МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Навчально-науковий інститут інноваційних освітніх технологій
Кафедра комп'ютерної інженерії

СТРУТИНСЬКИЙ Петро Ігорович

Розпаралелення алгоритмів навчання згорткових нейронних мереж на основі графічних процесорів/ The parallelization for convolutional neural networks training algorithms based on graphic processors

спеціальність: 123 - Комп’ютерна інженерія
магістерська програма - Комп’ютерна інженерія

Магістерська робота

Виконав студент групи КІм-22
П.І. Струтинський
Науковий керівник: д.т.н., професор,
О. М. Березький

Магістерську роботу допущено до захисту:

ТЕРНОПІЛЬ -2018

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CPU англ. – Central Processing Unit

GPU англ. – Graphics Processing Unit

ДП – довгострокова пам`ять

ЕО – елемент обробки

ЗНМ – згорткова нейронна мережа

ОРС – оптичне розпізнавання символів

ПЕ – процесорний елемент

РекНМ – рекурентна нейронна мережа

РНМ – рекурсивна нейронна мережа

ШІ – штучний інтелект

ШНМ – штучна нейронна мережа

ВСТУП

Штучна нейронна мережа – це обчислювальна модель, заснована на структурі та функціях біологічних нейронних мереж. Вона являє собою спробу імітувати мережу нейронів людського мозку і створюється шляхом програмування звичайних комп'ютерів.

Хоча нейронні мережі існують з 1940-х років, тільки в останні кілька десятиліть вони стали важливою частиною штучного інтелекту. Це пов'язано з приходом методу під назвою «алгоритм зворотного поширення», який дозволяє мережам налаштовувати свої приховані шари нейронів в ситуаціях, коли результат не відповідає тому, на що сподівається творець.

Згорткові мережі надзвичайно успішні в практичних застосуваннях. Вони використовуються для класифікації зображень, ідентифікації особи, знака, медичних діагнозів (наприклад пухлини), виконують оптичне розпізнавання символів та лікування людей з ослабленим зором. Саме нейронні мережі допомагають групувати і класифікувати біомедичні зображення (гістологічні та цитологічні і т.п.) за класом.

Актуальність роботи. Останнім часом зріс інтерес наукової громадськості до дослідженню і розвитку теорії штучної нейронної мережі, про що свідчить численні наукові публікації. В першу чергу це пов'язане з тим, що штучні нейронні мережі знайшли широке застосування в сучасних повсякденних гаджетах. Такі завдання, як класифікація векторів, передбачення сигналів, апроксимація функцій і розпізнавання образів (безперервне розпізнавання мови і розпізнавання рукописного тексту) відносно легко вирішуються на основі моделей нейронних мереж. Класифікація є одним з найбільш часто зустрічаються завдань прийняття рішень штучними нейронними мережами.

Сьогодні комп’ютерні технології активно впроваджуються в практичній медицині і позволяють здійснювати діагностику. В роботах, посвячених проблемам автоматизації процесу медицинської діагностики, були запропоновані

принципи алгоритмізації медицинського обстеження, які використовувалися для стандартизації дій лікаря в процесі постановки діагнозу. Сьогодні ця проблема не втратила свою актуальність і для її вирішення застосовуються адаптивні системи, основані на комп'ютерній імітації мозку. Для реалізації таких систем на практиці, в багатьох видах діяльності, в тому числі в медицині і біології застосовуються штучні нейронні мережі.

Метою даної роботи є аналіз алгоритмів навчання нейронних мереж та їхня паралелізація для виявлення найбільш ефективних способів реалізації на практиці, швидких і якісних програм для обробки великого обсягу інформації. Це спростить процес прийняття рішень для фахівців багатьох областей, зокрема у медичній галузі, дозволить автоматизувати класифікацію об'єктів, проводити постійний моніторинг, прогнозувати фази розвитку.

Таким чином під час написання даної роботи були визначені наступні завдання:

– виявити переваги і недоліки згорткових нейронних мереж;

– провести детальний аналіз найбільш поширених та ефективних алгоритмів навчання нейронних мереж;

– виявити можливі методи розпаралелення нейронних мереж;

– викласти методи паралельного програмування з використанням обраної технології;

– показати на практиці реалізацію паралельних алгоритмів нейронних мереж.

Об'єкт дослідження – біомедичні зображення (гістологічні та цитологічні).

Предмет дослідження –

Методи дослідження –

Новизна даної роботи полягає в детальному аналізі реалізації розпаралелення алгоритмів навчання нейронних мереж із використанням технологій, що дозволяють використовувати графічні процесори для опрацювання великої кількості вхідних даних.

Практичне значення отриманих результатів. Пришвидшення процесу аналізу біомедичних зображень згортковими нейронними мережами дозволяє в короткі терміни діагностувати будь-які відхилення у здоров'ї людини. Це дає можливість виявляти захворювання на ранніх стадіях та проводити лікування для запобігання їх поширення.

Публікація та апробація магістерської роботи. За результатами наукових досліджень, проведених у магістерській роботі, підготовлено тези доповіді «Згорткові нейронні мережі як засіб обробки біомедичних зображень» обсягом 2 сторінка на VII Міжнародній науково-технічній конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (Тернопіль – 2018) [18].

Впровадження результатів магістерської роботи. ТОВ «Апіко Україна».

Короткий зміст магістерської роботи.

Дана робота складається з трьох розділів.

Перший розділ -

1 СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Штучні нейронні мережі – прототип людського мозку

Штучні нейронні мережі (ШНМ) – це частини обчислювальної системи, призначені для моделювання того, як мозок аналізує і обробляє інформацію. Вони є основою Штучного інтелекту (ШІ) і вирішують проблеми, які виявилися б неможливими або важкими за людськими або статистичними стандартам.

Історія ШНМ починається із ранніх днів обчислень. У 1943 році математики Уоррен Маккаллох і Уолтер Піттс побудували схему, яка б була максимально наближена до функціонування людського мозку, в якому виконувалися прості алгоритми. Ця модель проклала шлях для дослідження нейронної мережі, щоб розділити на два підходи. Один з підходів сфокусований на біологічних процесах в мозку, а інший – на застосуванні нейронних мереж до штучного інтелекту. Ця робота привела до роботи над нейронними мережами і їх зв'язком з кінцевими автоматами.

У 1957 році дослідник Корнельського університету Френк Розенблат розробив перцептрон (рисунок 1.1), алгоритм, призначений для виконання розширеного розпізнавання образів [1]. В кінцевому рахунку створює здатність машин розпізнавати об'єкти на зображеннях. Нейронні мережі Розенблата складалися з таких елементів:

– S-елемент (сенсорний) – чутливий елемент, який під дією будь-якого з видів енергії (світла, звуку, тиску, тепла тощо) виробляє сигнал рівний одиниці, або в іншому випадку — нулю;

– A-елемент (асоціативний) – логічний елемент, який дає вихідний сигнал рівний одиниці;

– R-елемент (реагуючий) – елемент, який видає сигнал одиницю, якщо сума його вхідних сигналів є строго позитивною, і сигнал нуль, якщо сума його вхідних сигналів є строго негативною. Якщо сума вхідних сигналів дорівнює нулю, вихід вважається або рівним нулю, або невизначеним [2].

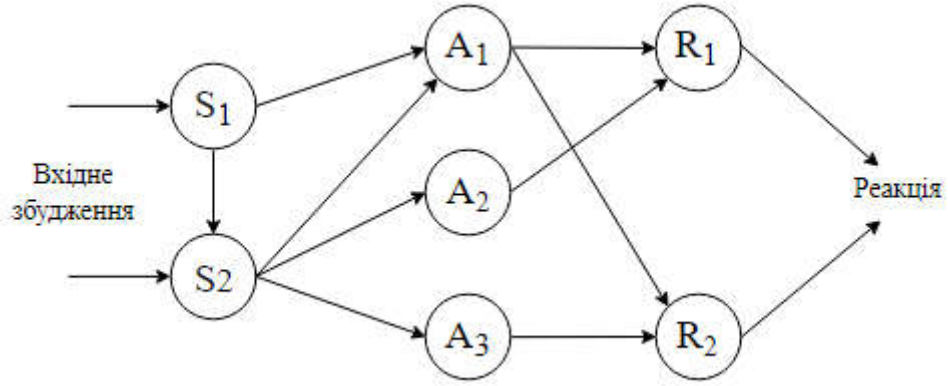


Рисунок 1.1 – Схема перцептрона Розенблата

Але перцептрон не зміг виконати свою обіцянку, і в 1960-і роки дослідження штучної нейронної мережі призупинилися.

У 1969 році дослідники Массачусетського технологічного інституту Марвін Мінськ і Сеймур Паперт опублікували книгу «Перцептрони», в якій викладено деякі проблеми з нейронними мережами, в тому числі той факт, що комп'ютери в день були занадто обмеженими в своїй обчислювальній потужності для обробки даних, необхідних для функціонування нейронних мереж. Багато хто вважає, що ця книга привела до тривалої «зими», в якій припинилися дослідження в нейронних мережах. Дослідження нейронної мережі сповільнилися до тих пір, поки комп'ютери не досягли набагато більшої обчислювальної потужності [3].

Імпульсом для відновлення інтересу до нейронних мереж і навчання був алгоритм зворотного поширення Werbos (1975), який ефективно вирішував проблему, роблячи тренування багаторівневих мереж здійсненним і ефективним. Зворотне поширення розподіляло помилку назад вгору через шари, шляхом зміни вагових коефіцієнтів в кожному вузлі [4].

В середині 1980-х років паралельна розподілена обробка стала популярною під з'єднанням. Румельхарт і Маккеланд (1986) описали використання з'єднань для імітації нейронних процесів [5].

Підтримка векторних машин і інших, набагато більш простих методів, таких як лінійні класифікатори, поступово обійшли нейронні мережі в популярності

машинного навчання. Однак використання нейронних мереж трансформувало деякі домени, такі як прогноз білкових структур.

У 1992 році було введено максимальне об'єднання, щоб допомогти з мінімальною здвиговою інваріантністю до деформації, щоб допомогти в розпізнаванні 3D-об'єктів. У 2010 році навчання зворотного поширення з допомогою максимального об'єднання прискорилося за допомогою графічних процесорів і показало, що вони працюють краще, ніж інші варіанти об'єднання.

У 2006 році Хінтон запропонував вивчити уявлення високого рівня з використанням послідовних шарів двійкових або речових прихованих змінних з обмеженою машиною Больцмана для моделювання кожного шару. Після того, як буде вивчено досить багато шарів, глибока архітектура може бути використана в якості генеративної моделі шляхом відтворення даних при вибірці моделі з активацій функції верхнього рівня.

В 2010 році дослідження знову піднялися [6]. Велика динаміка даних, коли компанії збирають величезні масиви даних, а паралельні обчислення дають вченим дані навчання і обчислювальні ресурси, необхідні для запуску складних штучних нейронних мереж.

У 2012 році Ng і Dean створили мережу, яка навчилася розпізнавати концепції більш високого рівня тільки від перегляду немаркованих зображень, знятих з відео YouTube. Нейронна мережа змогла перевершити людську продуктивність при вирішенні задач розпізнавання зображень в рамках конкурсу ImageNet. Ранні проблеми в навчанні глибоких нейронних мереж були успішно вирішені за допомогою таких методів, як неконтрольована попередня підготовка, в той час як доступна обчислювальна потужність збільшувалася за рахунок використання графічних процесорів і розподілених обчислень. Нейронні мережі були розгорнуті в великих масштабах, особливо в проблемах із зображенням і візуальним розпізнаванням. Це стало відомо як «глибоке навчання».

З тих пір інтерес до штучних нейронних мереж зростає, і технологія продовжує поліпшуватися.

Прототипом для створення нейронних мереж послужили, як це не дивно, біологічні нейронні мережі. У людському мозку є мільярди клітин, які називаються нейронами, що обробляють інформацію у вигляді електричних сигналів. Зовнішня інформація / стимули приймається дендритами нейрона, обробленими в тілі його клітини, перетворюється в вихід і проходить через аксон до наступного нейрона. Наступний нейрон може вибрати або прийняти його, або відхилити залежно від сили сигналу. Точка з'єднання дедрита і аксона називається синапсом. ШНМ є дуже спрощеним уявленням про те, як працює нейрон мозку [7].

ШНМ збирають свої знання, виявляючи шаблони і відносини в даних і навчаються через досвід, а не від програмування. ШНМ формується із сотень поодиноких одиниць, штучних нейронів або елементів обробки (ЕО), пов'язаних з коефіцієнтами (вагами), які складають нейронну структуру і організовані в слої. Сила нейронних обчислень виходить від підключення нейронів в мережі. Кожен ЕО має зважені входи, функцію передачі і один вихід.

Нейронна мережа може містити наступні 3 шари:

1) рівень введення;
2) прихований шар;
3) вихідний рівень.

Мета вхідного шару – отримати в якості вхідних значень пояснюючі атрибути для кожного спостереження. Як правило, кількість вхідних вузлів у вхідному шарі дорівнює кількості пояснюючих змінних. «Вхідний шар» представляє шаблони мережі, які сполучаються з одним або декількома «прихованими шарами».

Вузли вхідного шару є пасивними, тобто вони не змінюють дані. Вони отримують єдине значення на своєму вході і дублюють значення на свої численні виходи. На рівні введення він дублює кожне значення і відправляється на всі приховані вузли.

Приховані шари застосовують дані перетворення до вхідних значень всередині мережі. У цьому, вхідні дуги, які йдуть від інших прихованих вузлів або

від вхідних вузлів, пов'язаних з кожним вузлом. Він з'єднується з вихідними дугами для виведення вузлів або до інших прихованих вузлів. У прихованому шарі фактична обробка виконується через систему зважених «з'єднань». Може бути один або кілька прихованих шарів. Значення, що входять в прихований вузол, помножені на ваги, набір заданих чисел, що зберігаються в програмі. Потім зважені входи додаються для створення одного номера.

Потім приховані шари посилаються на «вихідний шар». Вихідний рівень отримує з'єднання від прихованих шарів або від вхідного рівня. Він повертає вихідне значення, відповідне передбаченню змінної відповіді. У задачах класифікації зазвичай є тільки один вихідний вузол. Активні вузли вихідного рівня об'єднуються і змінюють дані для отримання вихідних значень [8].

ШНМ спочатку проходить стадію навчання, де вона вчиться розпізнавати шаблони в даних, візуально, в усній або текстовій формі. Під час цієї контрольованої фази мережа порівнює свій фактичний результат, отриманий з тим, що він повинна була робити, тобто бажаний вихід. Різниця між обома результатами коригується за допомогою зворотного поширення. Це означає, що мережа працює в зворотному напрямку від вихідного блоку до вхідних пристроїв, щоб налаштувати вагу його з'єднань між блоками, поки різниця між фактичним і бажаним результатом не дасть найменшу можливу помилку.

Існують різні типи ШНМ. Залежно від людського нейрона і мережевих функцій мозку ШНМ виконує завдання аналогічним чином. Більшість з них будуть мати деяку схожість з більш складними біологічними аналогами і дуже ефективні по їх наміченим завданням, таким як, наприклад, сегментації та класифікації. Кожен тип ШНМ має свої специфічні варіанти використання і рівні складності.

Типи штучних нейронних мереж:

- багатошаровий перцептрон;
- згорткова нейронна мережа (ЗНМ);
- рекурсивна нейронна мережа (РНМ);
- рекурентна нейронна мережа (РекНМ);

– довгострокова пам'ять (ДП);

– модель послідовності до послідовності;

– дрібні нейронні мережі.

Багатошаровий персептрон має три або більше шарів. Він використовує нелінійну функцію активації (в основному гіперболічну дотичну або логістичну функцію), яка дозволяє класифікувати дані, які не є лінійно роздільними. Кожен вузол в шарі з'єднується з кожним вузлом на наступному рівні, роблячи мережу повністю підключеною. Наприклад, багаторівневі додатки для обробки персептрона – це розпізнавання мови і машинний переклад.

ЗНМ містить один або кілька згортувальних шарів, які об'єднані або повністю пов'язані і використовує варіації багатошарових перцептронів. Згорткові шари використовують операцію згортки для введення, який передає результат на наступний рівень.

ЗНМ демонструють видатні результати в додатках для зображення й мови. Юн Кім в згортувальних нейронних мережах для класифікації речень описує процес і результати завдань класифікації тексту з використанням ЗНМ [9]. Він являє модель, побудовану поверх word2vec, проводить серію експериментів з нею і тестує її на декількох тестах, демонструючи, що модель відмінно працює.

У текстовому розумінні від Scratch, Xiang Zhang і Yann LeCun, демонструють, що ЗНМ можуть досягти видатної продуктивності без знання слів, фраз, речень і будь-яких інших синтаксичних або семантичних структур щодо людської мови [10]. Семантичний аналіз [11], виявлення парафраз [12], розпізнавання мови [13] також є додатками ЗНМ.

РНМ являє собою тип глибокої нейронної мережі, утвореної шляхом рекурсивного застосування одного і того ж набору ваг за структурою, щоб зробити структуроване передбачення по вхідним структурам зі змінним розміром або скалярною величиною на ньому шляхом проходження даної структури в топологічному порядку [14].

РекНМ, на відміну від первинної нейронної мережі, являє собою варіант рекурсивної штучної нейронної мережі, в якій зв'язки між нейронами створюють

спрямований цикл. Це означає, що вихід залежить не тільки від поточних входів, а й від стану нейрона попереднього кроку. Ця пам'ять дозволяє користувачам вирішувати проблеми NLP, такі як розпізнавання рукописного введення або розпізнавання мови. У статті «Генерація природної мови, перефразування і підсумовування оглядів користувачів з повторюваними нейронними мережами» автори демонструють рекурентну модель нейронної мережі, яка може генерувати нові пропозиції та зведення документів [15].

