СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра програмування та математики

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)
спеціалізація „Інженерія програмного забезпечення”

на тему „Програмна реалізація паралельних алгоритмів тренування нейронних мереж ”

Виконав: студент групи ІПЗ-16д _____ С.Ю. Безменов
(підпис) (ініціали і прізвище)

Керівник _____ Ю.Г. Ковальов
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ В.О. Лифар
(підпис) (ініціали і прізвище)

Рецензент_____

Сєвєродонецьк – 2020

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра програмування та математики

Освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)
спеціалізація „Інженерія програмного забезпечення”
(назва спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПМ,
д.т.н., доцент _____ Лифар В.О.
“____” _____2020 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

Безменову Сергію Юрійовичу

1. Тема роботи Програмна реалізація паралельних алгоритмів тренування нейронних мереж
Керівник роботи: к.ф.-м.н., Ковальов Юрій Григорійович
затверджений наказом університету від “___”_____20__року №___
2. Строк подання студентом роботи 20 травня 2020 р.
3. Вихідні дані до роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд нейронних мереж: штучні нейронні мережі, логічні нейронні мережі
2. Побудова логічної нейронної мережі
2.1 Реалізація конструктора мережі і підстановки значень
2.2 Можливості паралелізації використаних алгоритмів
3. Застосування технології CUDA для паралельної реалізації логічної нейронної мереж
5. Перелік графічного матеріалу немає

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання__30 березеня 2020 року._____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	30.03.20	
2	Укладання і погодження з керівником плану і етапів виконання роботи	06.04.20	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	13.04.20	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	20.04.20	
5	Укладання та тестування програмного продукту	27.04.20	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	04.05.20	
7	Здача готової пояснювальної записки на кафедру	12.05.20	
8	Укладання доповіді і презентації	30.05.20	

Студент _____ ___С. Ю. Безменов_____
(підпис) (ініціали і прізвище)

Керівник роботи _____ ____Ю Г. Ковальов_____
(підпис) (ініціали і прізвище)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІПЗ-16Д Безменов С.Ю.

Науковий керівник

Доцент, к.ф.-м..н. _____ Ковальов Ю.Г.

Оцінка наукового керівника: _____

Рецензент доцент. каф. ПМ СНУ ім.В.Даля Захожай О.І.
місто роботи, посада, ПІБ

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:
_____

Голова ЕК

_____ Лифар В.О.
підпис

РЕФЕРАТ

Текст – 47 стор., рис. – 11, літературних джерел – 9

Для обробки великих обсягів даних широко використовуються нейронні мережі. Завдяки можливості навчання, нейронні мережі можуть адаптуватися до нових даних і різних змін, приймаючи при цьому правильні рішення при повній автономії. Таким чином, нейронні мережі дозволяють домогтися високої точності одержуваних результатів при різних, змінюються конфігураціях вхідних даних. Робота нейронної мережі безпосередньо залежить від використовуваних алгоритмів роботи, і природно, при реалізації паралельних алгоритмів навчання і роботи, швидкість виконання поставлених завдань буде набагато більше. Відповідними конфігураціями для реалізації паралельної роботи є багатоядерні, багатопроцесорні обчислювальні машини і комплекси, а також архітектури, що дозволяють використовувати графічну карту для обчислень, такі як CUDA (Nvidia). Процеси навчання і роботи мережі за своєю суттю не є дуже вимогливими до системи, а швидкодія буде залежати лише від кількості оброблюваної інформації. Таким чином, логічні нейронні мережі можуть бути реалізовані з використанням технології CUDA для того, щоб досягти максимальної продуктивності.

НЕЙРОННІ МЕРЕЖІ, ПАРАЛЕЛЬНИЙ АЛГОРИТМ, CUDA

ЗМІСТ

ВСТУП

Актуальність. Хто володіє інформацією - той володіє світом. Однак, сама інформація не настільки важлива, як зрозуміти, що робити, володіючи цією інформацією. Потрібні системи аналізу даних і прийняття рішень на їх основі. Системи прийняття рішень набули величезного поширення в різних сферах, в яких необхідне оперативне отримання кінцевих даних, наприклад в економіці, медицині, в сферах моделювання різних систем і процесів, а так само в системах контролю роботи підприємств та системах безпеки. Крім отримання необхідного результату, системи прийняття рішень можуть надавати користувачеві різні прогнози. Однак, не дивлячись на настільки широке охоплення виконуваних завдань, системи прийняття рішень за своєю специфікою не є автономними і не можуть працювати без участі людини. Все одно необхідне втручання людини. Для створення повністю автономної системи необхідний апарат, математична модель, що дозволяє аналізувати інформацію і приймати рішення, з урахуванням здатності до навчання даної системи. Таким критеріям відповідають нейронні мережі.

Робота нейронної мережі безпосередньо залежить від використовуваних алгоритмів роботи, і природно, при реалізації паралельних алгоритмів навчання і роботи, швидкість виконання поставлених завдань буде набагато більше. Для реалізації будь-якого паралельного алгоритму, будь-яких паралельних процесів, необхідна відповідна архітектура ЕОМ, що дозволяє одночасне виконання завдань. Відповідними конфігураціями для реалізації паралельної роботи є багатоядерні, багатопроцесорні обчислювальні машини і комплекси, а так само архітектури, що дозволяють використовувати графічну карту для обчислень, наприклад, CUDA (Nvidia). Процеси навчання і роботи мережі за своєю суттю не є дуже вимогливими до системи, а швидкодія буде залежати лише від кількості оброблюваної інформації. Таким чином, логічні нейронні мережі можуть бути реалізовані з використанням технології CUDA для того, щоб досягти максимальної продуктивності.

Об'єкт дослідження – нейронні мережі

Предмет дослідження – нейронні мережі з паралельним методом логічного слідування.

Мета дослідження: реалізація логічної нейронної мережі з використанням паралельного методу логічного слідування, виконана на мові C ++ із застосуванням CUDA технології.

Задачі дослідження:

1) Розглянути моделі нейронних мереж, їх реалізації і основні ідеї.

2) Визначити клас логічних нейронних мереж і виконуваних ними завдань.

3) Розглянути послідовний і паралельний методи резолюції, як засобу логічного висновку.

4) Встановити залежність між логічним проходженням і логічним висновком.

5) Проаналізувати отриману інформацію, і на її основі викласти ідеї паралельної реалізації логічної нейронної мережі і методу логічного слідування.

6) Реалізувати логічну нейронну мережу для завдання логічного слідування.

1. Задача логічного виводу в нейронних мережах

1.1 Штучні нейронні мережі

Вперше нейронні мережі були згадані в роботі Уоррена Мак-Каллока і Уолтера Питтса «Logical calculas of the ideas immanent in nervous activity» (рус. «Логічне числення ідей, що відносяться до нервової активності») в 1943 році. У статті було формалізовано поняття нейронної мережі і описана робота нейронів формулами математики та алгебри логіки, виходячи знань про побудову нервової системи. Було відзначено, що штучний нейрон є найпростішим логічним пристроєм, однак без навчання, не здатний виконувати ніяких функцій. Алгоритм навчання був запропонований в 1949 році Дональдом Хеббом в книзі «Організація поведінки: нейропсихологическая теорія» і припускав, що при частій стимуляції нервової системи утворюються нейронні структури.

Перший одношаровий персептрон з'явився в 1958 році, завдяки Френку Розенблату і мав можливість виконувати завдання класифікації, через що здобув популярність в різних сферах. Після публікації книги «Перцептрони» Марвіном Мінським і Сеймуром Паперт в 1969 році, в якій детально наводилося доказ неможливості навчання універсального перцептрону, і описувалися класи завдань, які перцептрон не міг дозволити, інтерес до нейронних мереж практично зник.

Дослідження в цій сфері продовжилися, і в 1972 році Теуво Кохонен і Джеймс Андерсон незалежно пропонують новий тип нейронних мереж. Так, до 1982 року, нейромережеві технології були в занепаді. Однак, модель мережі із зворотними зв'язками, створена Джоном Хопфілд, показала нові можливості застосування нейронних мереж. В цей же час Теуво Кохонен надав моделі нейронної мережі, з можливістю навчання без учителя. 4 роки по тому, двома незалежними групами вчених, що складаються з Девіда Румельхарта з Джеффрі

Хінтон і Рональдом Вільямсом, і С.І. Барцева з В.А Охоніним, був заново вивчений і суттєво розвинуто метод зворотного поширення помилки. З цього моменту в науковій сфері знову з'явився інтерес до розвитку технологій нейронних мереж, а завдяки широкому розповсюдженню персональних комп'ютерів, дослідження в цій області вкрай прискорилися.

На сьогоднішній день ми маємо понад двадцять різних моделей нейронних мереж, які отримали застосування в багатьох сферах сучасного життя. Для задач аналізу великої кількості даних, для складноформалізуемих і формалізації задач, для завдань з комплексними алгоритмами і алгоритмами з неявними параметрами, можливе застосування нейронних мереж. При необхідності одночасного аналізу великого числа даних, виділення тенденцій і прогнозування в залежності від отриманої інформації, як, наприклад, на ринку паперів або валютному ринку, нейронні мережі отримали дуже широке поширення.

