Доброго дня! Моє ім'я Віктор Лошак, і сьогодні я представляю свою курсову роботу на тему

СЛАЙД

'Паралелізація алгоритму навчання нейромережі за допомогою Python'. Метою роботи було вивчити способи паралелізації нейронних мереж, імплементувати паралельний алгоритм та досягти з ним прискорення до 1.2

СЛАЙД

Трохи про застосування нейронних мереж сьогодні.

СЛАЙД

Важливо зазначити що під час опису алгоритму я використовуватиму два терміни.

Міні-батч— декілька прикладів даних згрупованих в одну матрицю. Міні батч обробляється нейромережею одним проходом— це означає що при стандартній реалізації алгоритму один мінібатч подається на вхід мережі як матриця де рядок представляє один зразок даних. Градієнт обчислюється для міні-батчу а не для окремих рядків всередині нього. Це дозволяє зменшити вплив викидів в даних на результати ефективності нейромережі.

Група міні-батчів — в цій роботі під цим поняттям розуміється об’єднання декількох міні-батчів в одну групу, для якої окремі градієнти отримані для індивідуальних міні-батчів усереднюються. При паралелізації алгоритму ми будемо розподіляти обробку міні-батчів всередині цієї групи на різні процеси, що і довзолить досягти прискорення.

Слайди аж до паралелізації алгоритму без тексту

ПОЯСНЕННЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ:

Основна відмінність між паралельним і послідовним алгоритмом в тому як обробляються групи міні-батчів В послідовному алгоритмі ми використовуємо цикл для розрахунку градієнтів для кожного міні-батчу в групі, а потім усереднюємо це значення.   
В паралельній реалізації ми використовуємо пул процесів для паралельного розрахунку градієнтів міні-батчів всередині групи.

Як працює пул процесів в python

Стандартний інтерпретатор Python (CPython), який ми використовуємо в цій курсовій, має певні обмеження, такі як Global Interpreter Lock (GIL)[13], який ускладнює виконання багатопроцесорних обчислень з істинною паралельністю на одному інтерпретаторі. Це необхідно, тому що внутрішня пам'ять Python не є безпечною для використання в багатопоточних програмах. Для обходу цього обмеження Ми використовували бібліотеку *Multiprocessing, оскільки інші засоби є або неприйнятними або не доцільними для виконання завдання*:

ця бібліотека дозволяє створювати окремі процеси, кожен з яких має свій власний екземпляр інтерпретатора Python з власним GIL. Це робить можливим паралельне використання декількох ядер процесора, але також створює мінуси, як от те, що кожен процес займає більше пам’яті і потребує тривалого часу для розгортання. Бібліотека дозволяє використовувати багатопроцесорність за допомогою класів Process, Pool (для створення пулу процесів), Queue та Pipe (для обміну даними між процесами) [14]

В коді ми створюємо пул процесів, кількість процесів в якому за замовчуванням визначається програмним середовищем(те скільки процесів операційна система готова виділити для роботи програми). Але ми можемо встановлювати верхню межу.

Сам код, а точніше pool.starmap(

process\_batch, [(batch, W1, b1, W2, b2, W3, b3) for batch in batch\_group])

Цей код за принципом роботи можна порівняти з послідовним застосуванням з методів MPI Scatter та MPI.Reduce в джава. Але на відміну від джава, пайтон дуже спрощує операцію, приховуючи виділення буферів для переміщення даних між процесами, типу даних всередині масиву який ми хочемо відправити процесам воркерам і т.д. За інтерфейсом :

-------------------------------

MPI\_Scatter

Операція переміщення даних. Розподіляє різні повідомлення від одного джерела завдань до кожного завдання у групі.

MPI\_Scatter (&sendbuf,sendcnt,sendtype,&recvbuf, recvcnt,recvtype,root,comm)

MPI\_SCATTER (sendbuf,sendcnt,sendtype,recvbuf, recvcnt,recvtype,root,comm,ierr)

Ефективність та швидкодія алгоритмів:

Тестування к