Siwei Lai, Liheng Xu, Kang Liu і Jun Zhao створили рекурентну згорткову нейронну мережу для класифікації тексту без людино-орієнтованих функцій і описали її в періодичних згортувальних нейронних мережах для текстової класифікації. Їх модель порівнювалася з існуючими методами класифікації тексту, такими як Bag of Words, Bigrams + LR, SVM, LDA, ядра дерев, рекурсивна нейронна мережа та ЗНМ. Було показано, що їх модель перевершує традиційні методи для всіх використовуваних наборів даних [16].

ДП являє собою специфічну рекурентну архітектуру нейронної мережі, яка була розроблена для точного моделювання часових послідовностей і їх далекодіючих залежностей, ніж звичайні РНМ [17]. ДП не використовує функцію активації в своїх рекурентних компонентах, збережені значення не змінюються, і градієнт не прагне зникнути під час навчання. Зазвичай блоки ДП реалізовані в «блоках» з декількома одиницями.

У довгостроковій короткостроковій пам'яті рекурентні нейронні мережеві архітектури для масштабного акустичного моделювання Хасім Сак, Ендрю Стар і Франсуаза Бьюфайс показали, що глибокі архітектури ДП РНМ забезпечують найсучаснішу продуктивність для великомасштабного акустичного моделювання.

Зазвичай модель послідовності до послідовності складається з двох рекурентних нейронних мереж: кодера, який обробляє вхід і декодера, який виробляє вихід. Кодер і декодер можуть використовувати одні і ті ж або різні набори параметрів.

Моделі послідовності до послідовності в основному використовуються в автовідповідачах, чатах і машинних перекладах. Такі багатошарові осередки

успішно використовувалися в послідовних моделях для перекладу в послідовному навчанні послідовностям з дослідженнями нейронних мереж [18].

У режимі виявлення парафраз з використанням рекурсивного автокодера представлена нова рекурсивна архітектура автокодера.

Крім глибоких нейронних мереж, дрібні моделі також є популярними і є корисними інструментами. Наприклад, word2vec є групою неглибоких двошарових моделей, які використовуються для створення вкладень слів. Представлений в ефективному оцінюванні уявлень слів у векторному просторі, word2vec приймає велику частину тексту в якості свого введення і створює векторний простір [19]. Кожне слово в корпусі отримує відповідний вектор в цьому просторі. Відмінною особливістю є те, що слова із загальних контекстів в корпусі розташовані близько один до одного в векторному просторі.

Одним із найбільш популярних типів ШНМ є ЗНМ. Цей специфічний тип алгоритму нейронної мережі використовується в багатьох з найбільш передових додатків, включаючи розпізнавання осіб, оцифровку тексту і обробку природної мови, класифікацію зображень тощо.

1.2 Особливості функціонування загорткових нейронних мереж

Згорткові нейронні мережі (ЗНМ) являють собою спеціалізовану нейронну мережу для обробки даних, яка має відому топологію у вигляді сітки. Згорткові мережі надзвичайно успішні в практичних застосуваннях. Перш за все вони використовуються для класифікації зображень, ідентифікації особи, знака, медичних діагнозів (наприклад пухлини), виконують оптичне розпізнавання символів (ОРС) та лікування для людей з ослабленим зором.[20]

Назва «згорткова нейронна мережа» вказує, що в мережі використовується математична операція, яка називається згорткою. Згортка - це спеціалізований вид лінійної операції. Згорткові мережі - це просто нейронні мережі, які

використовують згортку замість загального матричного множення, по крайній мірі, в одному зі своїх шарів [21].

ЗНМ містить один або кілька згорткових шарів, які об'єднані або повністю пов'язані і використовує варіації багатошарових перцептронів. Згорткові шари використовують операцію згортки для введення, передають результат на наступний рівень. На рисунку 1.2 наведено типову структуру ЗНМ.

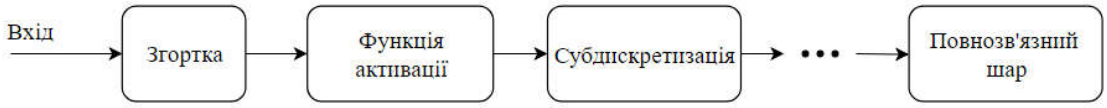


Рисунок 1.2 – Типова структура ЗНМ

У кожній ЗНМ є чотири основних блоки: рівень згортки, нелінійність (активація ReLU), шар об’єднання і повнозв’язний шар [22].

У згортковому шарі ми витягуємо функції з вхідного зображення:

Ми зберігаємо просторовий взаємозв'язок між пікселями, вивчаючи функції зображення, використовуючи малі квадрати вхідних даних. Ці квадрати вхідних даних також називаються фільтрами або ядрами.

Матриця, сформована шляхом зсуву фільтра по зображенню і обчислення точкового продукту, називається Feature. Чим більше фільтрів у нас є, тим більше функцій зображення витягується і тим краще наша мережа розпізнає шаблони в невидимих зображеннях.

Для того щоб будь-яка нейронна мережа була потужною, вона повинна містити нелінійність. LeNet використовує Sigmoid Non-Linearity, який приймає дійсне число і пригнічує його в діапазоні від нуля до одного. Зокрема, великі негативні числа стають нулями, а великі позитивні числа стають рівними одиниці.

Більш потужна нелінійна операція – ReLU [23]. Це елементна мудра операція, яка замінює всі негативні значення пікселів в таблиці функцій на нулі. Ми передаємо результат з шару згортки через функцію активації ReLU. Майже всі архітектури на основі ЗНМ, розроблені пізніше, використовували ReLU.

Пізніше виконується операція об'єднання, щоб зменшити розмірність кожної карти об'єктів. Це дозволяє скоротити кількість параметрів і обчислень в мережі, тим самим контролюючи перенавчання.

ЗНМ використовує max-pooling, в якому він визначає просторову околицю і бере найбільший елемент з випрямленої карти об'єктів всередині цього вікна. Після шару об'єднання наша мережа стає інваріантною до малих перетворень, перекручувань і перекладів у вхідному зображенні.

Після згортки і шарів об'єднання ми додаємо декілька повністю пов'язаних шарів для завершення архітектури ЗНМ. Вихідні дані з шарів згортки і об’єднання являють собою високорівневі функції вхідного зображення.

З більшою картиною архітектура ЗНМ виконує дві основні задачі: витяг ознак (рівні згортки і об'єднання) і класифікація (повністю пов'язані шари). Чим більше є кроків згортки, тим більш складніші функції мережа зможе навчитися розпізнавати.

На рисунку 1.3 наведена послідовність кроків при класифікації зображень за допомогою ЗНМ.

Рисунок 1.3 – Класифікація зображення з використанням ЗНМ

У дуже простому поясненні процес згортки працює наступним чином:

– спочатку ЗНМ використовує змінний віконний пошук, щоб розбити зображення на фрагменти;

– потім подає кожен фрагмент зображення в невелику нейронну мережу, використовуючи ті ж ваги для кожної плитки;

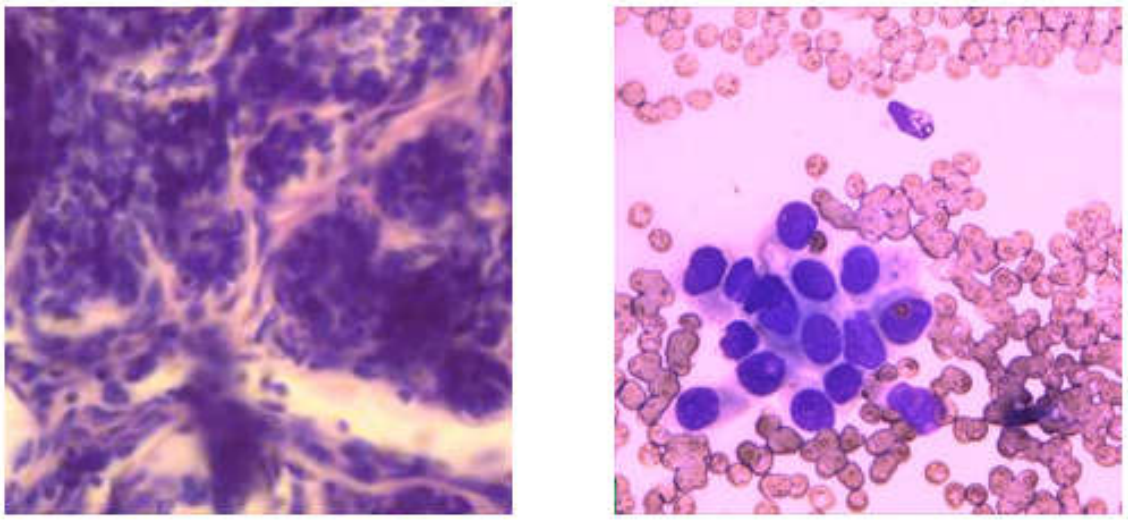
– далі зберігає результати від кожного фрагмента в новий вихідний масив;

– після цього ЗНМ субдискретизує масив, щоб зменшити його розмір;

– потім ЗНМ аналізує, чи є зображення збігом чи ні.

Таким чином ЗНМ класифікує і біомедичні зображення за класом. Саме згорткові нейронні мережі використовуються для класифікації гістологічних і цитологічних зображень.

На рисунку 1.4 зображено відмінність гістологічного зображення від цитологічного. Гістологічне зображення складніше за структурою ніж цитологічне, воно нижче по якості та чіткості.



а) б)

Рисунок 1.4 – Біомедичні зображення:
а) гістологічне; б) цитологічне

Гістологічне дослідження – це метод детального вивчення організму людини з метою виявлення патологічного процесу на тканинному рівні. Цитологія це наука, яка вивчає клітини, їх будову, функціонування, процеси розмноження, старіння і смерті [24].

1.3 Методи паралелізації штучних нейронних мереж

У типових нейронних мережах є мільйони параметрів, які визначають модель і вимагають більших даних для вивчення цих параметрів. Це обчислювально інтенсивний процес, який займає багато часу. Як правило, це займає декілька днів для навчання глибокої нейронної мережі. Інколи набір даних занадто великий для зберігання на одній машині.

Тому важливо придумати паралельні та розподілені алгоритми, які можуть працювати набагато швидше і які можуть значно скоротити час навчання [25].

Паралельна обробка являє собою об'єднання декількох процесорів для вирішення завдання. Це дозволяє зменшити час, який витрачається на обчислення, якщо порівнювати результати використання паралельних процесів і результати, що надаються одним процесором. Це дозволяє вирішувати об'ємні завдання і завдання фіксованої розмірності набагато швидше. Якщо говорити про можливості обробки інформації людським мозком, він так само використовує метод паралельної обробки, так як містить в собі величезну кількість інформації та взаємопов’язаних обробляючих елементів.

Модель штучної нейронної мережі містить в собі кілька паралельних структур. Їх використовують для поліпшення ефективності реалізації на паралельних архітектурах. Існують кілька типів рівнів паралелізації в штучних нейронних мережах.

Так Нордстремом були виділені основні рівні, де проводиться паралелізація:

- рівень фази навчання;
- рівень навчальної вибірки;
- рівень шару;
- рівень нейрона;
- рівень ваг [26], [27].

Всі ці рівні паралелізації знаходяться в нейроній мережі і можуть використовуватися одночасно. Для того, щоб вибрати необхідний рівень необхідно відштовхуватися від кількості нейронів в мережі, процесорів, особливостей комп'ютерної архітектури ШНМ і певного завдання, поставленого перед нею.

При навчанні штучної нейронної мережі відбувається повне її навчання по всій мережі. Для того, щоб визначити необхідні параметри для вирішення певної задачі, навчання складається з безлічі сеансів експериментального підбору. Наприклад, підбираються параметри необхідної кількості нейронів у всіх шарах нейронної мережі, швидкості обробки даних (навчання).

Навчання – це ітеративний процес, в якому дані навчання вводяться в модель партіями, прогнози виробляються поточною моделлю в прямому проході, обчислюються помилки між прогнозами і мітками основний істини помилки передаються через мережу, обчислюються коригування параметрів для всіх нейронів і параметри оновлюються для зведення до мінімуму помилок. Один цикл у всьому навчальному наборі – це епоха, і, як правило, для конвергенції необхідні кілька епох [28].

Навчання штучних нейронних мереж для великих наборів даних – трудомістке завдання.

Використовуючи метод паралельної обробки даних при фазі навчання в штучних нейронних мережах різні конфігурації мережі можуть досліджуватися в один і той же час [29]. Так, на рисунку 1.5 представлений паралелізм тренувальної сесії, де різниця між фазами навчання представлена у вигляді швидкості навчання.

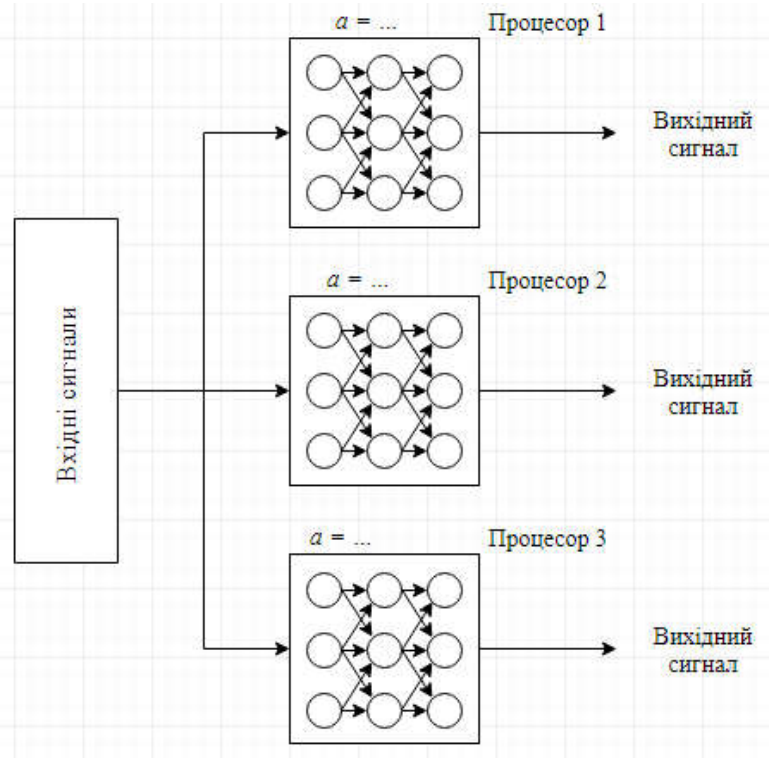


Рисунок 1.5 – Схема паралелізації фази навчання

При стадії підбору параметрів необхідно враховувати, що результати, отримані під час навчання залежать від початкових значень ваг. Так, при застосуванні алгоритму зворотного поширення помилки з градієнтною пошуковою методикою результати навчання дуже чутливі до первинних значень ваг.

Аналогічна ситуація зустрічається і в мережах Хопфілда [30] при вирішенні задач оптимізації. При використанні ж методу паралельної обробки даних при фазі навчання можна проводити дослідження мережі при різних первинних установках. Так само при використанні паралелізації навчання серед плюсів виділяється лінійний приріст швидкодії фази. Це можливо завдяки відсутності необхідності у взаємодії між процесорами.

При використанні паралелізації на рівні навчання вибірки використовують навчання одночасно на декількох різних навчальних вибірках. Це важливо, тому що часто потрібно досить велику кількість навчальних векторів в штучній нейронній мережі для вирішення певної задачі великого розміру [31].

На рисунку 1.6 представлена схема паралелізації навчальної вибірки.

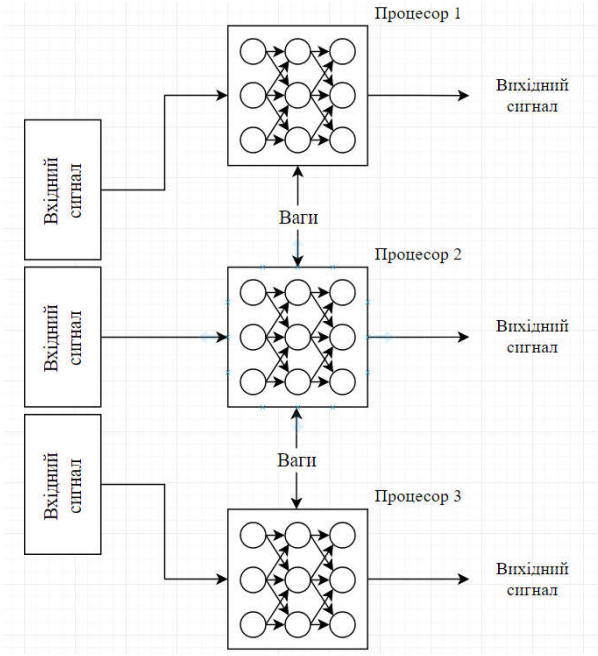


Рисунок 1.6 – Схема паралелізації навчальної вибірки

При системі, що використовує один потік векторів, вони будуть направлені в мережу послідовно, почергово, а при паралельній системі – навчальні вектори поділяються між процесорами, де кожен з процесорів має копію всіх даних штучної нейронної мережі. Тобто кожен процесор навчається на декількох вибірках.

При паралелізації рівнів шарів, використовуваних в моделях неокогнітрона і в моделях зі зворотним поширенням помилки вектори, що знаходяться на стадії навчання йдуть один за одним по мережі і знаходяться в мережі в один час [32]. У той же час канали деяких мереж нейронної мережі можуть йти по зворотному напрямку. У моделі зі зворотним поширенням помилки, вони можуть переходити і на зворотні шари назад.