Однак, не дивлячись на всі можливості нейронних мереж і їх продуктивність, людина в рази більше адаптивен і здатний розпізнавати і обробляти абсолютно формалізації речі, такі як емоції, інтонацію, поведінку. Але штучні нейронні мережі мають значну перевагу перед людським мозком, завдяки здатності обробляти більший обсяг однотипної інформації, через свою структури. Даний процес має перевагу за рахунок паралелізації завдань, розбитих на більш прості функції аж до найпростіших бінарних операцій. Ще одним значущим фактом є те, що нейронні мережі не виробляють занадто складних обчислень і ірраціональних операцій, характерних для послідовних архітектур.

Виходячи з цього, архітектуру нейронної мережі можна характеризувати схемою SIMD (Single Instruction - Multiple Data), тобто виконання однієї функції для безлічі даних. За такою схемою реалізується обчислювальний процес на графічних картах, а отже, архітектура графічної карти повністю підходить для реалізації на ній нейронних мереж. Однак перш ніж перейти до реалізації, слід розглянути її пристрій і алгоритми її роботи.

Структура штучного нейрона аналогічна біологічному. Нейрон має кілька входів, за якими сигнали надходять в сам нейрон. Вхідні сигнали можуть бути отримані як із зовнішнього джерела, так і від інших нейронів. Кожен вхід нейрона має свою вагу, тому що надходить сигнал потрапляє в обробний центр нейрона з деяким коефіцієнтом. Далі, після деяких перетворень, нейрон передає на вихід лише один сигнал. Отриманий вихідний сигнал може надходити або на наступний нейрон, або бути кінцевим результатом роботи для всієї мережі. Дана схема представлена на рисунку 1.1

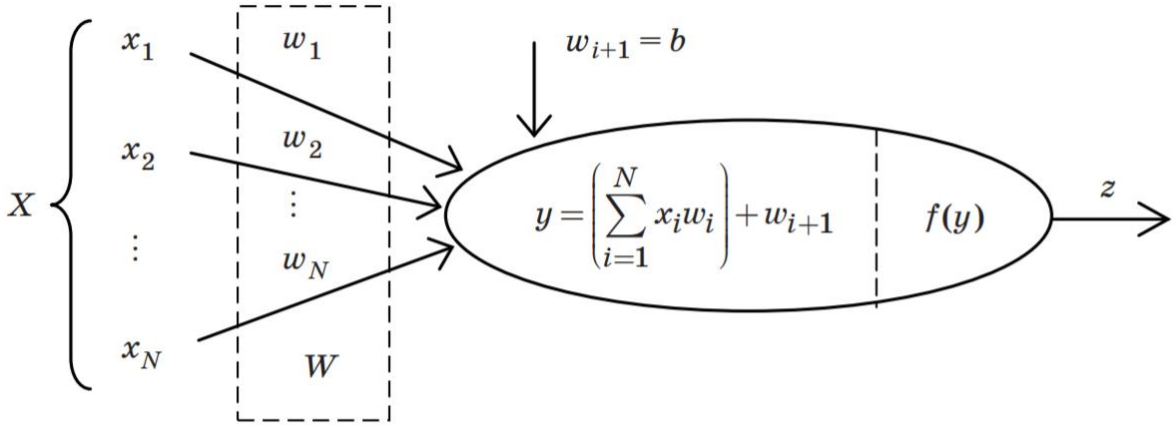


Рисунок 1.1 Схема роботи штучного нейрона

На вхід штучного нейрона надходить деяка множина сигналів. Кожен вхід зважується - множиться на певний коефіцієнт (Синаптичну вагу). Сума всіх добутків визначає рівень активації нейрона.

Активаційна функція F повинна бути монотонною. Зазвичай F (y) належить до інтервалу [0,1] або [-1,1]. найчастіше використовують множину варіантів активаційних функцій.

Таким чином, штучний нейрон виконує дві операції. Спочатку обчислюється сума скалярного добутку вектора ваги W і вхідного вектора X:

$$y = X^T W + b$$

Потім спрацьовує активаційна функція, яка визначає значення вихідного сигналу:

$$z = f(y).$$

Наприклад, якщо використовувати знакову активаційну функцію (див. Табл. 1.2), то вихід ІН можна описати формулою

$$f(y) = sgn(y)$$

Основні варіанти опису активізаційних функцій:

1. Лінійна:

$$F(y) = ky$$

де $k > 0$ – коефіцієнт активації. Має графік вигляду:

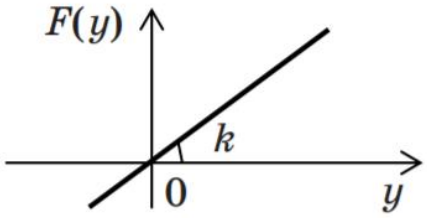


Рисунок 1.2 а). Гарфік лінійної функції активації

2. Лінійна з насиченням:

$$F(y) = \begin{cases} 1, & y > P, \\ ky, & |y| < P, \\ -1, & y < -P. \end{cases}$$

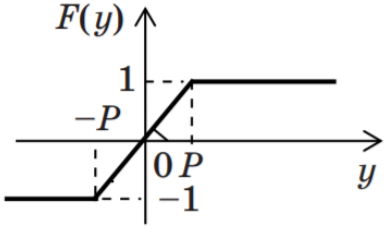


Рисунок 1.2б) Графік лінійної з насиченням функції активації

3. Визначення знака.

$$F(y) = sgn(y)$$

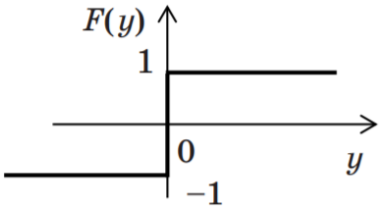


Рисунок 1.2в) Графік функції активації типу визначення знака

4. Уніполярна сигмоїдальна (S-образна):

$$F(y) = \frac{1}{1 + \exp(-ky)}, k > 0,$$

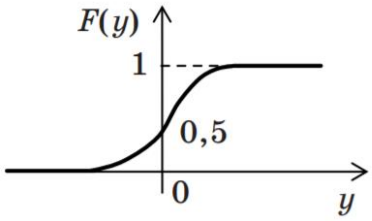
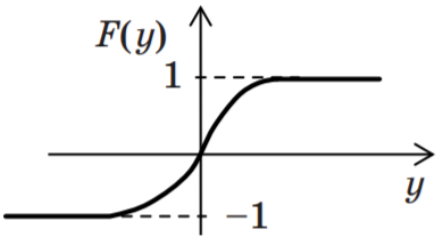


Рисунок 1.2 г) Графік уніполярної сигмоїдальної функції активації

5. Біполярна сигмоїдальна (гіперболічний тангенс)

$$F(y) = th(ky), k > 0$$



Рискнок 1.2г) Графік біполярної сигмоїдальної функції активації

6. Порогова:

$$F(y) = \begin{cases} 1, & y \geq P, \\ 0, & y < P. \end{cases}$$

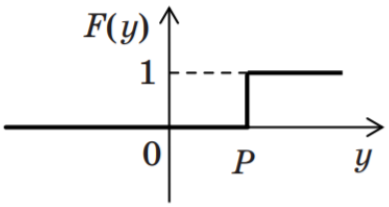


Рисунок 1.2д) Графік порогової функції активації

Для реалізації структури мережі, необхідно ввести зв'язки між нейронами. Основні види зв'язків, які використовуються в нейронних мережах, такі:

а) Прямий зв'язок:

Процес передачі вихідного сигналу відбувається послідовно, з виходу одного шару, на вхід наступного.

б) Зворотній зв'язок:

Вихідні сигнали передаються на вхід попереднього шару, що дозволяє виробляти рекурентні операції на нейронних мережах.

в) Бічна зв'язок:

Вихідні сигнали передаються всередині шару від нейрона до нейрона.

Перш ніж почати створювати нейронну мережу, проводиться довгий аналіз завдання, в процесі якого визначаються топологія і розміри мережі, передавальні і активаційні функції та інші параметри. Завершивши цей етап формалізації і перенесення поставленого завдання на модель нейронної мережі, необхідно почати процес навчання.

Навчання є ключовим етапом під час створення нейронної мережі, і представляє собою процес налаштування параметрів, наприклад вибір ваг синаптичних зв'язків і порогів в активаційних функціях. Прийнято розрізняти 3 види способів навчання:

а) Навчання з учителем;

б) Навчання без вчителя;

в) Навчання за алгоритмом з фіксованими вагами.

Процес навчання з учителем полягає в тому, що нейронної мережі передаються вхідні дані, і набір заздалегідь обчислених вихідних значень. Таким чином, з отриманих значень складаються навчальні пари, а вихідна функція виробляє коригування, щоб з максимальною точністю наблизити реальний результат до очікуваного значенню. Система буде продовжувати навчання до тих пір, поки результати не досягнуть певної точності. Основним

алгоритмом методу навчання з учителем є метод зворотного поширення помилки, що дозволяє рекуррентно коригувати параметри мережі.

