COMPUTER SYSTEMS AND NETWORKS

Vol. 6, No. 2, 2024

DOI: https://doi.org/10.23939/csn2024.02.232

УДК 004.02/.8

ОПТИМІЗАЦІЯ ШИРИНИ ПОТОКОВОГО ГРАФА АЛГОРИТМУ У НЕЙРОННИХ МЕРЕЖАХ ДЛЯ ЗМЕНШЕННЯ ВИКОРИСТАННЯ ПРОЦЕСОРНИХ ЕЛЕМЕНТІВ НА ОДНОПЛАТНИХ КОМП'ЮТЕРАХ

Є. І. Фастюк, Н. В. Гузинець

Національний університет "Львівська політехніка",
кафедра електронних обчислювальних машин
E-mail: yevhen.i.fastiuk@lpnu.ua, nataliia.v.huzynets@lpnu.ua

У статті представлено метод оптимізації потокового графа алгоритму глибинної нейронної мережі для зменшення кількості процесорних елементів (ПЕ), необхідних для виконання алгоритму на одноплатних комп'ютерах. Запропонований підхід ґрунтується на використанні структурної матриці для оптимізації архітектури нейронної мережі без втрати продуктивності. Дослідження показало, що завдяки зменшенню ширини графа вдалося зменшити кількість процесорних елементів від 3 до 2, зберігаючи при цьому продуктивність мережі на рівні 75 % ефективності. Цей підхід є важливим, оскільки дозволяє розширити можливості застосування нейронних мереж у вбудованих системах і IoT, підвищити ефективність використання обчислювальних ресурсів на пристроях з обмеженими обчислювальними можливостями, забезпечуючи ефективне використання обчислювальних ресурсів.

Ключові слова: нейронні мережі, потоковий граф алгоритму, оптимізація алгоритму, інтернет речей.

Вступ

У сучасному світі, де обчислювальна потужність є ключовим ресурсом, оптимізація алгоритмів набуває важливого значення для забезпечення ефективної роботи програмних систем. Одним з основних напрямів такої оптимізації є зменшення витрат на ресурси, зокрема, кількість процесорних елементів, які потрібні для виконання алгоритмів. Це питання стає особливо актуальним в контексті використання нейронних мереж, які знаходять широке застосування у розпізнаванні образів, автономних системах і багатьох інших сферах. Однак їх ефективність обмежується високими вимогами до обчислювальних ресурсів.

У цій роботі ставимо за мету дослідити можливість зменшення ширини потокового графа алгоритму глибинної нейронної мережі, щоб скоротити кількість процесорних елементів, необхідних для її виконання. Сучасні підходи до оптимізації нейронних мереж відкривають нові можливості для розробки більш ефективних алгоритмів, що можуть бути застосовані на пристроях з обмеженими ресурсами, таких як мікрокомп'ютери, використовувані в IoT-системах.

Огляд літературних джерел

У процесі дослідження оптимізації нейронних мереж з метою зменшення кількості процесорних елементів (ПЕ), використаних для виконання алгоритмів, важливо звернути увагу на сучасні підходи та інновації в галузі нейромережевих технологій та обчислювальних ресурсів.

Оптимізація нейронних мереж є ключовою проблемою для забезпечення їх ефективної роботи на пристроях з обмеженими апаратними ресурсами, таких як одноплатні комп'ютери.

Zhuang Liu et al., (2017) в своїй роботі пропонують новий підхід до оптимізації нейронних мереж шляхом звуження архітектури через видалення незначущих каналів. Цей підхід базується на використанні L1-регуляризації для прорідження мережі на рівні каналів, що дозволяє значно зменшити кількість обчислень без втрати точності моделі. Такий метод є корисним для оптимізації ресурсів при розгортанні нейронних мереж на обмежених апаратних платформах [1]. Vivienne Sze et al. (2017) описують кілька стратегій оптимізації, включаючи зменшення розрядності, прорідження ваг, розподіл великих фільтрів на менші, а також використання спеціалізованих апаратних платформ, таких як FPGA та ASIC. Це забезпечує прискорення обчислень та зменшення енергоспоживання, що особливо важливо для мобільних і вбудованих систем [2]. Hengyuan Hu et al. (2016) пропонують прорідження нейронних мереж на основі аналізу активації нейронів. Цей метод видаляє неважливі нейрони, вихід яких часто є нульовим, що дозволяє зменшити кількість параметрів і обчислювальну складність моделі. Такий підхід підходить для застосувань, де потрібна висока ефективність при мінімальних ресурсах [3]. Song Han et al. (2015) обговорюють оптимізацію нейронних мереж за рахунок зменшення кількості зв'язків між нейронами, що дозволяє значно зменшити кількість параметрів і обчислень [4].

