uzyskane wyniki prezentują się następująco:

Algorytm	Ilość elementów w wektorze wejściowym	Średni czas obliczeń	Ilość wykorzystywanej pamięci
Różniczkowanie w	10	14,09 µs	12,42 KB

przód	100	747,31 µs	486,13 KB
	1000	63,78 ms	32,03 MB
Różniczkowanie w tył	10	23,58 µs	16,05 KB
	100	233,19 µs	150,39 KB
	1000	2,48 ms	1,45 MB

Jak łatwo zauważyć z powyższej tabeli nasza implementacja różniczkowania w tył zgadza się z oczekiwaniami – algorytm różniczkowania w tył znacznie szybciej oblicza gradient funkcji Rosenbrocka, przy jednoczesnym mniejszym zużyciu pamięci. Dzieje się tak ponieważ różniczkowanie w tył oblicza w jednym kroku wszystkie pochodne cząstkowe pierwszego rzędu.

W celu prezentacji atutów różniczkowania w przód przeprowadziliśmy dodatkowo eksperyment polegający na obliczeniu macierzy Jacobiego funkcji składającej się z 6 podfunkcji, do obliczenia wartości których wykorzystywane są 3 zmienne:

Algorytm	Średni czas obliczeń	Ilość wykorzystywanej pamięci
Różniczkowanie w przód	4,42 µs	5,20 KB
Różniczkowanie w tył	14,05 µs	18,81 KB

Zgodnie z oczekiwani różniczkowanie w przód obliczyło macierz Jacobiego w krótszym czasie i przy mniejszym zużyciu pamięci, co wynika z natury różniczkowania w przód, które w jednym kroku oblicza wartości pochodnych dla wszystkich podfunkcji po jednej zmiennej.

Podsumowanie:

Obecnie udało się nam zaimplementować oba algorytmy różniczkowania automatycznego, a także zaobserwować jakimi cechami odznaczają się oba algorytmy i do jakich zastosowań powinno się je wykorzystywać. W drugiej części prac nad projektem zamierzamy skupić się na udoskonaleniu obecnych implementacji algorytmów, a zwłaszcza na poprawieniu implementacji algorytmu różniczkowania w tył. Zamierzamy także dokonać dokładniejszej analizy porównawczej obu metod, w celu dobrania dobrych przypadków testowych, które zostaną wykorzystane w raporcie końcowym.

Link do repozytorium:

https://github.com/Ligerd/Algorithms_in_data_engineering

Bibliogragia:

1. Reverse-mode automatic differentiation: a tutorial, https://rufflewind.com/2016-12-30/reverse-mode-automatic-differentiation?fbclid=IwAR2Z1l02U5tzndznM6e9wFt0hoqgTcwwf80Nly-cZ_qsZzNhqqJ0Esn-KlQ,
2. Sindre Grøstad, Automatic Differentiation in Julia with Applications to Numerical Solution of PDEs, https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2624618?locale-attribute=en,
3. Informacje przedstawione w ramach wykładu