Для алгоритму навчання без учителя, характерна передача тільки вхідних даних. Алгоритм є ітеративним, а процес коригування закінчується в разі, якщо однакові входи відповідають однаковим виходів. Прикладом нейронної мережі, яка навчається без учителя, служать самоорганізуються карти.

При навчанні нейронної мережі алгоритмом з фіксованими вагами, мережа визначає ваги за заздалегідь заданими методами розрахунку, обчислюючи їх по вхідних векторів.

В цілому, якщо коротко порівняти алгоритми, то результати, одержувані шляхом навчання без учителя, є більш реалістичними, оскільки в цих процесах не йде порівняння з заданим результатом.

Навчання нейронної мережі витрачає багато часу, і чим складніше структура, тим довше буде проходити навчання. Для прискорення даного процесу існує лише одне рішення - поліпшення самого алгоритму навчання. Найефективнішою, з точки зору витрат часу модифікацією для процесів навчання є розпаралелювання алгоритму.

Після завершення стадії навчання, структура нейронної мережі фіксується, тобто подальших змін ваг і зв'язків не відбувається, мережа переходить в робочий режим.

1.2 Логічні нейронні мережі

Першим кроком для створення нейронної мережі є аналіз обраної завдання, яку дана мережа буде виконувати, тобто по встановленим вимогам вибирається певний стандартний набір функцій. Залежно від поставлених цілей, прийнято класифікувати нейронні мережі. Для систем аналізу зображень, завдань автоматизації управління, економічної сфери, існують типові нейронні

мережі, які далі модифікуються під конкретно поставлене завдання. У даній роботі буде розглянуто клас логічних нейронних мереж.

Для того щоб визначити, в чому полягає суть логічних нейронних мереж, розглянемо приклад завдання, що стоїть перед даною системою. Припустимо, що у нас є деяка безліч висловлювань, за яким шляхом неаріфметіческіх перетворень повинен бути отриманий результат. Для роботи з подібними множинами, необхідно їх формалізувати, представивши всі висловлювання у вигляді предикатів. Так отримуємо можливість обробки предикатів за допомогою операцій алгебри логіки. Рішення в даній задачі залежить від істинності чи хибності висловлень вхідного безлічі. Виходячи з цього, логічну нейронну мережу можна охарактеризувати як нейронну мережу, базисом для якої є булева алгебра.

Продовжимо розгляд поставленого завдання для визначення функціоналу і виду мережі. Якщо взяти сукупність всіх подій, і виділити серед них ті висловлювання, які повністю покривають смисловий діапазон, отримаємо нове безліч. У тому випадку, якщо кожна допустима ситуація характеризується тим, що справжнє значення приймає тільки один вислів цієї сукупності, отримане безліч буде називатися вичерпним безліччю подій. На основі отриманого безлічі, ввівши деяку ієрархію, будемо будувати логічні ланцюжки, що дозволяють однозначно отримати результат. Систему з таких логічних шляхів, в разі її несуперечності і повноти, можна розглядати як систему ухвалення рішень. Однак, перш ніж перейти до побудови нейронної мережі з СПР, необхідний ще один крок у формалізації завдання - визначення величини вхідних сигналів.

Оскільки в мережі, нейрони представлені в деякому стандартному вигляді, то величини збудження, синапсичну зв'язку, пороги і функції повинні бути однакові, ну або лежати в загальному діапазоні значень для всієї мережі.

Після цього кроку, можна приступити до вибору параметрів мережі.

Візьмемо передавальну функцію деякого нейрона:

$$V = \varepsilon \sum_{j=1}^{n} (V_j \omega_{ij} - h_i)$$

де $V_i = \begin{cases} 1, & h_i \geq 1 \\ h_i, & h_i < 1 \end{cases}$; $\varepsilon(\mathrm{x}) = \begin{cases} \mathrm{x}, \mathrm{x} > 0 \\ 0, \mathrm{x} \leq 0 \end{cases}$

Vj - величина сигналу, що надходить на j-й вхід нейрона.

У моделі логічної нейронної мережі розглядаються 2 типу нейронів, що реалізують логічні функції. Це нейрон-кон'юнктор і нейрон-діз'юнктор. (Рис 1.3а, 1.3Б)

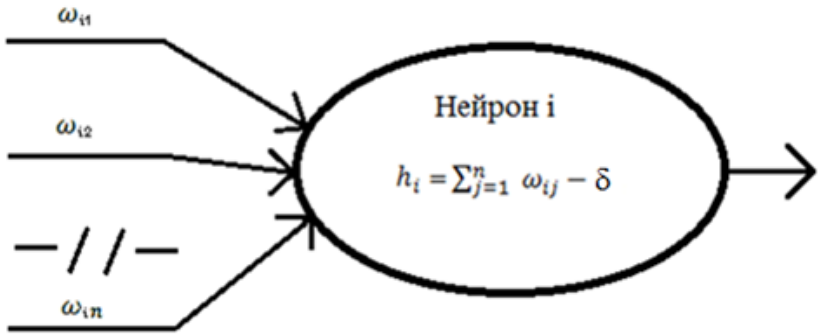


Рисунок 1.3а) Модель нейрона-кон'юнктора

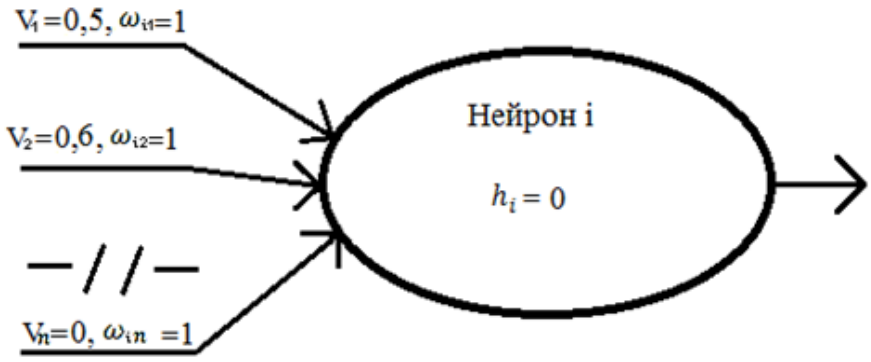


Рисунок 1.3б) Модель нейрона-диз'юнктора

Для реалізації моделі нейрона-кон'юнктор використовується високе значення порога. Значення «δ» - поправка, необхідна для того, щоб для подолання порогу, сигнал надходив по всьому входів.

Модель нейрона-діз'юнктора виконується при низькому значенні порога, але при високому значенні ваг. Ця умова забезпечує отримання вихідного

значення при хоча б одному вхідному сигналі, але в такому випадку необхідно додаткове умова, що обмежує вихідне значення, тобто $V_i \leq 1$. Під час процесу навчання діз'юнктора, передбачається отримання точних вхідних і вихідних даних, а отже логічна функція « АБО »перетворюється в« виключає АБО », оскільки в даному випадку мається на увазі порушення всього на одному вході.

Головною відмінністю логічних нейронних мереж від мереж інших видів є те, що не існує одного шаблону для реалізації всього класу логічних нейронних мереж. Кожна така мережа будується під задачу, шляхом об'єднання в мережі кон'юнктор і діз'юнкторов. Тільки шляхом аналізу конкретної задачі може бути створена логічна нейронна мережа. Таким чином метою даної роботи буде розробка конструктора, що дозволяє створити логічну нейронну мережу.

Проаналізувавши вхідна множина, необхідно виділити всі використовувані змінні, що подаються на вхід, після чого треба побудувати структуру, а потім навчити її. Перша частина навчання відбувається методом трасування, коли по висунутій мережі стандарту вибудовується шлях від вхідних значень до вихідного. При пред'явленні декількох еталонів нейронна мережа будує шляхи послідовно, вибираючи ще не задіяні нейрони і вводячи необхідні зв'язки. Даний алгоритм повторюється до тих пір, поки не буде досягнута повна визначеність для всіх вхідних еталонів, що пред'являються мережі. Для алгоритму трасування нейронної мережі використовується поняття матриці проходження - матриці, в якій рядки і стовпці - нейрони, а значення, що стоять на їх перетині відповідають за наявність або відсутність зв'язку. Алгоритм дозволяє вибудувати зв'язку, ланцюжка проходження з вхідного шару до вихідного через приховані шари. Таким чином, ми отримуємо готову структуру логічної нейронної мережі. Наступним кроком навчання є приведення нейронної мережі після трасування, тобто корекція ваг і порогів для досягнення необхідних результатів. Також, для отримання більш точних результатів, можна використовувати коефіцієнт приведення, який буде

розраховуватися для кожного нейрона. Коефіцієнт «K» буде обчислюватися за такою формулою:

$$K_j = \frac{U_j}{V_j}$$

де в якості значення Uj буде вибрано, наприклад, максимальне значення порушення нейрона, або деяка задовольняє завданню оцінка.

У підсумку, отримана логічна нейронна мережа здатна на основі формалізованих вхідних даних, видати значення шляхом логічних операцій. Таким чином, логічна нейронна мережа, по суті, є автоматизованою системою логічного висновку. Сферою застосування таких мереж є системи управління, банківська сфера, різні системи аналізу, а також система автоматичного доведення теорем, яка грунтується на методі логічного висновку.