Це дозволяє передати операцію іншому процесору. Такий же принцип може бути і в моделях штучних нейронних мереж, які не мають верств, але вектори так само йдуть через процесори за принципом конвеєра. Найточніші вектори, що пройшли через всі процесори, беруться за еталон. Після їх повторного переходу по процесорах мережі, процесори оновлюють ваги відповідно до цих стандартів. На рисунку 1.7 представлена дана схема роботи.

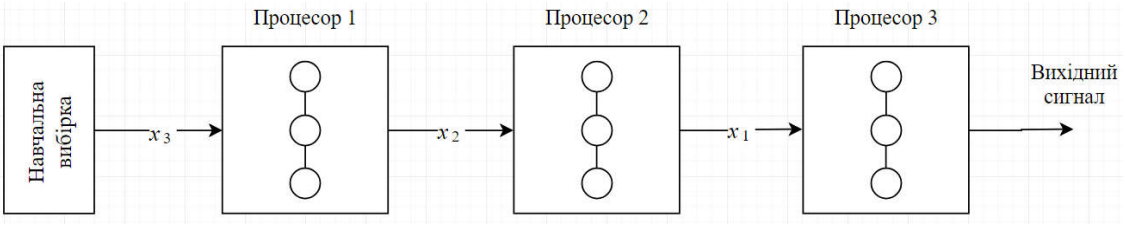


Рисунок 1.7 – Схема паралелізації на рівні шару

При використанні методу паралелізації на рівні нейронів, нейрон за функціями аналогічний процесору, так як є також обробляючим елементом. Паралельна обробка як процес на даному рівні розділяє нейрони в рамках одного шару по процесорах, а також при паралельних обчисленнях. В рамках даної системи обробки нейрон або деяка їх кількість співвідноситься з кожним процесором. Така модель паралелізації є в кожної з моделей штучних нейронних мереж і є одним з популярних методів. Схема роботи даного методу зображена на рисунку 1.8.

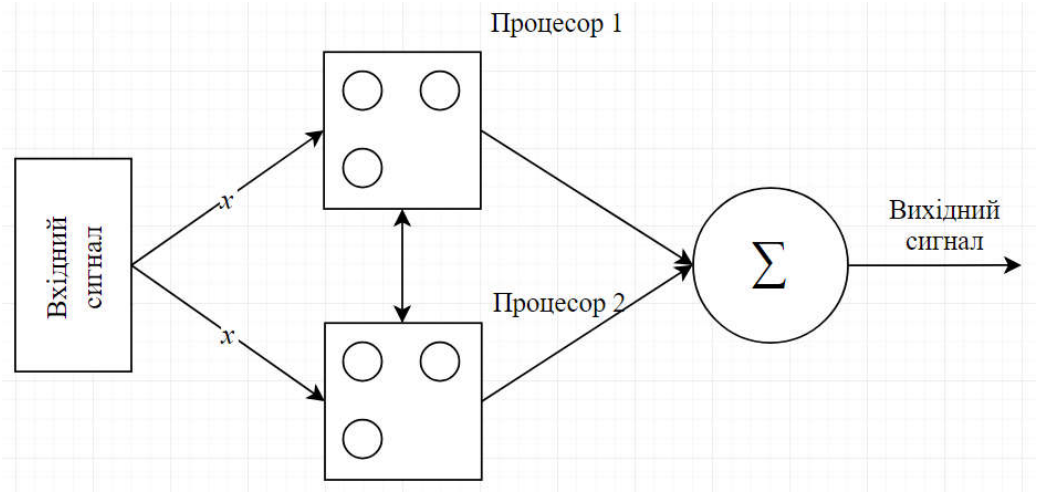


Рисунок 1.8 – Схема паралелізації на рівні нейрону [18]

Останнім методом паралелізації є – паралелізації на рівні ваг. Схема такого типу паралелізації наведена на рисунку 1.9.

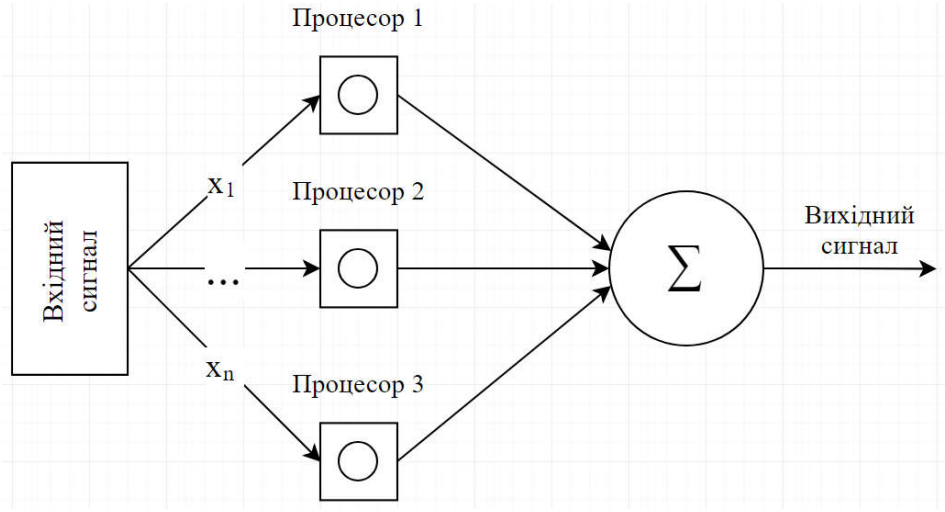


Рисунок 1.9 – Схема паралелізації на рівні ваг

Рівень ваг при методі паралелізації є дрібномодульним. Найчастіше даний метод використовують в апаратних реалізаціях. У даному методі обчислення в рамках одного нейрона розділені по декількох процесорах.

Для визначення ступеня успішності застосування вищевказаних методів паралелізації необхідно виробити систему релевантних показників.

1.4 Ефективність процесу парапаралелізації

Для оцінки успішності паралельного алгоритму використовують такі релевантні показники продуктивності як ефективність, збільшення швидкості (приріст), масштабованість системи. При цьому порівнюється ефективність при використанні методу паралельної обробки і при звичайній послідовній роботі. При порівнянні повинен бути знайдений найбільш швидкий алгоритм для вирішення проблеми при використанні послідовної обробки, в такому випадку будуть помітні переваги використання паралельного методу. Також необхідно мати на увазі, що може бути кілька паралельних алгоритмів, проте не всі найбільш швидкі алгоритми можуть бути придатними для процесу паралелізації.

Для оцінки швидкості роботи алгоритму порівняємо час найшвидшого з алгоритмів розв'язання певної проблеми при послідовній системі обробки даних і час, витрачений паралельним алгоритмом на певному числі процесорів для вирішення тієї ж проблеми. Якщо швидкість виконання паралельного алгоритму збільшилася в стільки ж разів, скільки процесорів було задіяно при розрахунку, то ми маємо справу з лінійним зростанням. Якщо швидкість виконання збільшилася сильніше, то алгоритм суперлінійний, якщо швидкість зросла менше, ніж при лінійному зростанні, то алгоритм сублінійний. Так само потрібно враховувати, що тут при використанні паралельного алгоритму оброблюючий елемент рівнозначний послідовному алгоритму.

Ефективністю процесу паралелізації в даному випадку є ступінь задіяності процесорів, які використовуються при паралельному алгоритмі обробки.

Час необхідний для навчання на багатопоточній системі можна виразити формулою:

$$\tau = \frac{t}{n} + \varphi,$$

де t – час навчання на однопоточній системі,

n – кількість потоків.

Таким чином, існує деяке φ яке відображає різницю між очікуваним часовим виграшем(в n разів) і реальним. φ – складається з часу передачі даних, часу на підготовку даних, часу очікування інших вузлів, побічних часових затрат. Наявність даних величин зумовлено тим, що паралельні алгоритми відрізняються від послідовних.

Величина φ – показує лиже абсолютну різницю між однопотоковим і багатопотоковими часовими затратами, що не дозволяє адекватно оцінити паралельну версію того чи іншого методу навчання, тому необхідно використовувати величину

$$\alpha = \frac{t}{n\tau}, \qquad (1.1)$$

де α – коефіцієнт ефективності паралельного алгоритму.

Ця величини показує відносний виграш у часі, та становить (0.7; 1) для ефективних алгоритмів.

При ідеальному випадку з лінійним збільшенням швидкості роботи ефективність буде рівнозначна одиниці. Але на ділі в більшості випадків при реалізації паралельного алгоритму обробки буває сублінейний приріст швидкості.

Показник масштабованості системи показує здатність системи впливати на прискорення обробки пропорційно кількості процесорів. Чим більше процесорів використовується, тим більше падає ефективність при фіксованому розмірі завдання. За підтримки ефективності системи шляхом збільшення розмірності задачі, система паралелізації буде масштабується.

При збільшенні числа процесорів до конкретного завдання з'являється проблема збалансованості навантаження. При постійному розмірі завдання навантаження на процесори зменшується. Щоб не було проблеми недостачі роботи процесорів, використовують масштабованість завдання.

При збільшенні числа процесорів ефективність падає. Так Амдал встановив верхній поріг приросту продуктивності, якого можливо досягти при використанні методу паралелізації. За його теорією, на практиці обчислення можуть виконуватися послідовно, не паралельно і, власне, паралельно. Так як кількість процесорів збільшується, час роботи, витрачений на паралельні обчислення зменшується. При цьому послідовна частина роботи над обчисленнями залишається постійною.

Час, витрачений на реалізацію послідовних обчислень, що не масштабується при збільшенні числа процесорів і швидкісний лінійний приріст досягається тільки при відсутності послідовної частини під час вирішення певної задачі. Але так як більшість завдань все ж містять в собі послідовні обчислення, максимальний приріст швидкості обмежується.

1.5 Засоби та технології паралелізації

Ультрасучасні нейронні мережі можуть мати від мільйонів до понад мільярд параметрів, які можна налаштувати за допомогою зворотного поширення. Їм також потрібен великий обсяг навчальних даних для досягнення високої точності. Величезний розмір цих мереж може являти собою складне обчислювальне навантаження навіть для сучасних центральних процесорів (CPU) [33]. З цієї причини для навчання і роботи таких мереж зазвичай використовуються графічні процесори (GPU) [34]. Оскільки нейронні мережі створені з великого числа ідентичних нейронів, вони дуже паралельні за своєю природою. Цей паралелізм природним чином відображається на GPU, які забезпечують значне прискорення обчислень в порівнянні з навчанням тільки на процесорі.

GPU стали кращою платформою для навчання великих і складних систем на основі нейронних мереж завдяки їх здатності прискорювати роботу систем.

В GPU можна виділити такі рівні паралелізму, як: суперскалярність процесорного елемента (ПЕ), векторність, множинність ПЕ і апаратних ниток, що запускаються на них, паралелізм роботи процесорних елементів і доступу до пам'яті, множинність графічних процесорів (з'явилися технології, що дозволяють поєднувати відеокарти в однорідні обчислювальні системи) [35].

Основні особливості традиційних обчислень загального призначення через графічний інтерфейс GPU полягають в наступному:

– у цих рамках топологія GPU як паралельної системи – двовимірна решітка. Це означає, що дані, що надходять на обробку в GPU і одержувані в результаті роботи програми на GPU, найкраще представляти в вигляді двовимірного масиву значень, розмір якого, однак, обмежений апаратним інтерфейсом;

– GPU дозволяють виробляти арифметичні операції обчислення елементарних функцій, виконувати інші операції над векторами даних. Ці

операції забезпечують одночасне виконання операцій над усіма елементами вектора;

– повільний обмін даними між відеокартою і CPU: необхідно по можливості зменшувати цей обсяг або збільшити відношення обсягу обчислень до обсягу даних, а також максимально повно використовувати кешування даних при завантаженні, так як доступ до кеш даних дає незначну в порівнянні з доступом до глобальної пам'яті відеокарти затримку;

– традиційна операція scatter – запис результатів обробки, виробленої на процесорному елементі, в пам'ять, доступну для інших потоків, – може призводити до значної втрати продуктивності (на відміну від операції gather – збору даних).

Графічні процесори відмінно підходять для глибокого навчання, тому що тип обчислень, для яких вони були розроблені, такий же, як і для глибокого навчання. Зображення, відео та інша графіка представлені у вигляді матриць, тому при виконанні будь-якої операції, наприклад, ефекту збільшення або повороту камери, все, що вам потрібно зробити, - це застосувати деякt математичне перетворення до матриці.

На практиці це означає, що графічні процесори більш спеціалізовані, ніж матричні процесори і деякі інші типи складних математичних перетворень в порівнянні з центральними процесорами (CPU). Це змушує алгоритми глибокого навчання працювати в кілька разів швидше на GPU у порівнянні з CPU. Час навчання часто може бути скорочено з декількох днів до декількох годин.

Бібліотека для роботи з глибокими нейронними мережами NVIDIA CUDA® (cuDNN) являє собою прискорену на GPU бібліотеку примітивів для глибоких нейронних мереж [36]. cuDNN надає добре налаштовані реалізації для стандартних підпрограм, таких як пряма і зворотна згортка, пулінг, нормалізація і рівні активації.

Дослідники і розробники фреймворків по всьому світу покладаються на cuDNN для високопродуктивного прискорення GPU. Це дозволяє їм зосередитися на навчанні нейронних мереж і розробці програмних додатків, а не витрачати час

на низькорівневе налаштування продуктивності графічного процесора. cuDNN прискорює широко використовувані фреймворки для глибокого навчання, включаючи Caffe, Caffe2, Chainer, Keras, MATLAB, MxNet, TensorFlow і PyTorch.

Модель програмування CUDA заснована на концепції ядра. Ядро – це функція, яка виконується кілька раз паралельно, кожен екземпляр працює в окремому нитка. Потоки організовані в одно-, дво- або тривимірні блоки, які в свою чергу організовані в одно- або двовимірні сітки. Блоки повністю незалежні один від одного і можуть бути виконані в будь-якому порядку. Потоки проте блок гарантовано буде працювати на одному багатопроцесорних. Сотні ядер можуть працювати разом тисячі потоки. Ядро може ділитися джерелами, включаючи пам'ять і записів. Серійний код повинен бути запущений в CPU, а паралельно код повинен бути запущений в GPU. Так як структури даних використовуються масиви, вектори і матриці, бібліотека CUBLAS була використана. CUBLAS це реалізація BLAS (підпрограми базової лінійної алгебри) на вершині CUDA і є автономним в API рівень, тому немає необхідності прямої взаємодії з CUDA. Це надає допоміжні функції для створення матричних і векторних об'єктів в пам'яті графічного процесора, заповнити їх даними, виконати BLAS операції і зачитати результати. CUDA раніше використовувався для паралельних реалізацій пікових нейронних мереж [37]. також було перевірено і є порівняно з реалізаціями в комп'ютерних кластерах працює OpenMP [38].

Одним з найбільш популярних засобів програмування для комп'ютерів із загальною пам'яттю, що базуються на традиційних мовах програмування і використання спеціальних коментарів, в даний час являється технологія OpenMP. За основу береться послідовна програма, а для створення її паралельної версії користувачеві надається набір директив, функцій і змінних оточення.

Передбачається, що створювана паралельна програма буде переноситься між різними комп’ютерами з пам'яттю, що підтримують OpenMP API [39].

Технологія OpenMP націлена на те, щоб користувач мав один варіант програми для паралельного і послідовного виконання. Однак є можливість створювати програми, які працюють коректно лише в паралельному режимі або

дають в послідовному режимі інший результат. Більш того, через накопичення помилок округлення результат обчислень із використанням користуванням різної кількості ниток може в деяких випадках відрізнятися.

Інтерфейс OpenMP задуманий як стандарт для програмування на SMPсистемах (SSMP, ccNUMA та інших) в моделі загальної пам'яті (shared memory model). У стандарт OpenMP входять специфікації набору директив компілятора, допоміжних функцій і змінних середовища. OpenMP реалізує паралельні обчислення за допомогою багатопоточності, в якій «головний» потік створює набір «підлеглих» потоків, і завдання розподіляється між ними. Передбачається, що потоки виконуються паралельно на машині з декількома процесорами, причому кількість процесорів не обов'язково має бути більше або дорівнювати кількості потоків.

POSIX-інтерфейс для організації ниток (Pthreads) підтримується практично на всіх UNIX-системах, однак з багатьох причин не підходить для практичного паралельного програмування: в ньому немає підтримки мови Фортран, занадто низький рівень програмування, немає підтримки паралелізму за даними, а сам механізм ниток спочатку розроблявся не для цілей організації паралелізму. OpenMP можна розглядати як високорівневу надбудову над Pthreads (або аналогічними бібліотеками ниток); в OpenMP використовується термінологія і модель програмування, близька до Pthreads, наприклад, динамічно породжувані нитки, загальні і роздільні дані, механізм «замків» для синхронізації.

Відповідно до термінології POSIX threads, будь який UNIX-процес складається з декількох ниток управління, які мають загальний адресний простір, але різні потоки команд і роздільні стеки. У найпростішому випадку процес складається з однієї нитки. Нитки іноді називають також потоками, легковаговими процесами, LWP (light-weight processes) [40].

Важливою перевагою технології OpenMP є можливість реаліції так званого інкрементального програмування, коли програміст поступово знаходить ділянки в програмі, що містять ресурс паралелізму, за допомогою наданих механізмів робить їх паралельними, а потім переходить до аналізу наступних ділянок. Таким

чином, в програмі не розпаралелених частин поступово стає все менше. Такий підхід значно полегшує процес адаптації послідовних програм до паралельних комп'ютерів, а також налагодження та оптимізацію.

Під час роботи над розділом було розглянуто основні поняття штучного інтелекту, штучних нейронних мереж, їх історію, види та галузі застосування. Детальніше опрацьовано згорткові нейронні мережі, що знайшли широке використання у різноманітних сферах, де об’єктом дослідження є зображення, насамперед, у медицині.