Мельник А. та ін. (2010–2012) зосереджуються на представленні алгоритмів нейронних мереж через потокові графи та їх поданні у вигляді структурної матриці. Це допомагає оптимізувати процеси виконання алгоритмів та розподілу обчислювальних ресурсів, що є корисним для ефективного використання обмежених ПЕ [5, 6]. Їх робота може бути ефективно використана для оптимізації алгоритму нейронної мережі. Christian Szegedy et al. (2015) представили архітектуру глибинної нейронної мережі під назвою Inception, яка спрямована на оптимізацію використання обчислювальних ресурсів у згорткових нейронних мережах (CNN). Основні методи оптимізації, описані в статті, охоплюють: Модуль Inception, зменшення розмірності, розпаралелювання, глибинне навчання з малою кількістю параметрів [7]. Maher G. M. Abdolrasol et al. (2021) розглядають різні методи оптимізації нейронних мереж, що використовуються для підвищення їх ефективності в різних прикладних задачах. Основними методами, які розглядаються в статті, є такі: генетичні алгоритми, рій частинок і оптимізація бджолиних колоній [8].

Отже, огляд літературних джерел демонструє, що існує широкий спектр методів оптимізації нейронних мереж, які спрямовані на зменшення кількості обчислень, ресурсів пам'яті та енергоспоживання без втрати точності моделей. Наявність таких досліджень свідчить про зацікавленість наукової спільноти у використанні алгоритмів на обчислювальних машинах з обмеженою кількість апаратних ресурсів, таких як одноплатні мікрокомп'ютери, що використовуються в системах IoT та вбудованих системах. В останні роки розвиток одноплатних мікрокомп'ютерів, таких як Raspberry Pi, став ключовим фактором у популяризації вбудованих систем та розвитку IoT (інтернет речей). Однак навіть з усім їхнім потенціалом ці пристрої мають обмежені ресурси, зокрема, обмежену кількість процесорних елементів. У таких умовах оптимізація алгоритмів набуває надзвичайної важливості для забезпечення ефективної роботи програмних систем.

Постановка задачі

В умовах обмежених обчислювальних ресурсів, таких як одноплатні комп'ютери, оптимізація алгоритмів нейронних мереж є важливим завданням для забезпечення ефективної роботи систем. Враховуючи, що глибинні нейронні мережі використовують велику кількість процесорних елементів для виконання своїх завдань, особливо в областях розпізнавання образів та аналізу даних, необхідно знайти шляхи скорочення цих ресурсів без втрати точності та продуктивності.

Мета завдання полягає у зменшенні ширини потокового графа алгоритму глибинної нейронної мережі прямого зв'язку з метою зменшення кількості процесорних елементів, необхідних для роботи нейронної мережі на одноплатних комп'ютерах. Це досягається шляхом застосування

структурної матриці та інших методів оптимізації, які дозволяють зменшити обчислювальні витрати, зберігаючи при цьому ефективність алгоритму. Рішення цієї задачі є важливим для покращення продуктивності алгоритмів на пристроях з обмеженими ресурсами, таких як системи IoT і вбудовані системи.

Огляд нейронної мережі

Для дослідження було обрано Deep Feed Forward модель нейронної мережі. Deep Feed Forward (DFF) нейронні мережі є одними з найпоширеніших типів нейронних мереж. Вона також відома як мережа прямого поширення (feed-forward network), оскільки дані проходять через неї тільки в одному напрямку, від входу до виходу. Їх використовують для вирішення різноманітних завдань, таких як розпізнавання образів, класифікація даних, передбачення та генерація тексту, машинний переклад та багато іншого (рис. 1). Основна ідея застосування DFF мереж полягає в тому, що вони здатні розпізнавати складні залежності між вхідними даними та цільовими вихідними значеннями.

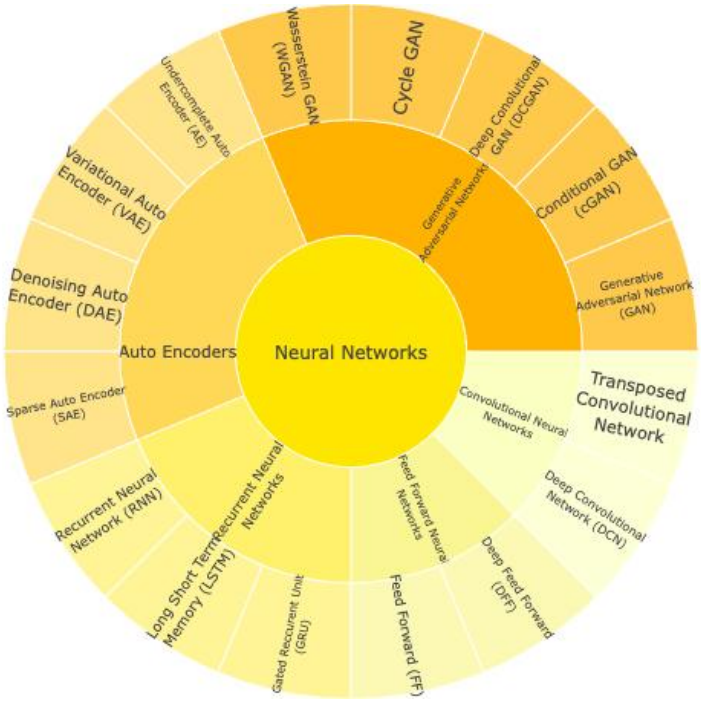

Рис. 1. Приклади нейронних мереж

Основною характеристикою DFF мережі є те, що вона складається з багатьох шарів, кожен з яких складається з нейронів. Нейрони кожного шару підключені до нейронів попереднього шару, інформація передається з вхідного шару до вихідного шару, пройшовши через проміжні шари, які називають прихованими. У кожному прихованому шарі зазвичай застосовується певна функція активації, яка дозволяє нейронам виявляти певні особливості вхідних даних. Зазвичай використовують такі функції, як ReLU, Sigmoid, Tanh [9].