1.3 Логічний висновок

1.3.1 Принцип резолюції Робінсона

Математична логіка, як наука, містить в собі безліч підрозділів. Один з таких підрозділів, званий теорією доказів, розглядає можливість подання доказів, як формальних математичних об'єктів, шляхом їх аналізу математичними методами. Теорія доказів є синтаксичним методом, і отримала свій розвиток завдяки роботам багатьох вчених, таких як Г. Фреге, Дж. Пеано. Однак вважається, що більш сучасна теорія доказів починається з робіт Д. Гільберта, К. Геделя і Яна Лукашевича. В результаті теорія доказів була зведена до методів дедукції та природного виведення. Так в 1930 році в докторської дисертації Жака Ербрана вперше з'явилося поняття правила резолюцій - основного механізму для логічного висновку.

Формулювання правила була наступною:

Маючи логічну формулу A, з якої виводиться логічна формула B, то доказова операція імплікації (логічного слідування) A => B.

Дане правило дозволяло аналітичним шляхом встановити існування висновків і доказів, не використовую їх побудови.

Подальшу розробку даного правила, використовувану по сьогоднішній день, провів Джон Алан Робінсон. У 1965 році Робінсон опублікував роботу «Машинно-орієнтована логіка, заснована на принципі резолюції», в якій метод резолюції розглядався більш детально, а також було опис методів побудови логічних систем на основі цього методу. Надалі своєму розвитку, роботи Робінсона стали основою для створення логічних систем, зокрема мови логічного програмування PROLOG.

Суть методу резолюції полягала в наступному: маючи 2 висловлювання, званих резольвіруемимі, необхідно отримати з них нове вираз, яке буде містити всі літерали початкових висловлювань, за винятком взаємно зворотних литералов. Тобто, маючи в якості вихідних даних висловлювання на кшталт A1 = P v B'1 і A2 =! P v B'2, необхідно отримати резольвенту B'1 v B'2. Знак «!» означає логічне заперечення. Правило виводу даного виразу називається правилом резолюцій. Дане правило застосовується для доведення теорем. Відбувається це за наступним алгоритмом:

а) Формулювання:

Нехай деяка формула H - гіпотеза теореми, є логічним наслідком деякого безлічі формул F1, ..., Fk. Або іншими словами, «якщо F1, ..., Fk - істинні, то істинна H».

б) Застосування методу:

1) Створимо нову множину формул {F1, ..., Fk, H}

2) Наводимо кожну формулу до КНФ, прибираємо знаки кон'юнкції, отримуємо безліч диз'юнктів S.

3) Шукається висновок порожнього диз'юнкт, попарним застосуванням правила резолюцій, тобто шляхом виключення взаємно зворотних литералов.

в) Результат методу:

Якщо після застосування методу до безлічі S, був отриманий порожній диз'юнкт, то формула H є логічним наслідком з безлічі формул F1, ..., Fk. В

іншому випадку, якщо порожній диз'юнкт не була отримана, то H не є наслідком з формул F1, ..., Fk.

В цілому, метод резолюції є методом докази «від протилежного», тобто спочатку взявши заперечення теореми, в кінці застосування методу ми отримуємо порожній диз'юнкт, тобто протиріччя, а отже, початкове заперечення було помилковим, звідки слід істинність теореми.

На основі методу резолюцій базується величезна безліч логічних систем і засобів логічного висновку. У логічному програмуванні даний метод застосовується в сукупності з одним з методів міркування - прямим або зворотним, правилами «modus ponens» і «modus tollens».

Основним недоліком методу резолюції є породження великої кількості нових резольвент, які найчастіше виявляються зайвими, чи не приводять до шуканого результату.

1.3.2 Паралельний метод резолюції

Незважаючи на універсальність, значимість і застосовність методу резолюції, сформульованого Робінсоном, він був недостатньо ефективний в плані тимчасових витрат. Повний перебір всього безлічі диз'юнктів і породжуваних додатково резольвент, приводили до того, що навіть на найбільш сильних ЕОМ того часу, виконання алгоритму відбувалося занадто довго.

З кінця 1970-х - початку 1980-х років, ідеологія в сфері комп'ютерної індустрії кинулася в нове русло - створення потужного суперкомп'ютера з функціями штучного інтелекту. Даний проект отримав найбільший пріоритет в Японії, оскільки тоді була складена урядова програма, тобто дані розробки були стратегією розвитку технологій для всієї країни. Оскільки суперкомп'ютер повинен був володіти штучним розумом, в нього необхідно було закласти основи роботи з логікою, методики обробки даних і логічного висновку. Основою для логічного висновку став метод резолюції, однак технічні можливості кілька випереджали сам метод, тобто необхідна була модифікація методу для досягнення максимальної ефективності. Головним завданням було -

зробити паралельні алгоритми, які підходять під структуру комп'ютера з паралельними процесорами.

Для реалізації паралельної системи логічного висновку, Ехуд Шапіро в другій половині 1980-х років розробив новий вид мови логічного програмування PROLOG, названий «Flat Concurrent PROLOG», в якому замість послідовної обробки логічної інформації, використовувався паралельний підхід. Розпаралелювання алгоритмів логічного висновку дозволило добитися істотного скорочення тимчасових витрат. Якщо колишні, послідовні версії, для функції часу від кількості даних видавали значення, пропорційні O (log n), то нововведеного паралельні методи дозволили домогтися залежності виду O (n / (log n) 2), що означало суттєвий прорив у цій технології.

Як мова програмування, паралельна реалізація Прологу, була мовою високого рівня, що забезпечує паралельну роботу з робочою даними. В основі логічного висновку лежав модернізований, розпаралеленого метод резолюції, зроблений Ехудом Шапіро. Ідея методу була проста - замість послідовного перебору всіх резольвіруемих диз'юнктів і резольвент, необхідна була паралельна обробка відразу всієї безлічі. Завдяки динамічному паралелізму процесів, породження нових резольвент не була проблемою для даної реалізації методу.

Алгоритм у своїй суті не відрізнявся від послідовного. Різниця була в паралельному виборі і обробці диз'юнктів. Розглянемо сам паралельний алгоритм:

а) Формулювання залишилася такою ж, як і при послідовному випадку, тобто «якщо формули F1, ..., Fk - істинні, то істинна гіпотеза H».

б) Застосування методу:

1) Також відбувається об'єднання множини формул {F1, ..., Fk, H}

2) Приводимо кожну формулу до КНФ, прибираємо знаки кон'юнкції, отримуємо множину диз'юнктів S.

3) Шукається висновок порожнього диз'юнкта, попарним застосуванням правила резолюцій. Кожна пара обробляється паралельно наступним чином:

для всіх індексів i, j, де i, j∈ [1, n], i ≠ j, створювалися пари (Fi, Fj), до яких застосовувалося правило резолюції.

4) Після виконання одного паралельного кроку, з'являлися нові резольвенти і додавалися до безлічі S, після чого для них також шикувалися пари, і застосовувалося правило резолюції.

в) Результат методу:

Результат застосування алгоритму залишився таким же, як і раніше, тобто в залежності від отримання порожнього диз'юнкт, судилося про істинність або хибність поставленої теореми.

Висновки

У першому розділі даної роботи були розглянуті системи аналізу даних, що називаються нейронними мережами, основні історичні етапи та основи створення нейронних мереж, а також сучасні нейронні мережі та їх придатність. Структуру нейронної мережі можна описати схемою SIMD, що означає виконання однієї функції на безлічі різних даних, отже для будь-яких завдань, в яких потрібно аналіз великої кількості даних одного типу, дана структура дозволяє распараллелить процеси, що відбуваються всередині мережі та значно збільшити швидкість роботи. Існує безліч різних топологій нейронних мереж, що застосовуються в залежності від поставлених завдань. У даній роботі розглядаються логічні нейронні мережі.

Основуючись на алгебрі логіки, логічні нейронні мережі можуть обробляти логічні предикати, які є формалізованим способом опису ситуації через висловлювання. Отже, такі нейронні мережі дозволяють оцінювати ситуацію, що склалася і дати вказівки до дій.

Логічні нейронні мережі не мають загального шаблону, і кожна така мережа будується в залежності від поставленого завдання. Шляхом аналізу вхідних даних, що використовуються формул і поставленої мети, логічна

нейронна мережа створюється у вигляді структури з кон'юнктор і діз'юнкторов, об'єднаних зв'язками, отриманими з умов завдання. Спираючись на булеву алгебру, логічні нейронні мережі покривають широкий спектр завдань, а також здійснювати логічний висновок типу «причина-наслідок», що застосовується для різних систем управління, а також для систем автоматичного доведення теорем.

Правило резолюції, сформульоване Жаком Ербраном в 1930-му році, і доопрацьоване Робінсоном в 1965-му, дозволило приступити до початку створення комп'ютерних систем логічного висновку і штучного інтелекту. Параллелизация методу резолюції має велике значення. Оскільки методи паралельного програмування є новим, більш сучасним підходом в сфері комп'ютерних технологій, в тому числі для реалізації різних аналітичних систем, систем прийняття рішень і реалізацій нейронних мереж, то створення паралельного алгоритму логічного висновку на основі модифікації Шапіро для методу резолюції, дозволяє значно збільшити швидкість роботи будь-якої логічної системи.