Після детального аналізу методів, засобів та технологій паралелізації нейронних мереж було вирішено будувати ЗНМ із використанням CUDA, оскільки дана технологія дозволяє використовувати графічні процесори, які при обробці графічної інформації показують набагато кращі результати ніж центральні процесори.

Для подальшої реалізації в програмному продукті було обрано метод паралельної обробки даних при фазі навчання в штучних нейронних мережах, оскільки різні конфігурації мережі можуть досліджуватися в один і той же час. Так само при використанні паралелізації навчання серед плюсів виділяється лінійний приріст швидкодії фази. Це можливо завдяки відсутності необхідності в взаємодії між процесорами.

2 АЛГОРИТМИ НАВЧАННЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

Найважливішою властивістю нейронних мереж є їх здатність навчатися на основі даних навколишнього середовища і в результаті навчання підвищувати свою продуктивність. Підвищення продуктивності відбувається з часом відповідно до певних правил. Навчання нейронної мережі відбувається за допомогою інтерактивного процесу коригування синаптичних ваг і порогів. В ідеальному випадку нейронна мережа отримує знання про навколишнє середовище на кожній ітерації процесу навчання.

Навчання – це процес, в якому вільні параметри нейронної мережі налаштовуються за допомогою моделювання середовища, в яку ця мережа вбудована [41]. Тип навчання визначається способом підстроювання цих параметрів.

Це визначення процесу навчання нейронної мережі передбачає наступну послідовність подій:

– у нейронну мережу надходять стимули із зовнішнього середовища;

– в результаті першого пункту змінюються вільні параметри нейронної мережі;

– після зміни внутрішньої структури нейронна мережа відповідає на порушення вже іншим чином.

Вищевказаний список чітких правил вирішення проблеми навчання нейронної мережі називається алгоритмом навчання [42]. Не існує універсального алгоритму навчання, відповідного для всіх архітектур нейронних мереж. Існує лише набір засобів, представлений безліччю алгоритмів навчання, кожен з яких має свої переваги.

Алгоритми навчання відрізняються один від одного способом налаштування синаптичних ваг нейронів. Ще однією відмінною характеристикою є спосіб зв'язку навченої нейронної мережі з зовнішнім світом. У цьому контексті говорять

про парадигму навчання, пов'язану з моделлю навколишнього середовища, в якому функціонує дана нейронна мережа.

Для того, щоб нейронна мережа була здатна виконати поставлене завдання, її необхідно навчити (рисунок 2.1).

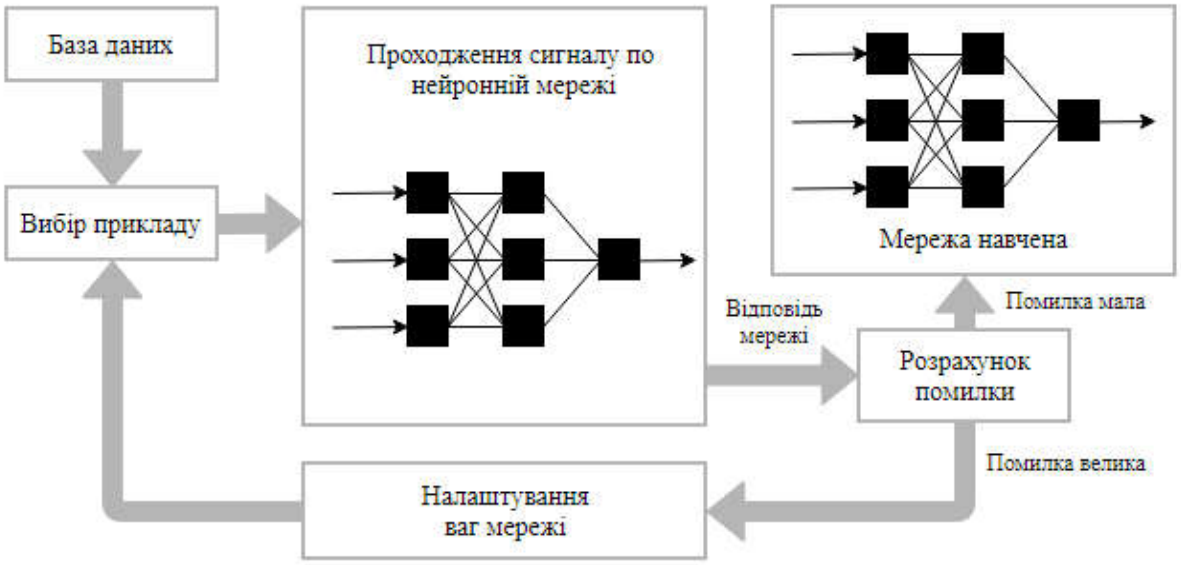


Рисунок 2.1 – Процес навчання ШНМ

Існують два концептуальних підходи до навчання нейронних мереж: навчання з учителем і навчання без учителя [43].

Процес навчання з учителем є пред'явленням мережі вибірки навчальних прикладів. Кожен зразок подається на входи мережі. Потім проходить обробку всередині структури НС, обчислюється вихідний сигнал мережі, який порівнюється з відповідним значенням цільового вектора, що представляє собою необхідний вихід мережі. Далі за певним правилом обчислюється помилка, і відбувається зміна вагових коефіцієнтів зв'язків всередині мережі в залежності від обраного алгоритму. Вектори навчальної множини пред'являються послідовно, обчислюються помилки і ваги підлаштовуються для кожного вектора до тих пір, поки помилка по всьому навчальному масиву не досягне прийнятного рівня.

Хоча метод навчання з вчителем успішно застосовується для вирішення прикладних задач, багато дослідників критикують його за біологічну

неправдоподібність. Дійсно, важко уявити, що в мозку людини є деякий механізм, який порівнює дійсні результати з бажаним.

Навчання нейронної мережі без вчителя є набагато більш правдоподібною моделлю навчання з точки зору біологічних коренів штучних нейронних мереж.

При навчанні без учителя навчальна множина складається лише з вхідних векторів. Навчальний алгоритм підлаштовує ваги мережі так, щоб виходили узгоджені вихідні вектори, тобто щоб пред'явлення досить близьких вхідних векторів давало однакові виходи. Процес навчання виділяє статистичні властивості навчальної множини і групує подібні вектори в класи. Пред'явлення на вхід вектора з даного класу дасть певний вихідний вектор, але до навчання неможливо передбачити, який вихід буде проводитися даним класом вхідних векторів. Отже, виходи подібної мережі повинні трансформуватися в деяку зрозумілу форму, зумовлену процесом навчання. Це не є серйозною проблемою. Зазвичай не складно ідентифікувати зв'язок між входом і виходом, встановлену мережею.

Нейронна мережа без навчання не готова до роботи. Тільки після багаторазової демонстрації прикладів знання поступово стабілізуються, системи дають правильні відповіді на запропоновані запитання. У подібних ситуаціях говорять про те, що проведено глибоке навчання. Нейронні мережі поступово знижують величину помилки. Коли її величина буде зведена до нуля, тренування припиняють. Утворену нейронну мережу вважають придатною для застосування на нових вихідних даних.

Отже, існує безліч алгоритмів навчання нейронних мереж. В даний час використовують кілька їх варіантів:

- сполучених градієнтів;
- зворотне поширення;
- квазі-Ньютонівський;
- псевдо-зворотний;
- навчання Кохонена;
- Левенберга-Марквардта;

– установка явних відхилень.

Це далеко не всі алгоритми навчання нейронних мереж, що застосовуються в даний час.

Розглянемо декілька із них, які є найбільш поширеними та найчастіше використовуються [44].

2.1 Алгоритм зворотного поширення

Зворотне поширення, скорочене від «зворотного поширення помилок», являє собою алгоритм для контрольованого навчання штучних нейронних мереж з використанням градієнтного спуску. З огляду на штучну нейронну мережу і функцію помилки, метод обчислює градієнт функції помилки по відношенню до ваги нейронної мережі. Це узагальнення дельта-правила для перцептронів на багатошарові нейронні мережі з прямим зв'язком [45].

Частина «назад» у назві виникає з того факту, що обчислення градієнта відбувається в зворотному напрямку через мережу, причому градієнт останнього шару ваг обчислюється першим, а градієнт першого шару ваг обчислюється останнім. Часткові обчислення градієнта з одного шару повторно використовуються при обчисленні градієнта для попереднього шару. Цей зворотний потік інформації про помилки дозволяє ефективно обчислювати градієнт на кожному рівні в порівнянні з наївним підходом обчислення градієнта для кожного шару окремо [46].

Зворотне поширення аналогічно обчисленню дельта-правила для багатошарової мережі з прямим зв'язком. Таким чином, як і дельта-правило, зворотне поширення вимагає трьох речей:

1) набір даних, що складається з пар введення-виведення $(\overrightarrow{x_i}, \overrightarrow{y_i})$, де $\overrightarrow{x_i}$ є входом і $\overrightarrow{y_i}$ є бажаним виходом мережі на вході $\overrightarrow{x_i}$. Безліч пар введення-виведення розміру N позначається $X = \{(\overrightarrow{x_1}, \overrightarrow{y_1}), \dots, (\overrightarrow{x_N}, \overrightarrow{y_N})\}$;

2) нейронна мережа з прямим зв'язком чиї параметри позначаються спільно θ. При зворотному поширенні параметри, що представляють першорядний інтерес є w_{ij}^k, вага між вузлом j в шарі l_k і вузлом i в шарі l_{k-1} і b_i^k зміщення для вузла i в шарі l_k. Немає ніяких з`єднань між вузлами в одному шарі, і шари повністю пов'язані;

3) функція помилки, $E(X, \theta)$, яка визначає помилку між бажаним виходом $\overrightarrow{y_l}$ і обчисленим виходом $\widehat{\overrightarrow{y_l}}$ нейронної мережі на вході $\overrightarrow{x_l}$ для набору пар вхід-вихід $(\overrightarrow{x_l}, \overrightarrow{y_l}) \in X$ і конкретного значення параметрів θ.

Навчання нейронної мережі з градієнтним спуском вимагає обчислення градієнта функції помилки $E(X, \theta)$ щодо ваг w_{ij}^k і зміщень b_i^k. Потім, згідно швидкості навчання α, кожна ітерація градієнтного спуску оновлює вагові коефіцієнти і зміщення, спільно позначаються (θ) відповідно до

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta},$$

де θ^t позначає параметри нейронної мережі на ітерації t в градієнтному спуску.

Висновок алгоритму зворотного поширення досить простий. Це випливає з використання правила ланцюга і правила творення в диференціальному обчисленні. Застосування цих правил залежить від диференціації функції активації, однією з причин, по якій не використовується важка крокова функція (переривчаста і, отже, недиференційована).

У решти цього розділу похідна функції $f(x)$ буде позначатися $f'(x)$, так що похідна сигмоїдної функції дорівнює $\sigma'(x)$.

Щоб ще більше спростити математику, зміщення b_i^k для вузла i в шарі k буде включено в вагові коефіцієнти, як w_{0i}^k при фіксованому виході $o_0^{k-1} = 1$ для вузла O в шарі $k - 1$. Таким чином,

$$w_{0i}^k = b_i^k.$$

Щоб побачити, що це еквівалентно оригінальному формулюванню, потрібно звернути увагу на те, що

$$a_i^k = b_i^k + \sum_{j=1}^{r_{k-1}} w_{ji}^k \, o_j^{k-1} = \sum_{j=1}^{r_{k-1}} w_{ji}^k \, o_j^{k-1},$$

де ліва сторона – оригінальне формулювання, а права сторона – нове формулювання.

Використовуючи позначення вище, зворотне поширення намагається мінімізувати таку функцію помилки по відношенню до ваг нейронної мережі:

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (\widehat{y_i} - y_i)^2,$$

де для кожної ваги w_{ij}^k обчислюється значення $\frac{\partial E}{\partial w_{ij}^k}$.

Оскільки функція помилки може бути розкладена на суму по окремим термінам помилок для кожної окремої пари введення-виведення, похідна може бути розрахована по кожній парі введення-виведення окремо і потім об'єднана в кінці (оскільки похідна від суми функцій - сума похідних кожної функції):

$$\frac{\partial E(X, \theta)}{\partial w_{ij}^k} = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial}{\partial w_{ij}^k} \left(\frac{1}{2} (\widehat{y_d} - y_d)^2 \right) = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial E_d}{\partial w_{ij}^k}.$$

Таким чином, для цілей деривації алгоритм зворотного поширення буде стосуватися тільки однієї пари вхід-вихід. Як тільки це отримано, загальна форма для всіх пар введення-виведення в X може бути згенерована шляхом об'єднання окремих градієнтів. Таким чином, розглянута функція помилок для диференціювання

$$E = \frac{1}{2}(\hat{y} - y)^2,$$

де індекс d в E_d, $\widehat{y_d}$ і y_d опущений для спрощення.

Диференціювання алгоритму зворотного поширення починається з застосування правила ланцюжка до окремої похідної функції помилки

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ij}^k},$$

де a_j^k – активація (творення-сума плюс зсув) вузла j в шарі k перед його передачею в нелінійну функцію активації (в даному випадку, в сигмоїдну функцію) для генерації вихідних даних.

Таке розкладання окремої похідної в основному говорить про те, що зміна функції помилки через вагу є продуктом зміни функції помилки E через час активації a_j^k, через зміну активації a_j^k через вагу w_{ij}^k.

Перший член зазвичай називають помилкою з причин, що обговорюються нижче. Позначається

$$\delta_j^k \equiv \frac{\partial E}{\partial a_j^k}.$$

Другий член може бути розрахований з рівняння для a_j^k вище:

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left(\sum_{l=0}^{r_{k-1}} w_{li}^k\, o_l^{k-1}\right) = o_i^{k-1}.$$

Таким чином, окрема похідна функції помилки E по вазі w_{ij}^k є

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1}.$$

Таким чином, часткова похідна ваги є продуктом члена помилки δ_j^k в вузлі j в шарі k і виходом o_i^{k-1} вузла i в шарі $k-1$. Це має інтуїтивний сенс, так як вага w_{ij}^k з`єднує вихід вузла i в шарі $k-1$ з входом вузла j в шарі k в графі обчислень.

Важливо відзначити, що всі наведені вище часткові похідні були розраховані без урахування будь-якої конкретної функції помилки або функції активації. Однак, оскільки член помилки δ_j^k все ще повинен бути обчислений і залежить від функції помилки E, на цьому етапі необхідно ввести конкретні функції для обох з них. Як згадувалося раніше, класичне зворотне поширення використовує функцію середньоквадратичної помилки (яка є квадратичною функцією помилки для випадку одиночної пари вхід-вихід) і функцію активації сигмоїда.

Буде показано, що обчислення помилки δ_j^k залежить від значень членів помилки в наступному шарі. Таким чином, обчислення умов помилки триватиме в зворотному напрямку від вихідного рівня до вхідного рівня. Це місце, де зворотне поширення або зворотне поширення помилок отримує своє ім'я.

Починаючи з останнього рівня, зворотне поширення намагається визначити значення δ_i^m, де m є останнім шаром (індекс рівний 1, а не j тому, що цей диференціювання відноситься до нейронної мережі з одним виходом, тому існує тільки один вихідний вузол $j=1$). Наприклад, нейромережа з чотирма шарами буде мати $m=3$ для останнього шару, $m=2$ для другого шару і так далі. Вираження функції помилки E через значення a_1^m (так як δ_1^m є частковою похідною по відношенню до a_1^m) дає

$$E=\frac{1}{2}(\hat{y}-y)^2=\frac{1}{2}(g_0(a_1^m)-y)^2,$$

де $g_0(x)$ – функція активації для вихідного шару.

Таким чином, застосування похідної та використання правила творення дає

$$\delta_1^m = (g_0(a_1^m) - y)g_0'(a_1^m) = (\hat{y} - y)g_0'(a_1^m).$$

Якщо скласти все разом, то часткова похідна функції помилки E по вазі в останньому шарі w_{i1}^m дорівнює:

$$\frac{\partial E}{\partial w_{i1}^m} = \delta_1^m o_i^{m-1} = (\hat{y} - y)g_0'(a_1^m)o_i^{m-1}.$$

Тепер виникає питання, як розрахувати часткові похідні шарів, відмінних від вихідного шару. Потрібно дотримуватись наступного рівняння для члена помилки δ_j^k в шарі $1 \leq k < m$:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k},$$

де l варіюється від 1 до r^{k+1} (кількість вузлів в наступному шарі).

Зверніть увагу, що оскільки вхідне значення зсуву o_0^k аналогічно до w_{0j}^{k+1} є фіксованим, його значення не залежить від вихідних даних попередніх шарів і l, таким чином, не приймає значення 0.

Підключення терміна помилки δ_l^{k+1} дає наступне рівняння:

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k}.$$

Згадуючи визначення a_l^{k+1}

$$a_l^{k+1} = \sum_{j=1}^{r^{k+1}} w_{jl}^{k+1} g(a_j^k),$$

де $g(x)$ є функцією активації для прихованих шарів,

$$\frac{\partial a_l^{k+1}}{\partial a_j^k} = w_{jl}^{k+1} g'(a_j^k).$$

Включення цього в вищенаведене рівняння дає остаточне рівняння для члена помилки δ_j^k в прихованих шарах, яке називається формулою зворотного поширення:

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1} g'(a_j^k) = g'(a_j^k) \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

Якщо скласти все разом, то часткова похідна функції помилки E по вазі в прихованих шарах w_{ij}^k для $1 \leq k < m$ є

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} = g'(a_j^k) o_i^{k-1} \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

Алгоритм зворотного поширення залежить від наступних п'яти рівнянь:

– для часткових похідних:

$$\frac{\partial E_d}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1}; \quad (2.1)$$

– для терміна помилки останнього шару:

$$\delta_1^m = g_0'(a_1^m)(\widehat{y_d} - y_d); \quad (2.2)$$

– для помилки прихованих шарів:

$$\delta_j^k = g'(a_j^k) \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}; \quad (2.3)$$

– для об'єднання часткових похідних для кожної пари введення-виведення:

$$\frac{\partial E(X,\theta)}{\partial w_{ij}^k} = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial}{\partial w_{ij}^k}\left(\frac{1}{2}(\widehat{y_d} - y_d)^2\right) = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial E_d}{\partial w_{ij}^k}; \quad (2.4)$$

– для поновлення ваг:

$$\Delta w_{ij}^k = -\alpha\frac{\partial E(X,\theta)}{\partial w_{ij}^k}. \quad (2.5)$$

Алгоритм зворотного поширення виконується в наступних кроках, припускаючи відповідну швидкість навчання і випадкову ініціалізацію параметрів w_{ij}^k:

1) розрахуйте пряму фазу для кожної пари введення-виведення $(\overrightarrow{x_d}, y_d)$, і збережіть результати $\widehat{y_d}$, a_j^k, і o_j^k для кожного вузла j в шарі k, переходячи від шару 0, вхідного шару, до шару m, вихідного шару;

2) розрахуйте зворотну фазу для кожної пари введення-виведення $(\overrightarrow{x_d}, y_d)$ і збережіть результати $\frac{\partial E_d}{\partial w_{ij}^k}$ для кожного вагового коефіцієнта w_{ij}^k, який зв'язує вузол i в шарі $k-1$ до вузла j в шарі k, переходячи від шару m, вихідного шару до шару 1, вхідного шару:

а) оцініть похибку останнього шару δ_1^m,використовуючи рівняння 2.2;

б) розповсюдьте умови помилки для прихованих шарів δ_j^k, працюючи в зворотному напрямку від останнього прихованого шару $k = m-1$, багаторазово використовуючи рівняння 2.3;

в) оцініть часткові похідні окремої помилки E_d по відношенню до w_{ij}^k, використовуючи рівняння 2.1;

3) об'єднайте окремі градієнти для кожної пари введення-виведення $\frac{\partial E_d}{\partial w_{ij}^k}$, щоб отримати загальний градієнт $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ для всього набору пар введення-

виведення $X = \{(\overrightarrow{x_1}, y_1), \ldots, (\overrightarrow{x_N}, y_N)\}$, використовуючи рівняння 2.4 (просте середнє значення окремих градієнтів);

4) оновлення ваги відповідно до швидкості навчання α і загальним градієнтом $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$, використовуючи рівняння 2.5 (переміщення в напрямку негативного градієнта).

Існує два режими реалізації методу зворотного поширення помилки:

– стохастичного (stochastic) градієнтного спуску;

– пакетного (batch) градієнтного спуску.

2.2 Градієнтний спуск

Градієнтний спуск, також відомий як найшвидший спуск, є найпростішим алгоритмом навчання [47]. Він вимагає інформації від вектора градієнта, тому вважається методом першого порядку.

Позначимо $f(w_i) = f_i$ та $\nabla f(w_i) = g_i$. Метод починається в точці w_0 і, поки критерій зупинки не буде задоволений, переміщається від w_i до w_{i+1} в напрямку навчання $d_i = -g_i$. Тому метод градієнтного спуску повторюється наступним чином:

$$w_{i+1} = w_i - g_i\eta_i, \qquad i = 0,1, \ldots$$

Параметр η є тренувальною швидкістю. Це значення може бути встановлено на фіксоване значення або знайдено шляхом одновимірної оптимізації вздовж напрямку навчання на кожному кроці. Оптимальне значення для швидкості навчання, одержуваної шляхом мінімізації лінії на кожному наступному кроці, зазвичай найбільш прийнятне. Тим не менш, є ще багато програмних інструментів, які використовують тільки фіксоване значення для швидкості навчання.

На рисунку 2.2 зображена діаграма активності тренувального процесу з градієнтним спуском. Як ми бачимо, вектор параметрів поліпшується в два етапи: по-перше, обчислюється напрямок навчання градієнтному спуску, по-друге, знайдений відповідний курс навчання.

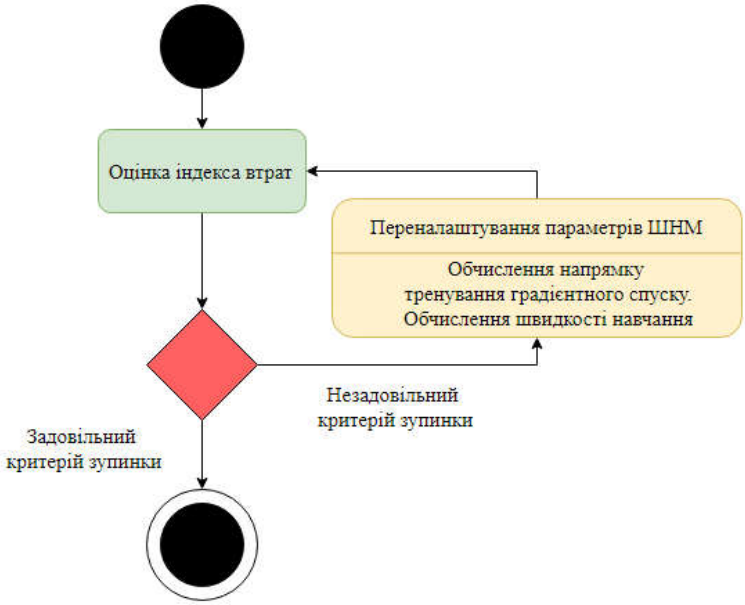


Рисунок 2.2 – Алгоритм навчання – градієнтний спуск

Алгоритм навчання градієнтному спуску має серйозний недолік: він вимагає багато ітерацій для функцій, що мають довгі вузькі структури долин. Дійсно, похилий ухил – це напрямок, в якому функція втрат зменшується найшвидше, але це не обов'язково призводить до найшвидшої збіжності. Рисунок 2.3 ілюструє цю проблему.

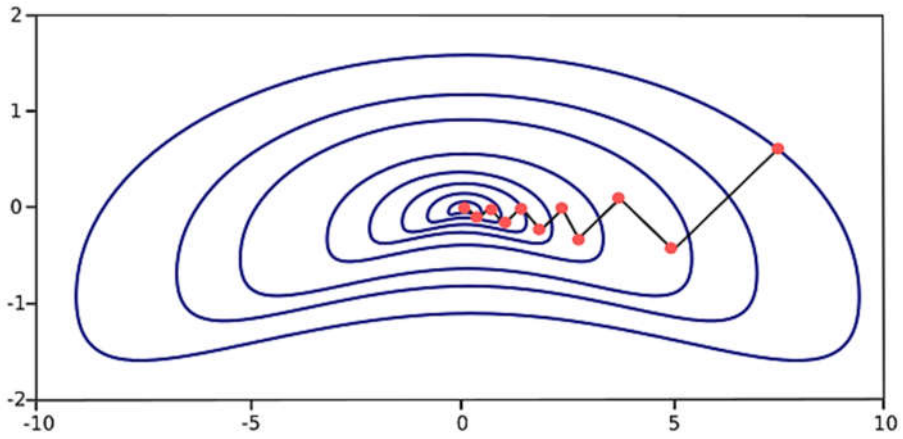


Рисунок 2.3 – Пошук мінімального значення функції втрат

методом градієнтного спуску

Метод градієнтного спуску - це рекомендований алгоритм, коли у нас дуже великі нейронні мережі з багатьма тисячами параметрів. Причина в тому, що цей метод зберігає тільки вектор градієнта (розмір n), і він не зберігає матрицю Гессе (розмір n^2).

2.3 Ньютонівський метод

Мета цього методу - знайти найкращі напрями навчання, використовуючи другі похідні функції втрат [48].

Позначимо $f(w_i) = f_i$, $\nabla f(w_i) = g_i$ та $Hf(w_i) = H_i$. Розглянемо квадратичне наближення f в w_0, використовуючи розкладання в ряд Тейлора

$$f = f_0 + g_0(w - w_0) + 0.5(w - w_0)^2 H_0,$$

де H_0– гессенська матриця функції f, обчислена в точці w_0.

Встановивши g рівним 0 для мінімуму $f(w)$, отримаємо наступне рівняння

$$g = g_0 + H_0(w - w_0) = 0.$$

Тому, починаючи з вектора параметрів w_0, метод Ньютона повторюється в такий спосіб

$$w_{i+1} = w_i + H_i^{-1} g_i, \;\; i = 0,1, \ldots$$

Вектор $H_i^{-1} g_i$ відомий як крок Ньютона. Ця зміна параметрів може рухатися до максимуму, а не до мінімуму. Відбувається це, якщо матриця Гессе не є

позитивно визначеною. Таким чином, оцінка функції не гарантується на кожній ітерації. Щоб запобігти таких неприємностей, рівняння методу Ньютона зазвичай модифікується як:

$$w_{i+1} = w_i - (H_i^{-1} g_i)\eta_i, \;\; i = 0,1, \ldots$$

Швидкість навчання η може бути встановлена на фіксоване значення, або знайдена шляхом мінімізації лінії. Вектор $d = H_i^{-1} g_i$ тепер називається напрямком навчання Ньютона.

Діаграма станів для тренувального процесу за методом Ньютона зображена на рисунку 2.4 Тут поліпшення параметрів виконується шляхом отримання спочатку напрями навчання Ньютона, а потім підходящої швидкості навчання.

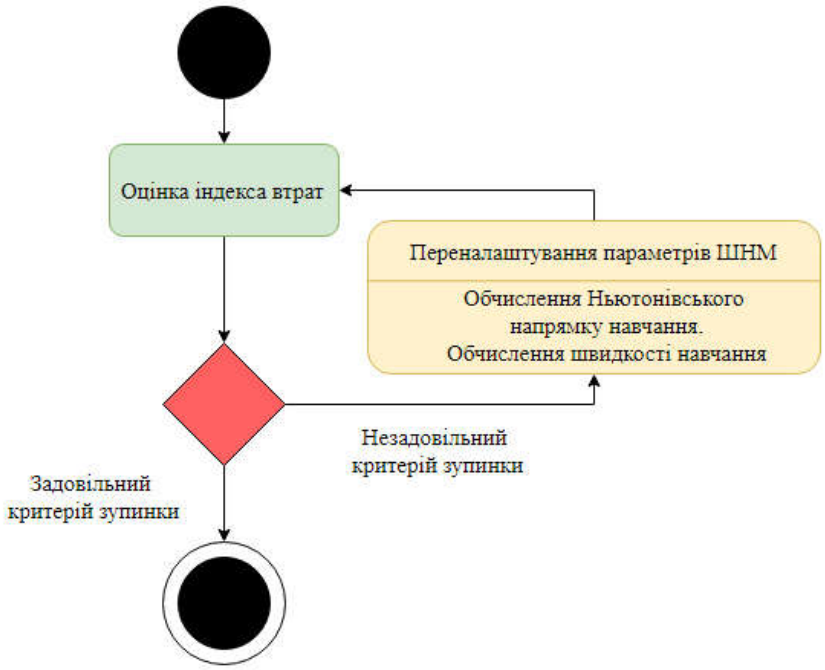


Рисунок 2.4 – Ньютонівський алгоритм навчання

Рисунок 2.5 ілюструє ефективність цього методу. Як ми бачимо, метод Ньютона вимагає менше кроків, ніж градієнтний спуск, щоб знайти мінімальне значення функції втрат.

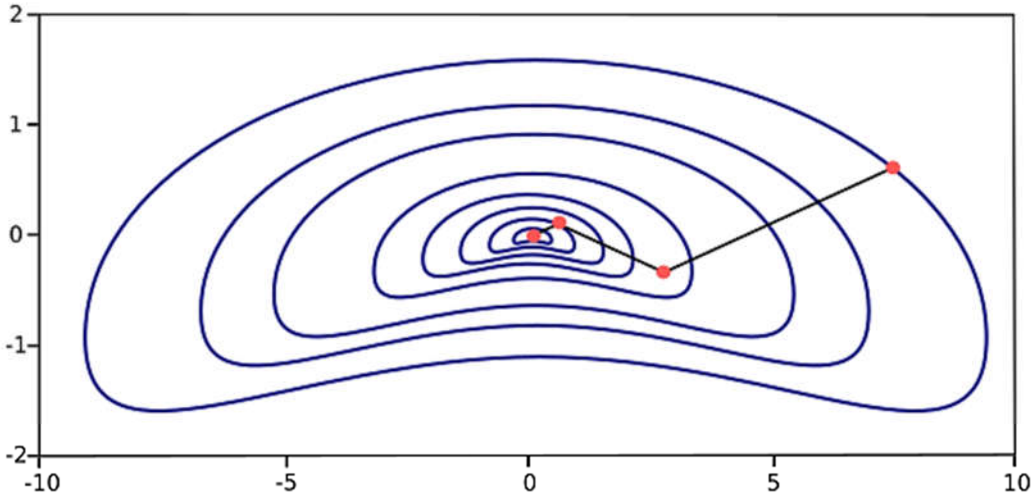


Рисунок 2.5 – Пошук мінімального значення функції втрат методом Ньютона

Проте, метод Ньютона стикається з труднощами, що полягають в тому, що точна оцінка Гессе і його зворотного значення досить дорогі в обчислювальному відношенні.

2.4 Метод сполучених градієнтів

Метод сполучених градієнтів можна розглядати як щось середнє між градієнтним спуском і методом Ньютона. Він прискорює повільну збіжність, пов'язану з градієнтним спуском. Цей метод також дозволяє уникнути вимог до інформації, пов'язаних з оцінкою, зберіганням і інверсією матриці Гессе, як того вимагає метод Ньютона [49].

В алгоритмі навчання пошук виконується по зв'язаних напрямках, що зазвичай призводить до більш швидкої збіжності, ніж до напрямів градієнтного спуску. Ці напрями навчання пов'язані щодо матриці Гессе.

Позначимо вектор напрямку навчання через d. Потім, починаючи з початкового вектора параметрів w_0 і початкового вектора напрямку навчання

$d_0 = -g_0$, метод сполучених градієнтів створює послідовність напрямів навчання в вигляді:

$$d_{i+1} = g_{i+1} + d_i y_i, \;\; i = 0,1, \ldots$$

Тут y називається параметром сполучення, і існують різні способи його обчислення. Два з найбільш часто використовуваних – спосіб Флетчера-Рівза, а також Полак-Рібьера. Для всіх алгоритмів сполученого градієнта напрямок навчання періодично скидається на негативне значення градієнта.

Потім параметри покращуються. Швидкість навчання η зазвичай визначається шляхом мінімізації лінії:

$$w_{i+1} = w_i + d_i \eta_i, \;\; i = 0,1, \ldots$$

На рисунку 2.6 зображена діаграма активності для алгоритму навчання методом сполучених градієнтів. Тут покращення параметрів виконується спочатку шляхом обчислення напрямку навчання сполучених градієнтів, а потім підходящої швидкості навчання в цьому напрямку.

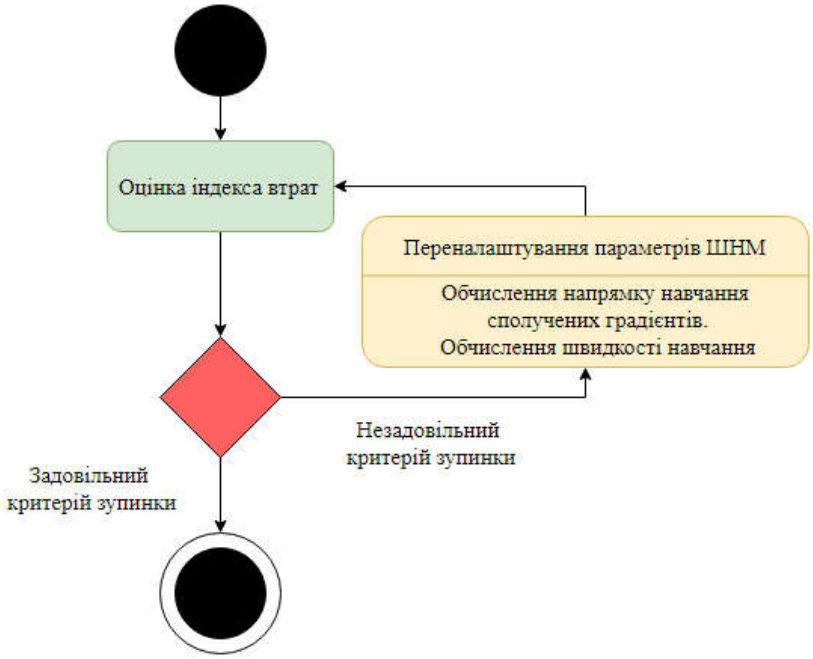


Рис. 2.6 – Алгоритм сполучених градієнтів

Цей метод виявився більш ефективним, ніж градієнтний спуск в навчальних нейронних мережах. Оскільки для нього не потрібна матриця Гессе, метод сполучених градієнтів також рекомендується, коли у нас дуже великі нейронні мережі.

2.5 Квазі-Ньютонівський метод

Застосування методу Ньютона обчислювально дорого, так як вимагає багато операцій для обчислення матриці Гессе і обчислення її у зворотному порядку. Для усунення цього недоліку розроблені альтернативні підходи, відомі як квазі-Ньютонівський або змінні метричні методи [50]. Ці методи, замість безпосереднього обчислення Гессе і подальшого обчислення його у зворотному порядку, створюють наближення до зворотного Гессе на кожній ітерації алгоритму. Це наближення обчислюється з використанням тільки інформації про перші похідні функції втрат.