Логічні нейронні мережі отримали можливість паралелізму для обробки даних, що значно збільшило популярність і застосовність в різних сферах, завдяки високій швидкодії.

У даній роботі логічна нейронна мережа буде реалізовувати функцію перевірки логічного слідування, замість виконання логічного висновку. Однак, якщо при виконанні логічного слідування будуть отримані істинні значення, можна говорити про логічне виведення.

2. Побудова логічної нейронної мережі для задачі логічного слідування

2.1 Реалізація конструктора мережі і підстановки значень

У першому розділі роботи говорилося про те, що кожна логічна нейронна мережа будується «під завдання», шляхом аналізу поставленої мети і вхідних даних.

Завданням в даній роботі стоїть розробка логічної нейронної мережі, що виконує операцію імплікації для всіляких наборів вхідних значень. Таким чином, через варіації видів вхідних диз'юнктів і експоненціального зростання кількості наборів значень, необхідний конструктор, що дозволяє створити відповідну структуру мережі. Тоді, першою частиною завдання є введення вхідних диз'юнктів і їх аналіз, на основі якого буде створюватися структура мережі.

Розглянемо алгоритм, що:

1) Вводиться множина формул.

2) Проводиться посимвольний аналіз для отримання розміру вхідного вектора

3) Також посимвольним аналізом, встановлюються залежності між використовуваними перемінними і їх композиціями у введений формулах

Формули вводяться вручну, після чого обчислюється загальна кількість різних змінних, використовуваних у всьому введеному безлічі, а так само наявність зворотних змінних, до яких застосований оператор заперечення. Далі оцінюється загальна кількість використовуваних формул і для кожної з введених формул розглядається використовується безліч змінних. Будується динамічний двовимірний масив, розмірності якого - кількість змінних і використаних формул.

Первинно в масив заносяться значення 1,0 і -1, для позначення використання змінної, відсутності змінної і використання зворотного значення

змінної відповідно. Дана структура зображена на малюнках 2.1А і Б, де 1-й стовпець масиву - порожній, який використовується в подальшому для приміщення в нього вхідних значень. 2-й стовпець відповідає за використання зворотних значень. Стовпці, починаючи з 3-го, відповідають за введені функції. Під матрицею відображений масив, що містить інформацію про кількість використовуваних змінних для кожної формули.

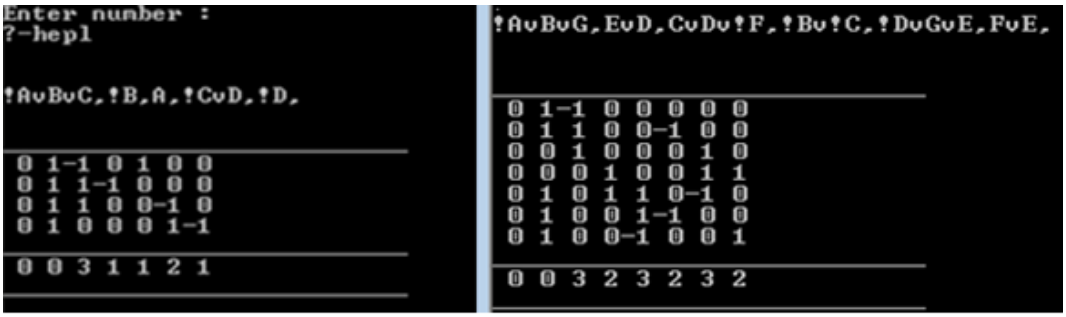
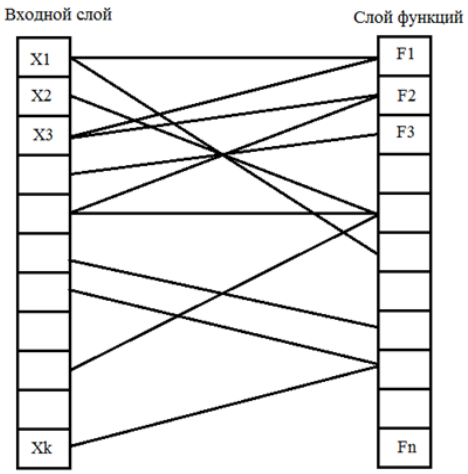


Рисунок 2.1 А Рисунок 2.1Б

Таким чином, реалізований алгоритм дозволив побудувати первинну структуру мережі. Масив є динамічним, тобто дозволяє створювати структуру мережі для будь-якої кількості формул і змінних, що показано на малюнках 2.1А, Б.

Кожен стовпець, починаючи з 3-го, є нейроном, які реалізують вхідні функцію. Якщо уявити цей масив у вигляді мережі, то ці стовпчики знаходяться в одному шарі.



Як було сказано раніше, перший стовпець залишається порожнім, а потім в нього записуються значення наборів змінних. Набір значень змінних є безлічі нулів з одиниць. Приклад зображений на рисунку 2.2

Рисунок 2.2 Приклад вхідних значень

Далі значення з кожного стовпчика підставляється замість першого стовпчика в матриці структури мережі, і для кожного з стовпців відбувається операція підстановки вхідних значень в кожну з використаних формул, з урахуванням заперечень змінних в формулах.

Даний спосіб підстановки значень дозволяє отримати значення для кожної з використаних функцій незалежно, після чого залишиться останній крок роботи мережі - твір аналізу отриманих значень і перевірки істинності проходження останньої формули, з усіх попередніх введених формул.

2.2 Можливості паралелізації використаних алгоритмів

Розглянуті алгоритми, для підвищення ефективності їх роботи, можуть бути реалізовані паралельно, що істотно скоротить тимчасові витрати. Для паралелізації буде використовуватися технологія, що дозволяє виконувати одночасно кількість операцій дорівнює кількості доступних ядер в процесорі графічної карти.

Першим застосуванням паралельних обчислень буде занесення значень змінних в масив. Замість почергової передачі кожного нуля або одиниці, може бути реалізована передача всього стовпчика, що скорочує кількість операцій в

«n» раз. Однак, більш ефективним буде розгляд паралельної передачі всіх вхідних значень. Тоді кількість передач буде обмежено тільки кількістю ядер.

Головним завданням в цьому розділі є паралельний спосіб обчислення введених функцій. Оскільки для «n» змінних ми отримаємо 2n різних наборів вхідних значень, максимально витратним є саме обробка всього цього безлічі. Оскільки обмеження кількості операцій полягає тільки в потужності пристрою, на якому буде реалізована робота даної мережі, можливо отримати зменшення кількості операцій завдяки паралельної організації обчислень, аж до одного кроку.

Використання графічної карти, як обчислює пристрої є найбільш оптимальним, оскільки навіть в слабких графічних картах нараховуються десятки ядер, що мають можливість паралельної роботи. У передових графічних картах кількість ядер досягає декількох тисяч, завдяки чому швидкодія може бути збільшено в рази.

Паралельна передача безлічі вхідних значень в структуру мережі є одним з розглянутих способів паралелізації. Крім паралельної передачі безлічі значень в структуру, можна виконати паралельну обробку значень для кожної функції.

Спосіб паралельної передачі вхідних даних зображений на рисунку 2.3, паралельна передача значень в кожну функцію зображена на рисунку 2.4.

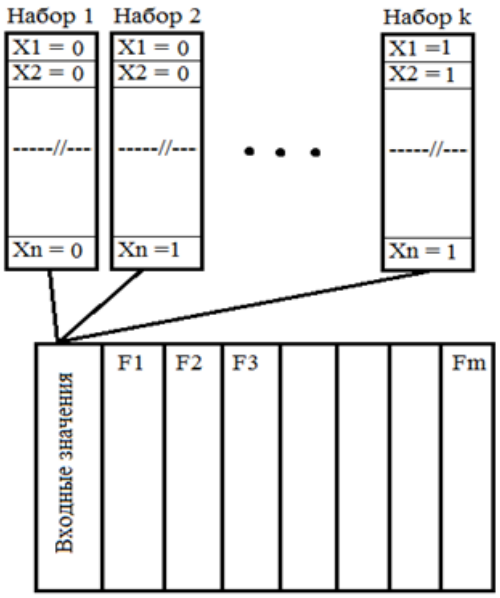


Рисунок 2.3 Параллельна передача значень

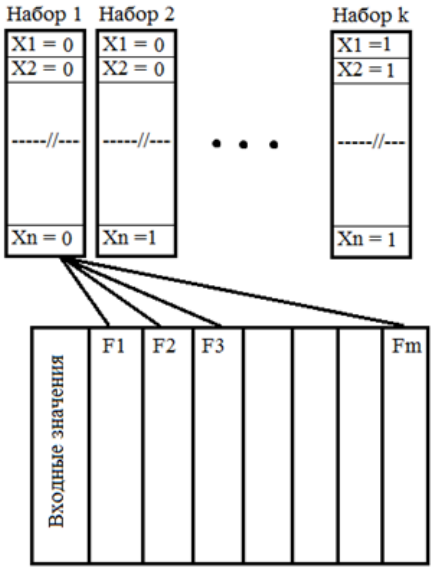


Рисунок 2.4 паралельного виконання кожної функції для одного набору

Для досягнення максимальної продуктивності і ефективності роботи, необхідно комбінувати 2 даних способу. Паралельно передаючи кожен з наборів вхідних значень на кожну функцію, ми позбавляємося від послідовних операцій, отримуючи тим самим незалежність від розмірів вхідного вектора і кількості функцій. Даний спосіб зображений на рисунку 2.5.