Матриця Гессе складається з других часткових похідних функції втрат. Основна ідея квазі-Ньютонівського методу – апроксимувати зворотний Гессе іншою матрицею G, використовуючи тільки перші часткові похідні функції втрат. Тоді квазі-Ньютонівська формула може бути виражена як:

$$w_{i+1} = w_i - (G_i g_i)\eta_i, \;\; i = 0,1, \dots$$

Тривалість навчання η може бути встановлена на фіксоване значення або знайдена шляхом мінімізації лінії. Зворотне наближення Гессе G має різні особливості. Двома найбільш використовуваними є формула Девідона-Флетчера-Пауелла і формула Бройде-Флетчера-Гольдфарба-Шанно [51], [52].

Діаграма активності квазі-Ньютонівського алгоритму навчання показана нижче на рисунку 2.7 Поліпшення параметрів виконується спочатку шляхом

отримання квазі-Ньютонівського напрямку навчання, а потім шляхом знаходження задовільної швидкості навчання.

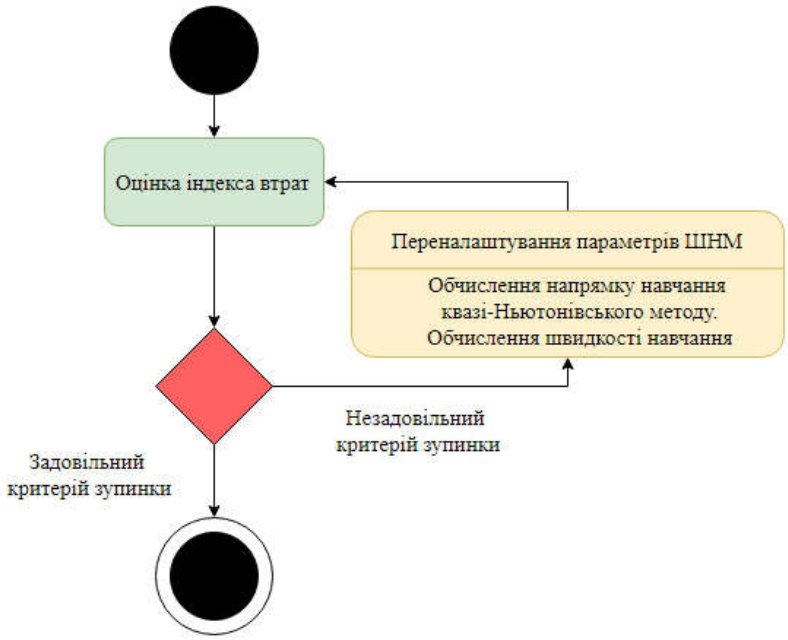


Рисунок 2.7 – Квазі-Ньютонівський алгоритм навчання

Цей метод використовується по замовчуванню в більшості випадків: він швидше градієнтного спуску і сполучених градієнтів, і точніший Гессе не потребує обчислення і інвертування.

2.6 Алгоритм Левенберга-Марквардта

Алгоритм Левенберга-Марквардта, також відомий як метод пошуку найменших квадратів, був розроблений спеціально для роботи з функціями втрат, які приймають форму суми квадратів помилок [53]. Він працює без обчислення точної матриці Гессе. Замість цього він працює з вектором градієнта і матрицею Якобі.

Розглянемо функцію втрат, яка може бути виражена як сума квадратів помилок виду

$$f = \sum e_i^2 , \;\; i = 0, \dots, m,$$

де m – кількість прикладів в наборі даних.

Ми можемо визначити функцію втрат у матриці Якобі як ту що містить похідні помилок за параметрами,

$$J_{ij} f(w) = \frac{de_i}{dw_j} , \;\; i \, \epsilon \, [1, \dots, m] \; \& \; j \, \epsilon \, [1, \dots, n],$$

де m – кількість прикладів в наборі даних,

n – кількість параметрів в нейронній мережі.

Зверніть увагу, що розмір матриці Якобі дорівнює mn.

Вектор градієнта функції втрат може бути обчислений як:

$$\nabla f = 2J^T e,$$

де e – вектор всіх помилок.

Нарешті, ми можемо апроксимувати матрицю Гессе наступним виразом:

$$Hf \approx 2J^T J + \lambda I,$$

де λ – коефіцієнт затухання, що забезпечує позитивність Гессе,

I – одинична матриця.

Наступний вираз визначає процес покращення параметрів за допомогою алгоритму Левенберга-Марквардта:

$$w_{i+1} = w_i - \left(J_i^T J_i + \lambda_i I\right)^{-1} (2J_i^T e_i), \;\; i = 0,1, \dots$$

Коли коефіцієнт затухання λ дорівнює нулю, це всього лише метод Ньютона, який використовує наближену матрицю Гессе. З іншого боку, коли λ великий, це стає градієнтним спуском з невеликою швидкістю навчання.

Параметр λ ініціюється як великий, так що перші поновлення представляють собою невеликі кроки в напрямку градієнтного спуску. Якщо будь-яка ітерація призводить до помилки, то λ збільшується на деякий коефіцієнт. В іншому випадку при зменшенні втрат λ зменшується, так що алгоритм Левенберга-Марквардта наближається до методу Ньютона. Цей процес зазвичай прискорює збіжність до мінімуму.

На рисунку 2.8 представлена діаграма стану процесу навчання нейронної мережі з використанням алгоритму Левенберга-Марквардта. Першим кроком є обчислення втрат, градієнта і гессенського наближення. Потім коефіцієнт затухання регулюється так, щоб зменшити втрати на кожній ітерації.

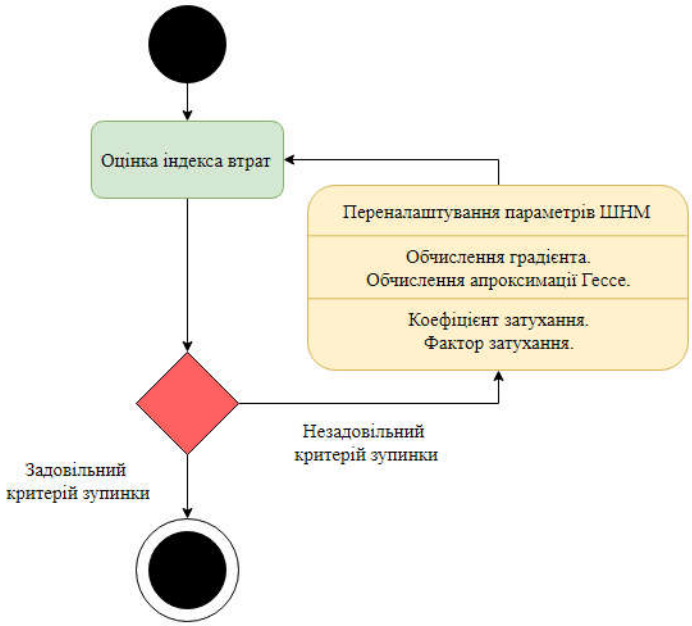


Рисунок 2.8 – Алгоритм Левенберга-Марквардта

Як ми вже бачили, алгоритм Левенберга-Марквардта являє собою метод, адаптований для функцій типу суми квадратів помилок. Це робить його дуже швидким при навчанні нейронних мереж, вимірюваних на такого роду помилки. Однак цей алгоритм має деякі недоліки. По-перше, його не можна застосовувати

до таких функцій, як середньоквадратична помилка або помилка крос-ентропії. Крім того, це не сумісно з умовами регуляризації. Нарешті, для дуже великих наборів даних і нейронних мереж матриця Якобі стає величезною, і, отже, вона вимагає багато пам'яті. Тому алгоритм Левенберга-Марквардта не рекомендується, коли у нас великі набори даних і / або нейронні мережі.

Рисунок 2.9 відображає швидкість обчислень і вимоги до пам'яті алгоритмів навчання, обговорюваних в цьому розділі. Найповільніший алгоритм навчання – градієнтний спуск (backpropagation) але він вимагає менше пам'яті. Найшвидшим є алгоритм Левенберга-Марквардта, але він вимагає багато пам'яті. Хорошим компромісом є квазі-Ньютонівський метод.

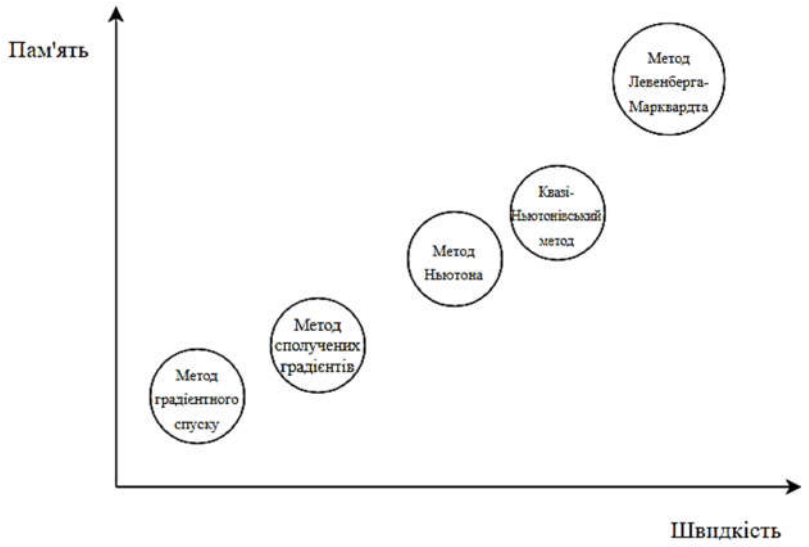


Рисунок 2.9 – Порівняння ефективності алгоритмів навчання ШНМ

Якщо нейронні мережі мають багато тисяч параметрів, використовується метод градієнтного спуску або метод сполучених градієнтів, щоб заощадити пам'ять. Якщо є багато нейронних мереж для навчання з декількома тисячами примірників і декількома сотнями параметрів, кращим вибором є алгоритм Левенберга-Марквардта.

3 РОЗРОБКА ПРОГРАМИ ПАРАЛЕЛЬНОГО НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ

3.1 Засоби та платформа реалізації

Проаналізувавши задачу яка має бути реалізована, зрозуміло що перевага має надаватися мові програмування високого рівня, яка здатна продуктивно виконувати вихідний код програми. Так було обрано для розробки мову C++ в поєднанні з бібліотекою QT. QT – крос-платформовий інструментарій розробки програмного забезпечення (ПЗ) мовою програмування C++. Дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем (ОС), просто компілюючи текст програми для кожної операційної системи без зміни сирцевого коду. Містить всі основні класи, які можуть бути потрібні для розробки прикладного програмного забезпечення, починаючи з елементів графічного інтерфейсу й закінчуючи класами для роботи з мережею, базами даних, OpenGL, SVG і XML. Бібліотека дозволяє керувати нитями, працювати з мережею та забезпечує крос-платформовий доступ до файлів.

Qt 5, який вийшов у грудні 2012, примітний модульною структурою та зміщенням акценту в бік використання для написання застосунків засобів декларативного опису інтерфейсу з визначенням логіки взаємодії з користувачем мовою JavaScript, у той час як застосування C++ позиціонується для реалізації критичних до часу виконання або надмірно складних частин програми, а також для створення нових модульних бекендів для Qt Quick. Незважаючи на багато істотних поліпшень і змін, Qt 5 зберігає базову зворотну сумісність із минулими випусками, підтримує повною мірою засоби для створення Qt-програм мовою C++ і містить майже всі компоненти Qt 4 (припинена підтримка давно застарілих елементів), більшість модулів колишнього Qt Mobility й деякі експериментальні елементи з Qt Labs.

В якості засобів паралелізації обрано QTConcurrent – високорівневе API, яке дозволяє запускати багатопоточні програми без використання 42 низькорівневих

потокових примітивів, для розмежування основної функціональності та інтерфейсу користувача. В якості основного засобу паралелізації обрано технологію OpenMP через можливість простої реалізації однопотокового та паралельного виконання програми.

3.2 Розробка схем функціонування програми

Було досліджено достатньо методів та технологій для реалізації паралельного методу навчання нейронної мережі. Вирішено використовувати метод паралелізації на рівні фази навчання, оскільки саме тут можна проводити дослідження мережі при різних первинних установках. Так само при використанні паралелізації навчання серед плюсів виділяється лінійний приріст швидкодії фази. Це можливо завдяки відсутності необхідності в взаємодії між процесорами.

В якості алгоритму навчання було обрано метод навчання з вчителем для багатошарового перцептрону – метод зворотного розповсюдження помилки, розглянутий раніше.

На вхід алгоритму, крім параметрів методу навчання, потрібно також подавати формат структури мережі. На практиці непогані результати показують мережі досить простої структури, що складаються з двох рівнів нейронів — прихованого рівня (hidden units) і нейронів-виходів (output units), кожен вхід мережі з'єднаний з усіма прихованими нейронами, а результат роботи кожного прихованого нейрона подається на вхід кожному з нейронів-виходів. У такому випадку досить подавати на вхід кількість нейронів прихованого рівня.

На рисунку 3.1 зображено блок-схему алгоритму, побудовану із врахуванням обраної технології та підходу до паралелізації.

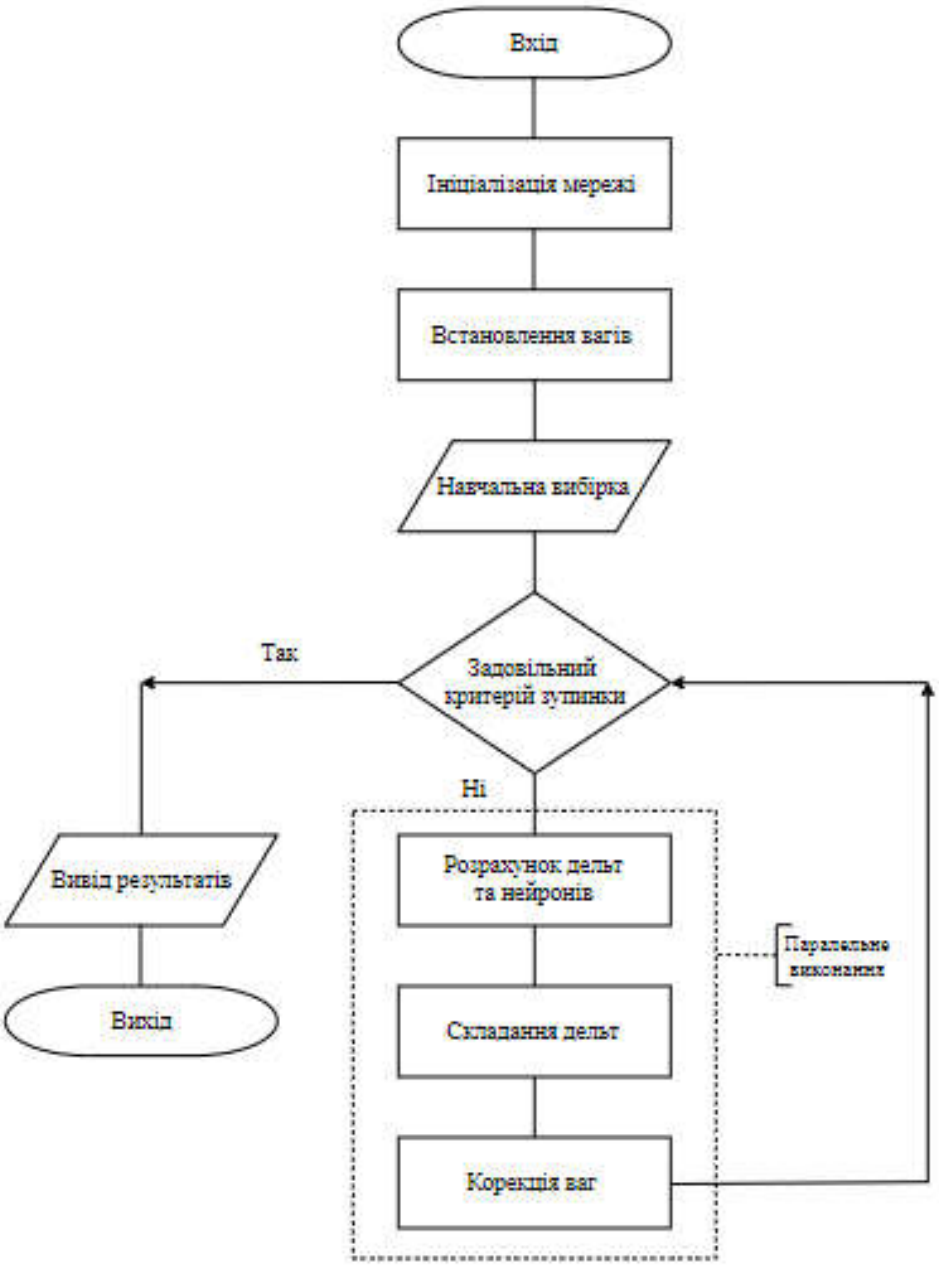


Рисунок 3.1 – Блок-схема алгоритму паралельного навчання нейронної мережі

Програма розрозблена згідно з наведеною на рисунку 3.2 діаграмою класів. Розглянемо складові діаграми, щоб детальніше зрозуміти роботу програми.

Клас MainController унаслідований від QObject відповідає за виконання основного функціоналу програми, а також застосовує параметри та функції які викликаються із графічного інтерфейсу користувача. Відповідає за запуск нового

потоку в якому будуть проводиться обчислення, не залежно від інтерфейсу, а також приймає та відображає статистичну та графічну статистику після завершення навчання мережі.