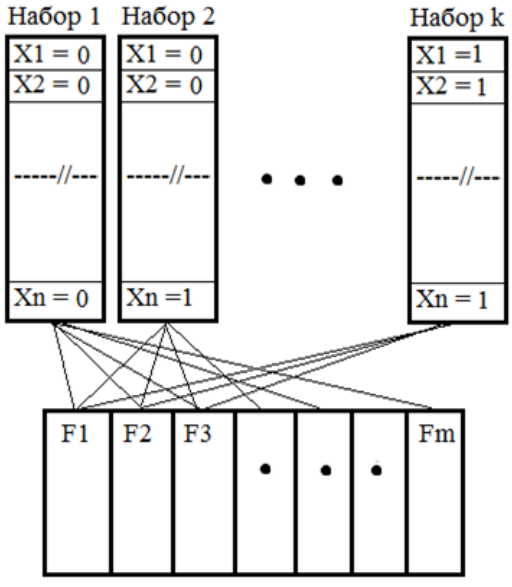


Рисунок 2.5 Повна параллелизация завдання

Єдиним обмеженням в такій ситуації буде кількість задіяних ядер графічної карти. У випадках, коли число паралельних операцій є меншими від кількості ядер, для такої реалізації буде потрібно всього один паралельний крок. Однак, при великій розмірності вхідного вектора або при великій кількості функцій, з урахуванням експоненціального зростання кількості вхідних наборів, а також, при паралельному обчисленні кожної функції, ми отримуємо неможливість виконання всієї перевірки за один крок. В такому випадку необхідно виробляти виконання паралельних операцій в циклі, охоплюючи максимально можливий діапазон значень і функцій, що не перевищує кількості можливих паралельних операцій. Вибиратися буде або кількість вхідних значень для виконання на всіх функціях, або на деяких з них.

В такому випадку, графічне представлення буде наступним.

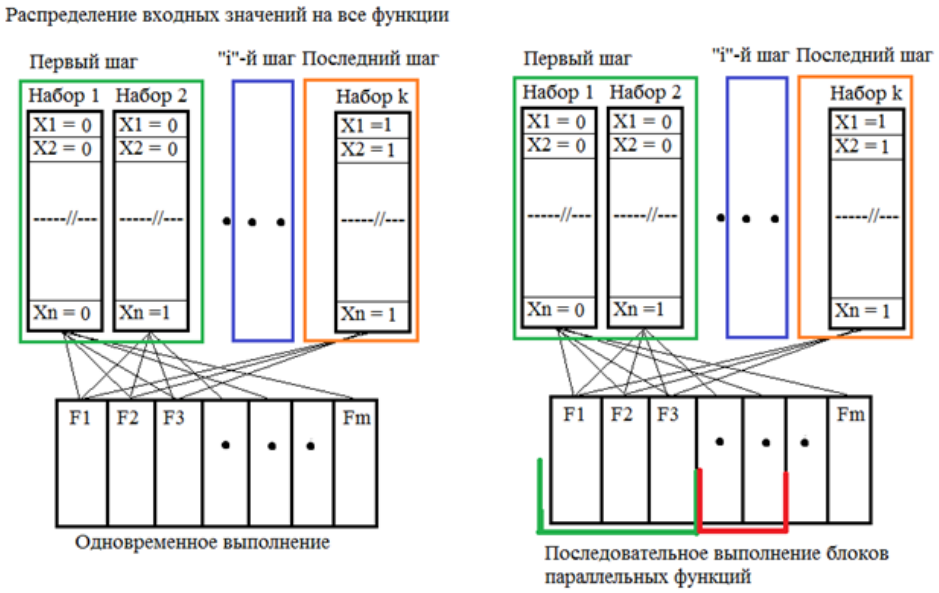


Рисунок 2.6 Розбиття паралельних операцій

Дана організація обчислень дозволить створити логічну нейронну мережу, яка не буде залежати від продуктивності системи, на якій вона буде запущена.

Висновки

Алгоритм динамічного побудови логічної нейронної мережі є основним, оскільки даний тип нейронних мереж не має стандартних шаблонів, і кожна така мережа будується під конкретну задачу.

Кількість різних наборів вхідних значень зростає залежно від кількості різних змінних за формулою 2n, що означає при невеликих кількостях змінних величезні розміри вхідних масивів, а отже, обробка цих значень буде вкрай витратною за часом. Тому ефективно використовувати паралельні алгоритми для реалізації даних обчислень.

Паралельне обчислення кожної з функцій для кожного набору вхідних значень дозволить домогтися максимальної швидкодії роботи мережі. Однак максимальне число паралельних операцій обмежена пристроєм, на якому виконується робота, отже, необхідно ввести додаткові умови для оптимізації роботи.

Для оптимальної реалізації даної структури необхідно пристрій з великою кількістю ядер. Максимально підходить для обчислень в даній задачі архітектура це використання графічної карти. Таким чином, застосування технології CUDA є основною умовою для оптимальної реалізації структури логічної нейронної мережі.

3. Застосування технології CUDA для паралельної реалізації логічної нейронної мережі

3.1 Історія, структура

Технології обробки інформації постійно розвиваються, для досягнення більшої ефективності нової архітектури. Так в 2003 році було запропоновано використовувати процесор графічної карти для паралелізації алгоритму обчислень, шляхом передачі оброблюваних даних з центрального процесора на процесор відеокарти. 3 роки по тому, одна з передових компаній, що виробляють графічні карти втілило нову архітектуру побудови процесора графічної карти, що дозволяє задіяти її в якості співпроцесора. Технологія була названа CUDA, що було абревіатурою від «Compute Unified Device Architecture». Вперше дана архітектура була втілена в графічних картах серії Nvidia GeForce 8800 gtx. У картах цієї моделі використовувався чіп восьмого покоління G80, який в подальшому був присутній у всіх наступних графічних картах.

Технологія CUDA є не тільки нову архітектуру пристрою графічної карти. У це поняття також входить набір програм і надбудов, що дозволяють створювати свої додатки на різних мовах програмування, використовуючи паралелізм графічної карти. Мовами, що підтримують дану технологію є Fortran і C / C ++. Для того, щоб почати роботу з даною технологією, необхідно завантажити з офіційного сайту компанії Nvidia відповідний пакет програм, названий CUDA Toolkit, і встановити його.

В основі інтерфейсу створення додатків CUDA лежить мова C, з деякими невеликими змінами, а також безліч бібліотек, що підключаються для використовуваного середовища програмування. Для реалізації програм, що використовують CUDA, створений компілятор на мові «Сі», названий nvcc, що дозволяє обробити код програми і розподілити його в залежності від використовуваної технології.

Архітектура CUDA використовує модель пам'яті «Grid → Block → Thread», що дозволяє незалежний паралельний доступ до пам'яті відеокарти. Дана структура зображена на рисунку 3.1.

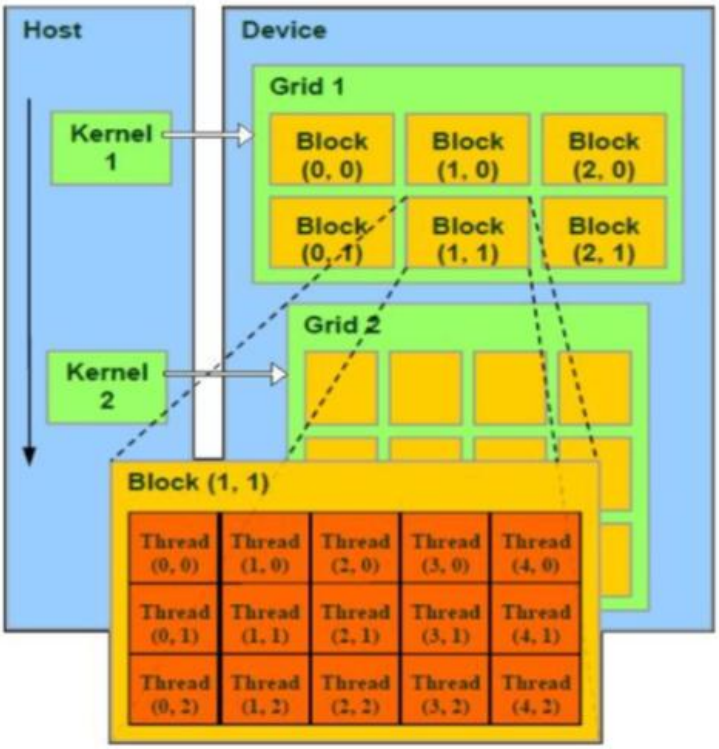


Рисунок 3.1 Структура пам'яті в графічній карті

На цій ілюстрації присутні такі поняття:

1) Host - центральний процесор використовуваної ЕОМ.

2) Kernel - ядро ЦП.

3) Device - графічна карта

4) Grid - розмічена сітка, масив з блоків пам'яті.

5) Block - блок пам'яті, масив, який об'єднує кілька ниток

6) Thread - нитка, покажчик на конкретне місце в пам'яті графічної карти.

Кожному блоку відповідає одне ядро графічної карти, що дозволяє домогтися високого паралелізму при використанні обчислень на графічній карті. У сучасних графічних картах Nvidia Tesla кількість таких ядер налічує близько п'яти тисяч, що означає можливість використання до 5 тисяч паралельних операцій. Завдяки синхронізованої бар'єрної пам'яті, не існує проблем з використанням однієї ділянки різними процесами, а отже дозволяє

уникнути безлічі помилок. В цілому, систему пам'яті в CUDA можна зобразити, як на рисунку 3.2

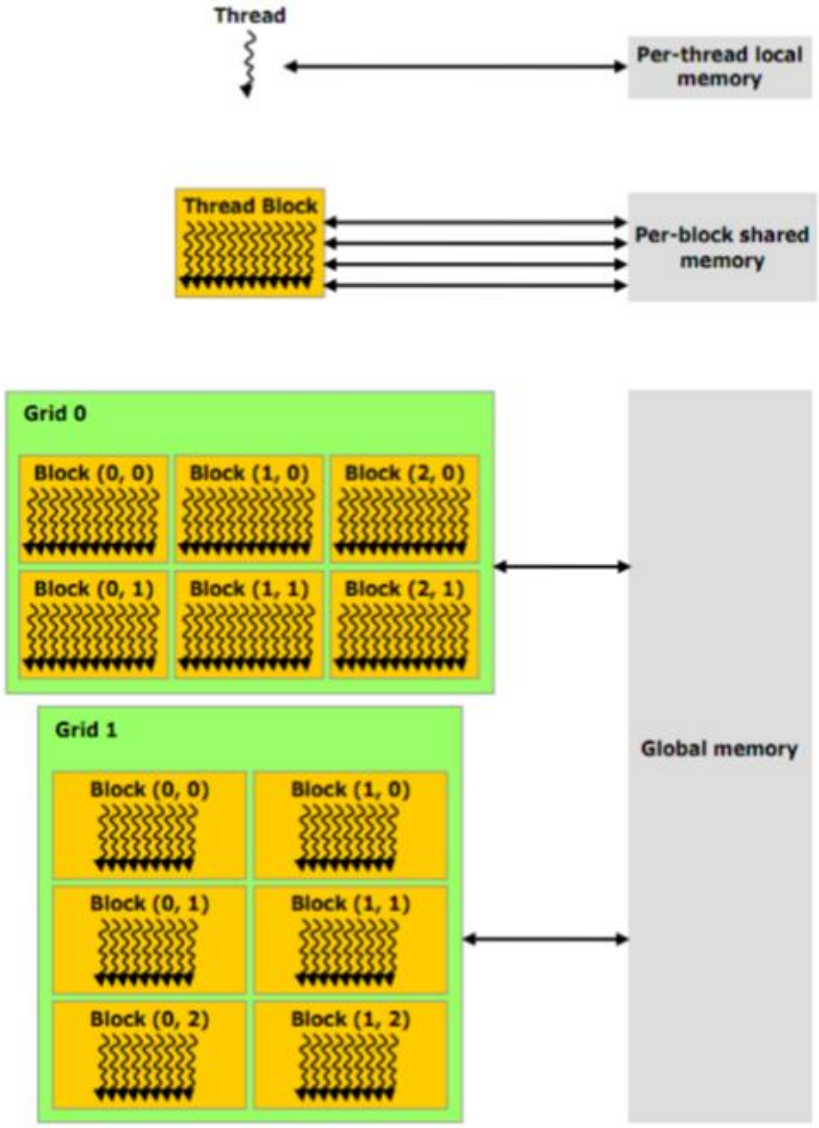


Рисунок 3.2 Організація пам'яті в CUDA

3.2 Застосування

Технологія CUDA набула широкого поширення на сьогоднішній день. Оскільки в більшості сфер не потрібно складних, витратних з точки зору продуктивності ЕОМ, операцій, то вигідніше провести виконання таких простих процесів не на центральному процесорі, тим самим звільнивши його, і надавши можливість обробки складніший операцій. Таким чином, вигідно

використовувати графічні карти для подібних простих обчислень, особливо при необхідності реалізації схеми SIMD, для одночасної обробки великої кількості однотипних даних.

Крім створення різних аналітичних систем, технологія розпаралелювання обчислень на ядра графічної карти використовується для отримання надпотужних суперкомп'ютерів. Подібні розробки в Японії і Китаї дозволили домогтися продуктивності в десятки гігафлоп.

Нейронні мережі, що використовують архітектуру CUDA, дозволяють проводити процес навчання і обробки інформації в десятки разів швидше, ніж при послідовних операціях на центральному процесорі. Наприклад, для систем аналізу зображення, замість послідовної обробки кожного пікселя, можливий аналіз всього зображення, або його великих фрагментів.

Для реалізації логічної нейронної мережі також можливе застосування паралельної архітектури CUDA для навчання, і для подальшої роботи. Якщо розглядати реалізацію логічної нейронної мережі, як засобу логічного висновку і автоматичного доведення теорем, то для застосування паралельного методу резолюції Шапіро, необхідна паралельна структура, яка дозволить одночасно обробляти все вхідна множина диз'юнктів. Таким чином технологія CUDA по всіх параметрах підходить для даної реалізації.

3.3 Реалізація паралельного алгоритму

Розглянуті у другому розділі алгоритми створення мережі і обробки вхідних даних можуть бути реалізовані за допомогою паралельної архітектури CUDA. Як було сказано, найбільш витратним процесом є передача безлічі значень для вхідних змінних, і обчислення функцій їх використовують. Розберемо, як саме технологія CUDA дозволить распараллелить цей процес.

Перш ніж почати виконання обчислень, при використанні CUDA, необхідно виділити пам'ять під всі використовувані значення. Це робиться

функцією «cudaMalloc». Для використання цієї функції необхідно створити покажчик на початок ділянки, в який будуть поміщені використовувані дані, а також розмір цих даних в байтах. У реалізації це буде виглядати наступним чином:

Int * dev_ptr = 0;

cudaMalloc ((void **) & dev_ptr, sizeof (int) * N), де N - кількість використовуваних значень.

Після виділення пам'яті, в вказане місце необхідно помістити самі значення, для чого використовується функція

«CudaMemcpy (dev_ptr, input, N * sizeof (int), cudaMemcpyHostToDevice);» , де

dev_ptr - наш покажчик,

input - передане значення,

N * sizeof (int), - розмір в байтах,

cudaMemcpyHostToDevice - напрямок передачі даних, HostToDevice означає передачу з центрального процесора (Host) на процесор графічної карти (Device).

Якщо кількість вхідних значень перевищує кількість ядер на графічній карті, необхідна реалізація циклу, що відповідає за часткову передачу значень.

Після того, як дані були передані в пам'ять графічної карти, необхідно запустити «device» - функцію, що виконує задані нами операції на процесорі відеокарти. Для цього використовується запис наступного типу:

«AddKernel <<< grid, threads >>> (dev_ptr, dev_array);», де

addKernel - ім'я функції,

grid - кількість використовуваних блоків пам'яті, ціле число

threads - кількість ниток в кожному блоці, ціле число

dev_array - масив, в якому записана структура мережі і всі введені спочатку функції.

Запис виду <<< A, B >>> (dev_ptr, dev_array) є вказівкою компілятору виконати функцію на графічній карті, а не на центральному процесорі. У дужках вказуються передані параметри.

Сама функція, виконувана на графічній карті має наступний прототип:

__global__ void addKernel (int * dev_ptr, int * dev_array);

Далі, кожне передане значення необхідно пронумерувати. Робиться це шляхом вказівки до яких блоку і нитки відноситься дане значення. Введемо наступне позначення:

Int i_x = blockIdx.x + threadIdx.x;

Оскільки використовувані нами масиви є двовимірними, необхідно ще одне позначення:

Int i_y = blockIdx.y + threadIdx.y;

Так ми отримуємо унікальні номери для кожного з використовуваних значень а отже можемо паралельно обчислити всі функції.

Висновки

У даній роботі були розглянуті системи аналізу, звані нейронними мережами, а зокрема клас логічних нейронних мереж. Використовуючи алгебру логіки як базис, даний тип мереж дозволяє вирішувати різні завдання, в тому числі трудноформалізуемие і формалізації. До таких завдань відносяться роботи з висловлюваннями, предикатами, для яких використовується формалізація даних для отримання значень істинності чи хибності.