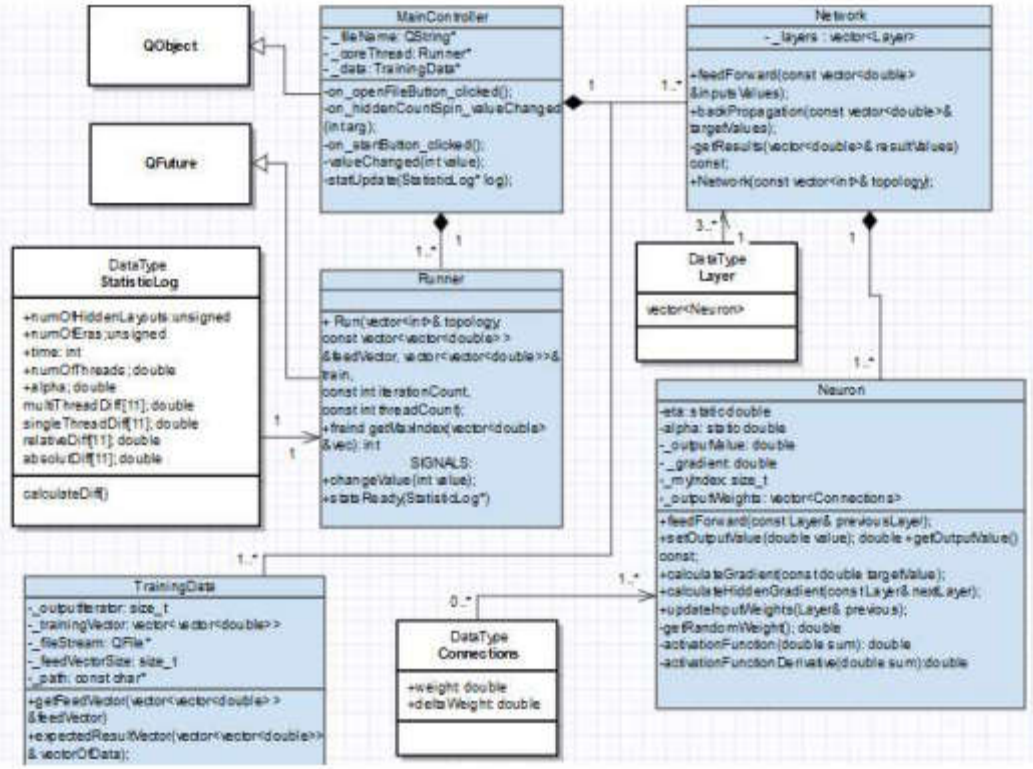


Рисунок 3.2 – UML діаграма класів програми

TrainingData – реалізує зчитування навчального набору даних, у форматі Semion dataset – бази даних зразків рукописного написання цифр. Дані складаються з заздалегідь підготовлених прикладів зображень, на основі яких 46 проводиться навчання та тестування систем. Дані представляють собою набір із 1593 рукописних символів написаних 80 різними людьми, які були відскановані, нормалізовані та зменшені до розміру 16x16.

Runner – клас, що реалізує високорівневе API QTConcurrent / QFuture, що дозволяє винести основну функціональність в окремий потік, що в свою чергу запобігає зависанню інтерфейсу користувача. Саме в функції Run цього класу і відбувається основна паралелізації навчання. Ця функція виконується двічі- в послідовному та паралельному режимах. Також після виконання функції Run формується StatisticLog – службовий клас для збереження інформації про часові

затрати даного раунду навчання, далі він використовується для представлення статистики в текстовому, або графічному режимах.

Клас Network реалізує навчання нейронної мережі методом зворотного розповсюдження помилки. Даний клас забезпечує наступний функціонал: ініціалізацію мережі та встановлення ваг в конструкторі класу, ініціалізацію входів мережі згідно навчального вектору за допомогою функції feedVector. Сам алгоритм зворотного розповсюдження помилки реалізований функцією backpropagation. За розрахунок дельт на нейронах та їх складання відповідає клас Neuron. Також в ньому реалізовані функція активації та її похідна, в нашому випадку в якості функції активації було обрано гіперболічний тангенс, а в якості похідної його апроксимацію $1 - x^2$. Також в даному класі зберігаються змінні що відповідають за швидкість навчання мережі. $\eta(eta)$ – коефіцієнт швидкості навчання мережі та α– множник ваг попередніх ваг. Експериментально було визначено значення цих змінних 0.35 та 0.05 відповідно/

На рисунку 3.3 наведено діаграму прецедентів. Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціації між акторами та прецедентами, відношення серед прецедентів та відношень узагальнення між акторами.

Рисунок 3.3 – Діаграма прецедентів

У системі представлено одну роль – Користувач. Взаємодія з користувачем відбувається за рахунок графічного інтерфейсу (рисунок 3.4).

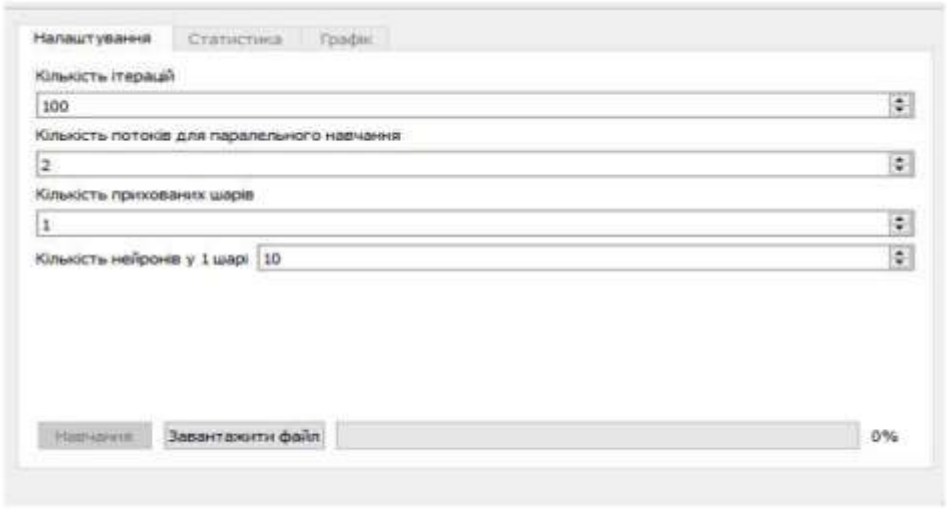


Рисунок 3.4 – Графічний інтерфейс програми

Інтерфейс представлений у вигляді 3х вкладок Налаштування, Статистика, та Графік. Вкладки Статистика та Графік стають доступними відразу після завершення процесу навчання.

На сторінці налаштувань має доступ налаштувань процесу навчання, а саме, може встановлювати: кількість ітерацій навчання, потоків для паралельного режиму навчання, кількість прихованих шарів нейронної мережі, а також кількість нейронів у кожному шарі. При цьому накладаються наступні обмеження:

- кількість ітерацій [100 – 10000];
- кількість потоків [2 – 8];
- кількість прихованих шарів [1 – 4];
- кількість нейронів для кожного шару [10 – 512].

Також користувач має можливість завантажувати файли для навчання типу semion data set (рисунок 3.5).

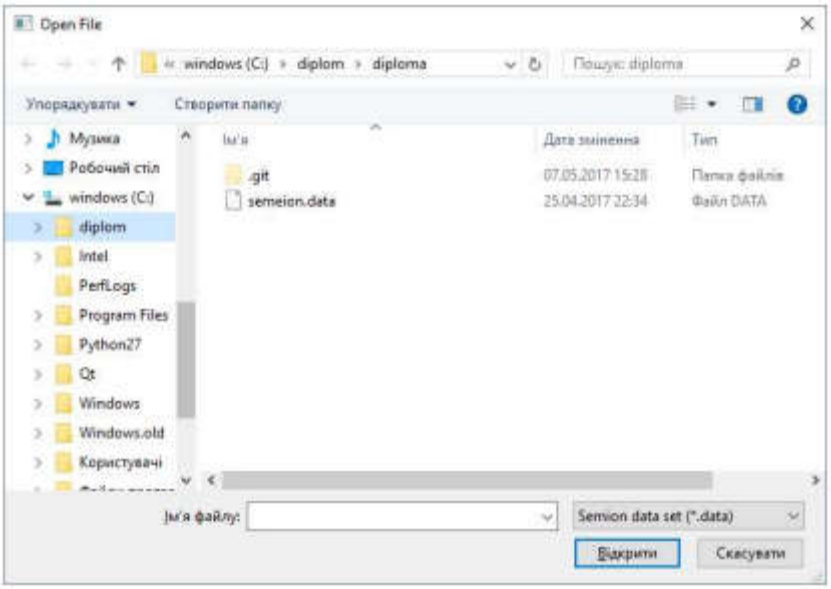


Рисунок 3.5 – Діалогове вікно завантаження файлу

Після налаштувань, та вибору файлу користувач може почати процес навчання. Навчання відбувається двічі в однопотоковому і паралельному режимах відповідно. Поточний статус виконання відображається за допомогою прогресс бару.

На вкладці Статистика користувач може переглянути наступну інформацію:

– кількість Ітерацій;

– кількість Прихованих шарів;

– кількість ядер системи;

– кількість згенерованих потоків;

– час однопотокового навчання;

– час багатопотокового навчання;

– коефіцієнт ефективності, обрахований по формулі 1.1 (рисунок 3.6).

А також таблицю часових затрат на кожну декаду навчання, для однопотокового та паралельного режимів, абсолютна та відносна різниця між режимами.

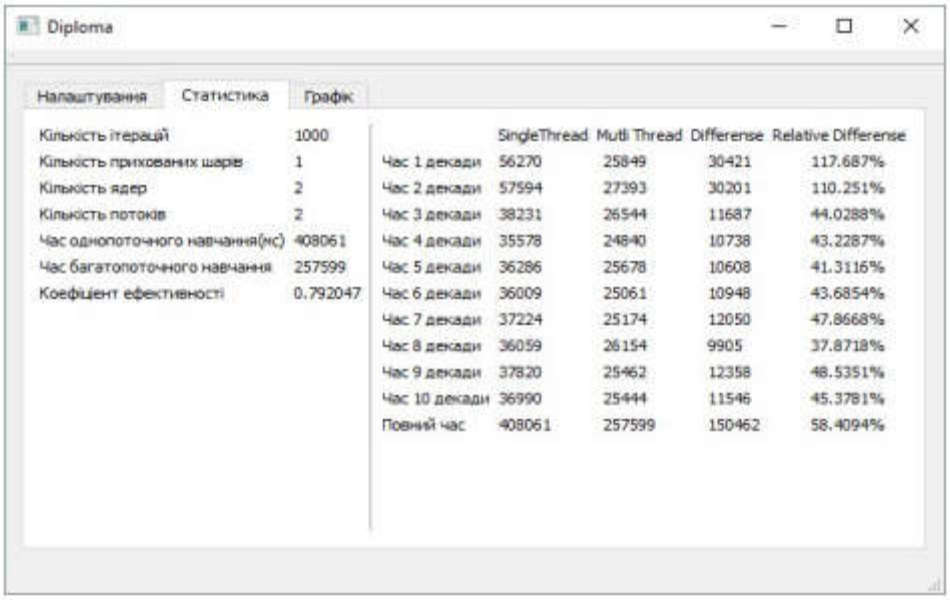


Рисунок 3.6 – Вкладка статистики

На рисунку 3.7 наведено абсолютну різницю у часі виконання між декадами представленої в вигляді графіка.

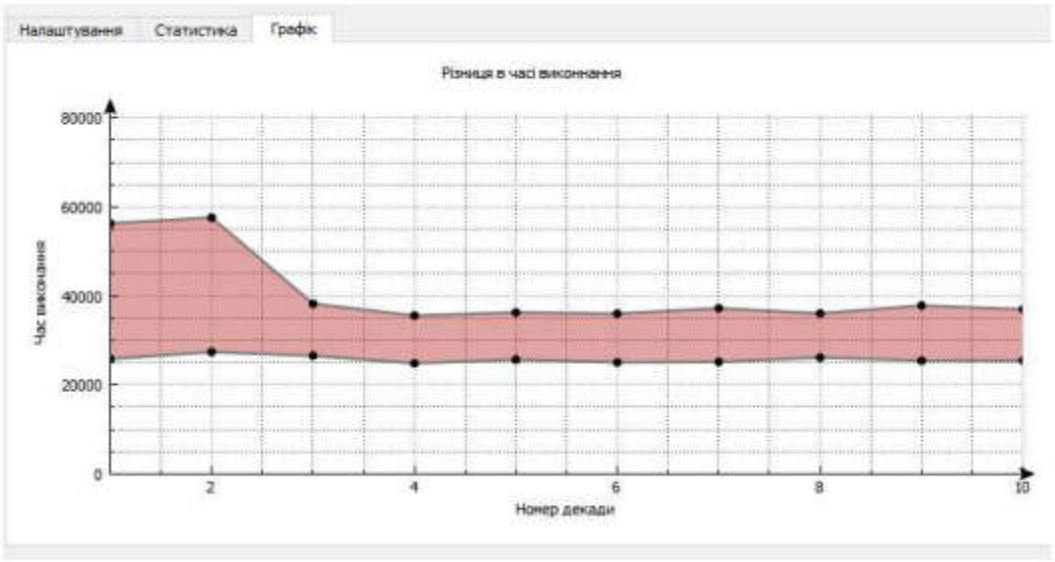


Рисунок 3.7 – Графічне представлення різниці в часі виконання

У ефективності процесу паралелізації нейронних мереж можна переконатись лише шляхом тестування.

3.4 Тестування та верифікація програми

Для оцінки роботи будемо розглядати абсолютний та відносний виграш у часі та коефіцієнт ефективності. В тестах перевірено вплив паралелізації на різні моделі навчальної системи, де ступінь навчання залежить від кількості: ітерацій, нейронів, шарів, та комбінацій цих параметрів. Тести виконувались послідовно при однаковій завантаженості системи, та повторювались декілька разів, для усереднення результатів.

Для першого тесту обрано такі параметри:

– кількість Ітерацій – 1000;

– кількість Прихованих шарів – 1;

– кількість потоків – 4;

– кількість нейронів у шарі – 10.

На рисунку 3.8 наведено результати даного тесту.

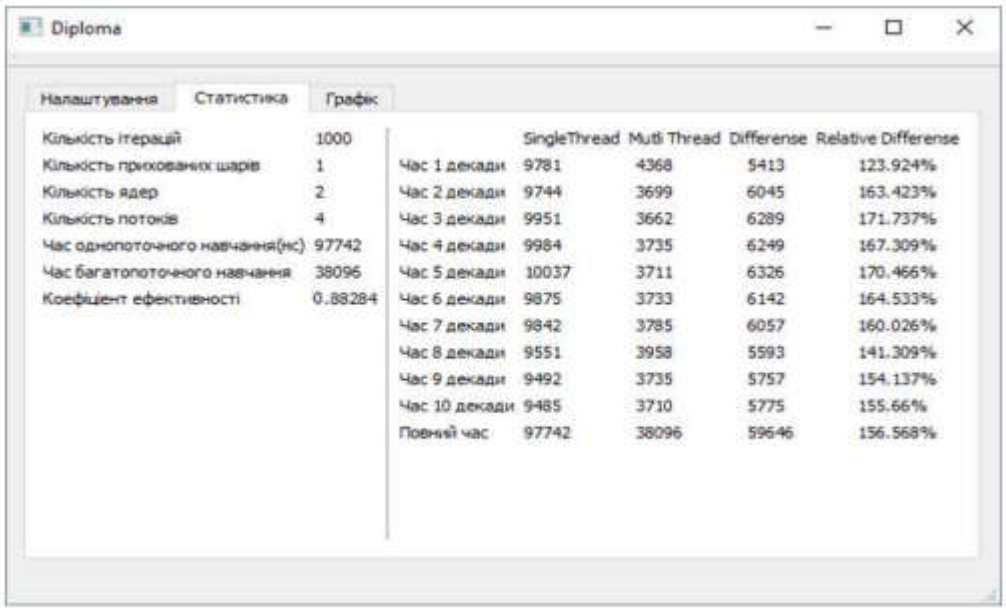


Рисунок 3.8 – Результати тесту №1

Абсолютний виграш у часі становив приблизно 1хв, а ефективність алгоритму паралелізації 0.88284. Отже, можна вважати, що обрана методика

паралелізації істотно покращує результати навчання при великій кількості ітерацій навчання.

Для другого тесту обрано такі параметри:

– кількість Ітерацій – 100;

– кількість Прихованих шарів – 1;

– кількість потоків – 4;

– кількість нейронів у шарі – 512.

На рисунку 3.9 наведено результати другого тесту.

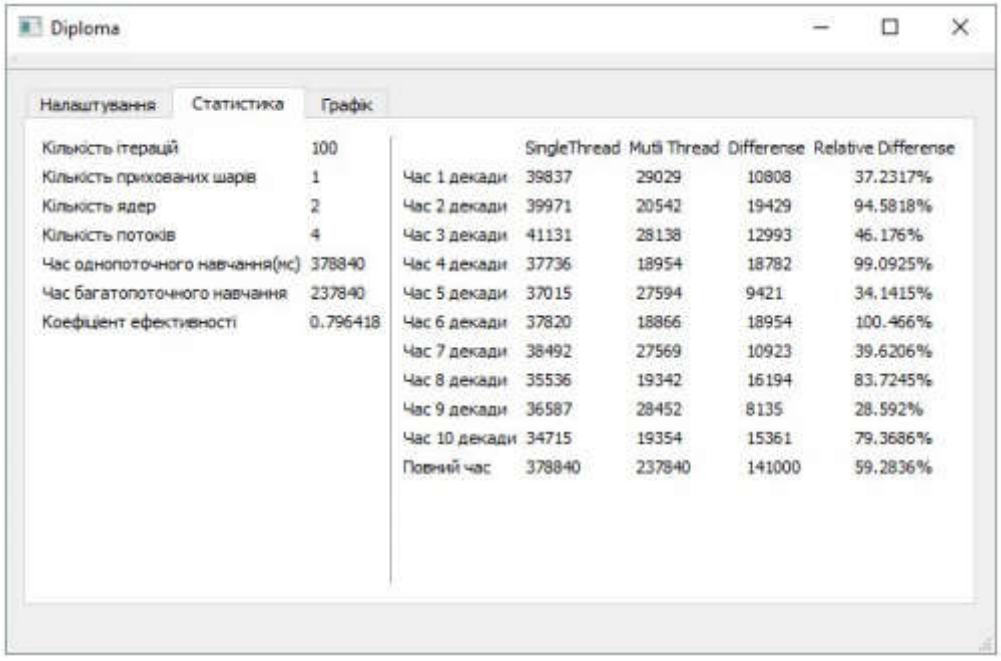


Рисунок 3.9 – Результати тесту №2

Абсолютний виграш у часі становив 141000мс, а ефективність алгоритму паралелізації 0.796418.