Оскільки логічні нейронні мережі, будучи одним з видів всього безлічі нейронних мереж, не мають шаблонів для використання, а кожна така мережа будується під конкретну задачу, було прийнято рішення створити алгоритм, що дозволяє динамічно вибудовувати структуру логічної нейронної мережі. Даний конструктор мережі є універсальним, оскільки мережа вибудовується в залежності від введених умов, отже, ми отримуємо автоматично масштабується мережу під кожну задачу. Використання реалізованого алгоритму є доказом того, що неможливо реалізувати шаблонну логічну нейронну мережу, яка підходить під будь-які завдання даного класу.

Реалізована мережу виконує перевірку на логічне слідування, шляхом перебору всієї множини значень змінних, що є дуже витратним за часом процесом. Однак параллелизация даного алгоритму дозволяє домогтися збільшення швидкодії в кілька разів, що є основною метою даної роботи.

Розгляд різних паралельних архітектур привело до висновку, що оптимальною технологією для реалізації використовуваних алгоритмів є використання графічної карти для паралельних обчислень, що не тільки збільшує швидкодію системи, але і дозволяє звільнити центральний процесор від зайвого навантаження і залишити його для виконання фонових програм або складних операцій, вимогливих до продуктивності системи.

Технологія паралельних обчислень на графічній карті CUDA, випущена компанією Nvidia в 2007 році, на сьогоднішній день є однією з передових технологій обчислень на графічній карті. Використання паралелізму в багатьох сферах дозволяє досягати поставлених цілей в рази швидше, ніж при використанні послідовних обчислень на центральному процесорі.

Використання технології CUDA в даній роботі дозволило реалізувати алгоритми роботи логічної нейронної мережі. Завдяки спеціальній структурі пам'яті, при використанні технології CUDA можливе одночасне обчислення до декількох тисяч процесів. Так, перебір всіх вхідних значень змінних може бути змінений з послідовного на паралельне, завдяки чому кількість операцій буде обмежено тільки кількістю ядер в графічній карті. Таким чином, застосування технології CUDA дозволяє домогтися максимальної ефективності роботи від реалізованої програми.

Реалізована мій в процесі даної роботи логічна нейронна мережа дозволяє провести операцію логічного слідування для безлічі вхідних формул. Динамічне побудова структури відкриває безліч варіантів для використання мережі. Шляхом нескладних модифікацій, програма може бути змінена для реалізації перевірки множин істинності логічних функцій. Одним з очевидних прикладних застосувань даної програми є рішення логічних задач, наприклад, типових завдань з інформатики.

В цілому, застосовність даної програми дуже висока, так як за рахунок динамічної масштабованості можлива реалізація систем прийняття рішень, управління та аналізу даних.

Список літератури

1 McCulloch W.S., Pitts W. A logical calculus of the immanent in nervous activity // Bulletin of mathematical biophysics. 1943. V5.

2 Nordstrom T. Designing parallel computers for self-organizing maps. Forth Swedish Workshop on Computer System Architecture. 1992.

3 Барський А.Б. Логічні нейронні мережі. НОУ «ІНТУЇТ», 2016.

4 Хайкін С. Нейронні мережі: повний курс, 2-е изд., Испр. - М .: Видавничий дім «Вільямс», 2006. - 1104 с.

5 Чень Ч., Лі Р. Математична логіка і автоматичне доведення теорем. М .: Наука. Головна редакція фізико-математичної літератури, 1983.

6 Shapiro E.Y. Concurrent prolog: Collected papers. Cambridge. MIT Press, 1987

7 Боголюбов Д.П., Чанкін А.А., Стеміковская К.В. Реалізація алгоритму навчання самоорганізованих карт Кохонена на графічних процесорах // Промислові АСУ та контролери. 2012. № 10. С. 30-35.

8 Офіційний сайт компанії Nvidia присвячений технології CUDA.

URL: https://developer.nvidia.com/cuda-zone (дата звернення 20.01.2016)

9 Сандерс Дж., Кендрот Е. Технологія CUDA в прикладах: введення в програмування графічних процесорів: Пер. з англ. - М.: ДМК Пресс, 2011 року.

Додаток 1

Вихідний код програми на с ++.

файл main.cpp

```
#include <iostream>
#include <string>
#include <vector> // підключаємо заголовок списку
#include <iterator>
#include <stdlib.h>
#include <math.h>
#include "kernel.cu"

using namespace std;

int main ()
{

char B = '0';
int e = 0;
string s;
while (e == 0)
{
vector <string> VS;
std :: cout << "Enter number:" << '\ n';
std :: cin >> B;
std :: cout << '\ n';
if ((B == 'S') || (B == 's'))
{
while (s! = ".")
{
```

```
std :: cin >> s;
VS.push_back (s);
}
s = '0';
VS.pop_back ();
}
if ((B == 'E') || (B == 'e'))
{
e ++;
}
int PC = 0, NC = 0;
int ind [27];

string tot_elms, neg_elms;
for (unsigned int i = 65; i <91; i ++)
{
ind [i - 65] = i;
// cout << ind [i-65] << "," << (char) i << '\ n';
}



if (VS.begin ()! = VS.end ())
{

copy (VS.begin (), VS.end (), ostream_iterator <string> (cout, ","));
std :: cout << '\ n';
for (unsigned int i = 0; i <VS.size (); i ++)
{
for (unsigned int j = 0; j <VS [i] .size (); j ++)
{
```

```
if (((int) VS [i] [j]> = 65) && ((int) VS [i] [j] <91))
{
if (j> 0) {
if (VS [i] [j-1] == '!')
{
neg_elms = neg_elms + VS [i] [j];
NC ++;
}
}
for (unsigned int DN = 0; DN <26; DN ++)
{

if (((int) VS [i] [j] == ind [DN]) && (int (VS [i] [j]! = 0)))
{
tot_elms = tot_elms + VS [i] [j];
ind [DN] = 0;
PC ++;

}
}
}
}
}
```

// cout << PC << "<+>" << tot_elms << '\ n' << NC << "<+>" << neg_elms << '\ n'; // отримав кількість вхідних нейронів, безліч неотр елеентов і тих, для яких необхідно заперечення

```
// std :: cout << '\ n' << neg_elms.length () << "====" << tot_elms.length () <<
"===" << VS.size () + 2 << '\ n';

int TL = tot_elms.length ();
int LS = VS.size () + 2;
int ** array = new int * [TL];
for (int i = 0; i <TL; i ++) {array [i] = new int [LS];}

for (unsigned int i = 0; i <tot_elms.length (); i ++)
{
for (unsigned int j = 0; j <VS.size () + 2; j ++)
{
array [i] [j] = 0;
}

}
// допоміжна матриця для побудови структури роботи

for (int i = 0; i <tot_elms.length (); i ++)
{
for (int j = 0; j <neg_elms.length (); j ++)
{
if (tot_elms [i] == neg_elms [j])
{
array [i] [1] = 1;
}
}
}

// стовпець, що відповідає за наявність зворотних ел-тів
```

```
int * NOITN = new int [LS]; // number of inputs to neuron;
for (unsigned int i = 0; i <LS; i ++)
{
NOITN [i] = 0;
}
for (unsigned int i = 0; i <VS.size (); i ++)
{
for (unsigned int j = 0; j <VS [i] .size (); j ++)
{
for (unsigned int k = 0; k <tot_elms.length (); k ++)
{
if (j == 0) {
if (tot_elms [k] == VS [i] [j])
{
array [k] [i + 2] = 1; NOITN [i + 2] ++;
}
}
if (j> 0) {
if ((tot_elms [k] == VS [i] [j]) && (VS [i] [j - 1]! = '!'))
{
array [k] [i + 2] = 1; NOITN [i + 2] ++;
}
if ((tot_elms [k] == VS [i] [j]) && (VS [i] [j - 1] == '!'))
{
array [k] [i + 2] = -1; NOITN [i + 2] ++;
}
}


}
}
```

```
}
cout << '\ n';
// аналіз всіх вхідних формул

int NumOfInp = tot_elms.length ();
int sr = pow (2, TL);
int tl2 = TL;
int ** input = new int * [tl2];
for (int i = 0; i <tl2; i ++) {input [i] = new int [sr]; }

for (unsigned int i = 0; i <TL; i ++)
{
for (unsigned int j = 0; j <sr; j ++)
{
input [i] [j] = 0;
}
}
cout << '\ n';
for (unsigned int i = 0; i <TL; i ++)
{
for (unsigned int j = 0; j <sr; j ++)
{

input [i] [j] = (j >> i) & 0xF1; cout << input [i] [j];
} Cout << '\ n';
}
cout << '\ n';
// создание двійковій таблиці вхідних даних

double * ResultArray = new double [VS.size () + 2];
```

```
double * h = new double [VS.size () + 2];

double * W = new double [VS.size () + 2];

for (unsigned int i = 0; i <VS.size () + 2; i ++)
{
ResultArray [i] = 0;
W [i] = 1;
h [i] = 0;
}
cudapeqi (array, input, ResultArray, TL, LS, sr, NOITN);
// cout << "--------------" << '\ n';

copy (VS.begin (), VS.end (), ostream_iterator <string> (cout, ","));
std :: cout << '\ n';
VS.clear ();

}

}

return 0;
}
```