Для третього тесту обрано такі параметри:

– кількість Ітерацій – 1000;

– кількість Прихованих шарів – 4;

– кількість потоків – 4;

– кількість нейронів у шарі - 10 10 10 10.

На рисунку 3.10 наведено результати даного тесту.

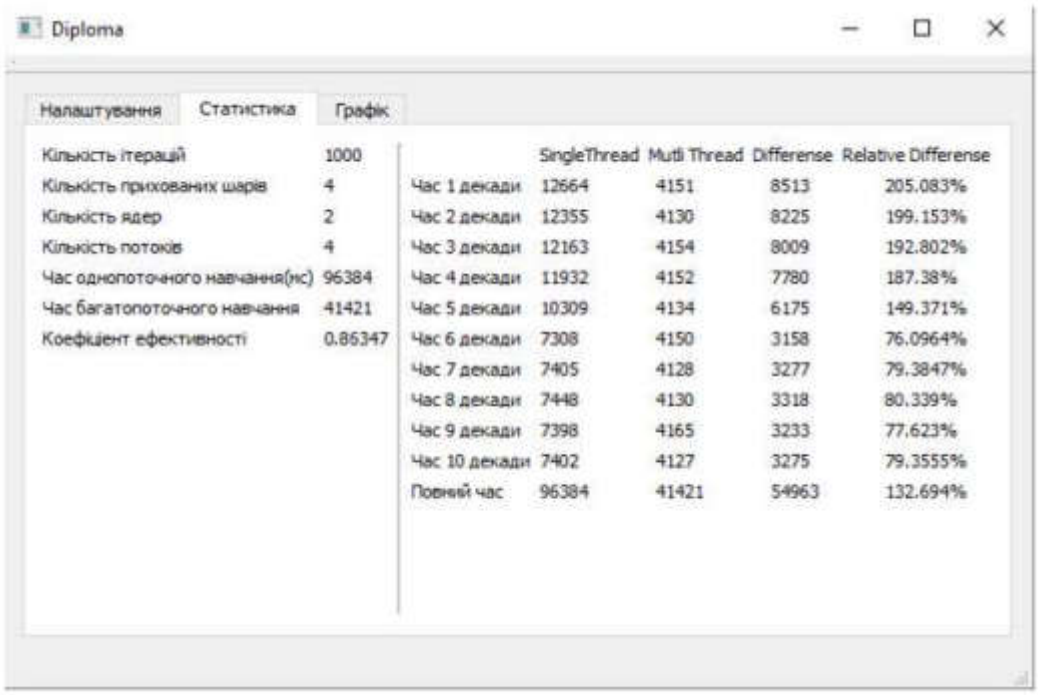


Рисунок 3.10 – Результати тесту №3

В даному тесті протестована багатошаровість мережі в поєднанні з великою кількість ітерацій. Коефіцієнт ефективності становив 0.865347, відносний виграш у часі 132%. Також помітно, що незначна кількість нейронів в шарах, майже не впливає на час виконання. Так в порівнянні з Тестом №1 час лінійного та паралельного навчання майже не змінився, хоча й мережа мала додаткові 3 шари по 10 нейронів у кожному.

Для четвертого тесту обрано такі параметри:

– кількість Ітерацій – 100;

– кількість Прихованих шарів – 4;

– кількість потоків – 4;

– кількість нейронів у шарі – 100 100 100 100.

На рисунку 3.11 наведено результати даного тесту.

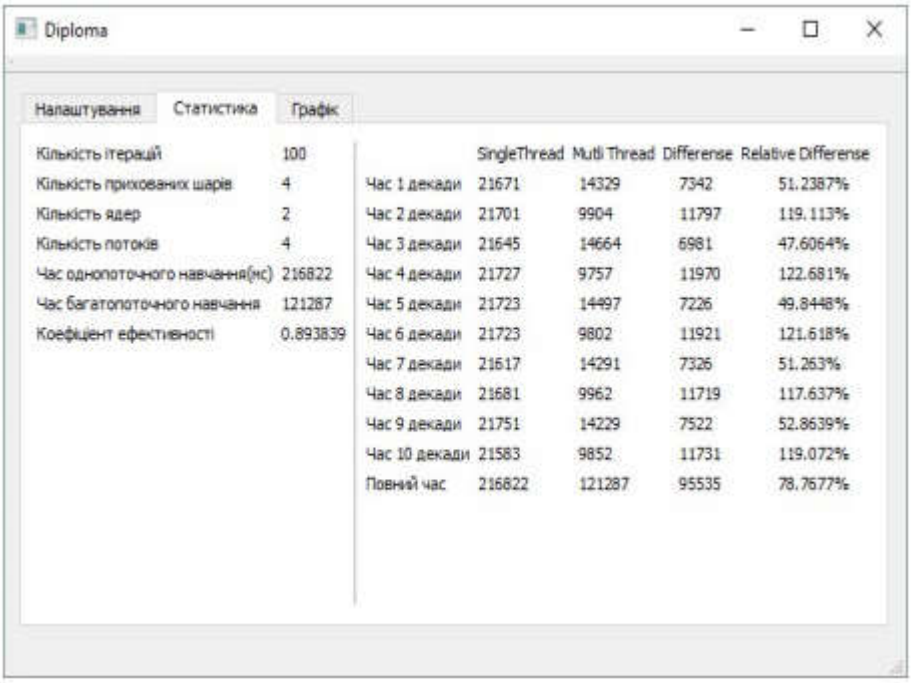


Рисунок 3.11 – Результати тесту №4

В даному тесті протестована багатошаровість мережі в поєднанні з великою кількість нейронів в мережі. Отриманий коефіцієнт ефективності 0.893839 та відносний виграш у час в 78.77%, що свідчить про ефективність алгоритму на даній конфігурації.

Таким чином розроблена програма паралельного навчання нейронної мережі, відповідно до спроектованої архітектури, та визначених основних функціональних аспектів системи, при тестуванні показала високу ефективність. Середній виграш у часі в ході тестування становив на всіх тестах був більше 80%. Також програма використовувала всі можливі ресурси системи для пришвидшення паралельного навчання, про що свідчать показники завантаженості системи під час навчання 23% для лінійного і 94% для паралельного режимів, що свідчить про ефективність обраної технології паралелізації.

ВИСНОВКИ

В роботі розв’язано актуальну задачу розпаралелення алгоритмів навчання загорткових нейронних мереж на основі графічних процесорів. При цьому отримано наступні результати:

1) проаналізовано методи розпаралелення штучних нейронних мереж, переваги та недоліки використання кожного із них;

2) досліджено сучасні технології та засоби розпаралелення, зокрема ті, які підтримують роботу із графічними процесорами, а саме: CUDA, OpenMP;

3) проаналізовано алгоритми навчання згорткових нейронних мереж, та проведемо порівняння швидкодії при обробці великої кількості даних;

4) розроблено схеми функціонування програми та програмний засіб паралельного навчання нейронних мереж алгоритмом зворотного поширення помилки із можливість задання користувачем основних параметрів роботи мережі;

5) проведено ряд тестів, що підтвердили ефективність використання графічних процесорів для опрацювання великої кількості даних нейронними мережами.

Таким чином використання графічних процесорів при роботі із нейронними мережами дозволяє отримати бажаний результат в рази швидше, ніж при використанні центральними процесорами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is artificial neural network (ANN)? [Електронний ресурс] / – Режим доступу до ресурсу: https://searchenterpriseai.techtarget.com/definition/neural-network.

2. Розенблатт Ф. Принципы нейродинамики. Перйептроны и теория механизмов мозга / Ф. Розенблат. – М.: Мир, 1965. – 175с.

3. Minsky M. L. Perceptrons / Minsky M. L. Papert S. A. – Cambridge, MA: MIT Press, 1969.

4. Werbos, P.J. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences [Електронний ресурс] / – Режим доступу до ресурсу: https://www.researchgate.net/publication/35657389

5. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Rumelhart, D.E; McClelland, James / Cambridge: MIT Press. ISBN 978-0-262-63110-5, 1986 – 567c.

6. What is big data? [Електронний ресурс] / – Режим доступу до ресурсу: https://searchdatamanagement.techtarget.com/definition/big-data

7. How Does the Brain Works? [Електронний ресурс] / Sandra Blakeslee // The New York Times – Nov. 11, 2003 – Режим доступу до ресурсу: https://www.nytimes.com/2003/11/11/science/how-does-the-brain-work.html.

8. Artificial Neural Network (ANN) in Machine Learning [Електронний ресурс] /–Режим доступу до ресурсу: https://www.datasciencecentral.com/profiles/blogs/artificial-neural-network-ann-in-machine-learning.

9. Yoon Kim Convolutional Neural Networks for Sentence Classification // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) – Doha, Qatar. – October 25-29, 2014. – P. 1746 – 1751.

10. Xiang Zhang Text Understanding from Scratch / Xiang Zhang, Yann LeCun // Computer Science Department, Courant Institute of Mathematical Sciences, New York University – New York, USA. – Apr 4, 2016.

11. Wen-tau Yih Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base / Wen-tau Yih, Ming-Wei Chang, Xiaodong He, Jianfeng Gao // Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing – Beijing, China. – July 26-31, 2015. – P. 1321 – 1331.

12. Dasha Bogdanova Detecting Semantically Equivalent Questions in Online User Forums / Dasha Bogdanova, C´ıcero dos Santos, Luciano Barbosa† and Bianca Zadrozny // Proceedings of the 19th Conference on Computational Language Learning – Beijing, China. – July 30-31, 2015. – P. 123 – 131.

13. Ossama Abdel-Hamid Convolutional Neural Networks for Speech Recognition / Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu // IEEE/ACM Transactions On Audio, Speech, And Language Processing, VOL. 22, № 10 – October 2014. – P. 1533 – 1545.

14. Recursive neural network [Електронний ресурс] /–Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Recursive_neural_network.

15. Natural Language Generation, Paraphrasing and Summarization of User Reviews with Recurrent Neural Networks [Електронний ресурс] /–Режим доступу до ресурсу: http://www.meanotek.ru/files/TarasovDS292015-Dialogue.pdf.

16. Siwei Lai Recurrent Convolutional Neural Networks for Text Classification / Siwei Lai, Liheng Xu, Kang Liu, Jun Zhao // Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence – China. – 2015. – P.2267–2273.

17. Hasim Sak Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling / Hasim Sak, Andrew Senior, Francoise Beaufays – Singapore. – 14-18 September 2014. – P.338– 342.

18. Ilya Sutskever Long Sequence to Sequence Learning with Neural Networks / Ilya Sutskever, Oriol Vinyals, Quoc V. Le. – 14 December 2014. – P.1– 9.

19. Tomas Mikolov Efficient Estimation of Word Representations in Vector Space / Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. – 7 Sep 2013. – P.1–12.

20. Convolutional neural network [Електронний ресурс] /– Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Convolutional_neural_network.

21. Yoon Kim Convolutional Neural Networks for Sentence Classification / Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) – Doha, Qatar. – October 25-29, 2014. – P. 1746 – 1751.

22. A Friendly Introduction to Convolutional Neural Networks [Електронний ресурс] / – Режим доступу до ресурсу: https://hashrocket.com/blog/posts/a-friendly-introduction-to-convolutional-neural-networks.

23. A Practical Guide to ReLU [Електронний ресурс] / – Режим доступу до ресурсу: https://medium.com/tinymind/a-practical-guide-to-relu-b83ca804f1f7

24. Цитологічні терміни у світлі нового списку гістологічної термінології / Чайковський Ю.Б., Луцик О.Д., Геращенко С.Б., Дєльцова О.І. // Вісник морфології. – 2016. – № 2. – С. 399-403.

25. Parallel and Distributed Deep Learning [Електронний ресурс] / Vishakh Hegde, Sheema Usmani // – Режим доступу до ресурсу: https://web.stanford.edu/~rezab/classes/cme323/S16/projects_reports/hedge_usmani.pdf

26. Боголюбов Д. П., Чанкин А. А., Стемиковская К. В. Реализация алгоритма собучения самоорганизующихся карт Кохонена на графических процессорах // Промышленные АСУ и контроллеры. – 2012. № 10. С. 30-35

27. Моделі нейронних мереж. [Електронний ресурс] – Режим доступу: http://techn.sstu.ru/kafedri/подразделения/1/MetMat/Terin/neiro/neiro.htm.

28. Сегментація гістологічних зображень за допомогою ЗНМ [Електронний ресурс] / – Режим доступу до ресурсу: https://www.researchgate.net/publication/323904915_Segmentation_of_histological_images_and_fibrosis_identification_with_a_convolutional_neural_network.

29. Parallel Processing [Електронний ресурс] / – Режим доступу до ресурсу: https://searchdatacenter.techtarget.com/definition/parallel-processing.

30. Нейронна мережа Гопфілда [Електронний ресурс] / – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Нейронна_мережа_Гопфілда.

31. N. L. Roux A stochastic gradient method with an exponential convergence rate for finite training sets / N. L. Roux, M. Schmidt, and F. Bach // Advances in Neural Information Processing Systems, vol.4 – 2012. – P. 2663– 2671.

32. Advanced parallel processing technologies: 7th international symposium, APPT 2007 – Guangzhou, China – November 22-23, 2007.

33. What is neural net processor? [Електронний ресурс] / – Режим доступу до ресурсу: https://searchenterpriseai.techtarget.com/definition/neural-net-processor.

34. Which GPU(s) to Get for Deep Learning: My Experience and Advice for Using GPUs in Deep Learning? [Електронний ресурс] / – Режим доступу до ресурсу: http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning.

35. André R. Brodtkorb GPU programming strategies and trends in GPU computing / André R. Brodtkorb, Trond R. Hagen, and Martin L. Setra – 2012. – P.1–15.

36. NVIDIA cuDNN GPU Accelerated Deep Learning [Електронний ресурс] / – Режим доступу до ресурсу: https://developer.nvidia.com/cudnn.

37. J. M. Nageswaran Efficient simulation of large-scale spiking neural networks using CUDA graphics processors / J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau and A. Veidenbaum // IJCNN'09: Proceedings of the 2009 international joint conference on Neural Networks. – Atlanta, Georgia, USA. – 2009.

38. H. Jang Neural network implementation using CUDA and OpenMP / H. Jang, A. Park and K. Jung // DICTA '08: Proceedings of the 2008 Digital Image Computing: Techniques and Applications. – Washington, DC, USA. – 2008.

39. OpenMP [Електронний ресурс] / – Режим доступу до ресурсу: https://www.openmp.org.

40. Barbara Chapman Using OpenMP: portable shared memory parallel programming (Scientific and Engineering Computation) / Barbara Chapman, Gabriele Jost, Ruud van der Pas – Cambridge, Massachusetts: The MIT Press. – 2008. – PP.353.

41. Обучение нейронной сети [Електронний ресурс] / – Режим доступу до ресурсу: https://neuronus.com/theory/nn/238-obucheniya-nejronnoi-seti.html.

42. Обучение нейронной сети [Електронний ресурс] / – Режим доступу до ресурсу: http://www.aiportal.ru/articles/neural-networks/learning-neunet.html.

43. Нейронные сети для начинающих [Електронний ресурс] / – Режим доступу до ресурсу: https://habr.com/post/313216.

44. Backpropagation [Електронний ресурс] / John McGonagle, George Shaikouski, Christopher Williams – Режим доступу до ресурсу: https://brilliant.org/wiki/backpropagation.

45. A Step by Step Backpropagation Example [Електронний ресурс] / – Режим доступу до ресурсу: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example.

46. Backpropagation [Електронний ресурс] / – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Backpropagation.

47. How to understand Gradient Descent, the most popular ML algorithm [Електронний ресурс] / – Режим доступу до ресурсу: https://medium.freecodecamp.org/understanding-gradient-descent-the-most-popular-ml-algorithm-a66c0d97307f.

48. Newton's Method [Електронний ресурс] / – Режим доступу до ресурсу: http://mathworld.wolfram.com/NewtonsMethod.html.

49. Ningsheng Gong The Conjugate Gradient Method with Neural Network Control / Ningsheng Gong, Wei Shao, Hongwei Xu. – School of Electronics and Information Engineering / Nanjing University of Technology, Nanjing, Jiangsu, China. – 2010.

50. Quasi-Newton Methods [Електронний ресурс] / – Режим доступу до ресурсу: https://neos-guide.org/content/quasi-newton-methods.

51. Алгоритм Дэвидона–Флетчера–Пауэлла [Електронний ресурс] / – Режим доступу до ресурсу: https://revolution.allbest.ru/mathematics/00301736_0.html.

52. Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно [Електронний ресурс] / – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Алгоритм_Бройдена_Флетчера_Гольдфарба_Шанно.

53. What is the Levenberg–Marquardt Algorithm? [Електронний ресурс] / – Режим доступу до ресурсу: https://www.statisticshowto.datasciencecentral.com/levenberg-marquardt-algorithm.