Побудова алгоритму нейронної мережі

Deep feed-forward (DFF) мережу можна представити потоковим графом, де вузлами будуть шари нейронів, а ребрами – зв'язки між ними. На початку графу буде вузол вхідного шару, де кожен вхідний приклад буде входити до мережі. Потім вихід з кожного нейрона вхідного шару буде передаватися через ребра до першого прихованого шару, де виконуватимуться різні операції, зазвичай з використанням функції активації. Після обробки даних в кожному прихованому шарі

вихід буде передано до наступного прихованого шару, де процес повториться. Нарешті вихідний шар буде містити вихід мережі, де кожен нейрон представляє один із можливих класів (у задачах класифікації) або числове значення (у задачах регресії).

Потоковий граф DFF мережі може бути відображений візуально з допомогою різних інструментів, таких як TensorBoard, який дозволяє відслідковувати навчання мережі та зображувати її архітектуру у вигляді графу. Граф глибинної мережі прямого зв'язку зображено на рис. 2.

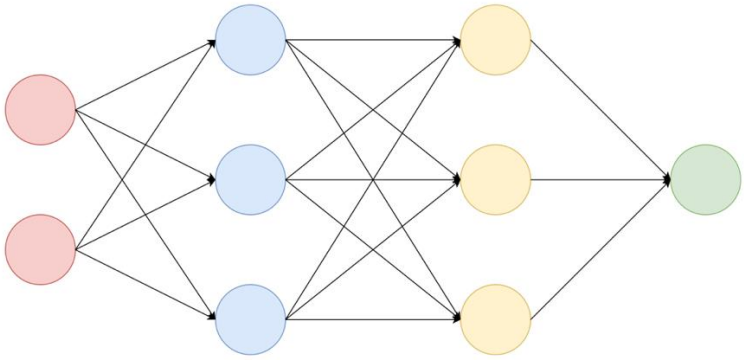

Рис. 2. Граф глибинної нейронної мережі прямого зв'язку

Як видно з наведеного рисунку, глибинна мережа прямого зв'язку має два приховані шари (синій (2) та жовтий (3), що й робить її "глибинною".

Побудова потокового графу (ПГА) заданої нейронної мережі

На рис. 2. зображено граф нейронної мережі. Проте ми не можемо його використати для побудови потокового графа алгоритму, оскільки всі його функціональні оператори (ФО) різні. Нам потрібно розбити ФО, зображені на рис. 2, на менші ФО, які матимуть загальний вигляд. Також це допоможе представити інформацію про ПГА за допомогою структурної матриці. Для зображення та дослідження графа алгоритму G необхідно точно описати множини вершин K та дуг R графа алгоритму [5] (рис. 3).

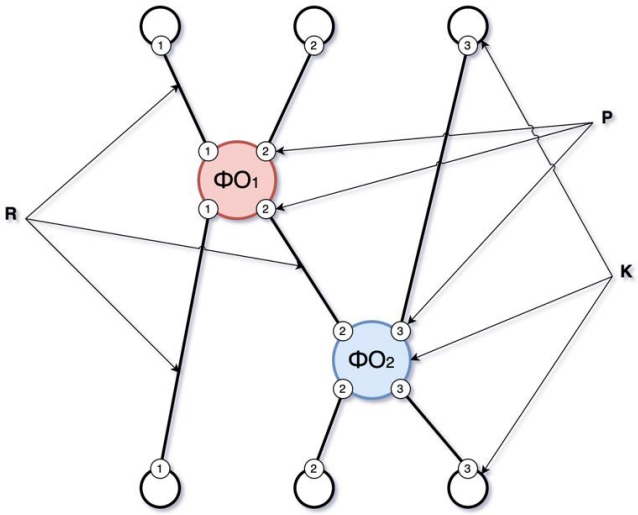

Рис. 3. Граф алгоритму G (K, P, R)

Усі вершини отримують ідентифікатори (порядкові номери) в порядку створення. Ідентифікатори утворюють множину ідентифікаторів Ik, де {Ik} набуває значення із множини натуральних чисел і k = 1, Nf , Nf – кількість вершин ГА [5].

Побудова ПГА

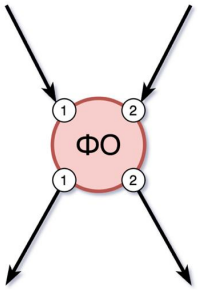

Рис. 4. Функціональний оператор

Розбиваємо граф нейронної мережі на менші функціональні оператори (ФО). При побудові потокового графу алгоритму було використано базовий ФО з двома входами та двома виходами (рис. 4). Після розбиття будуємо потоковий граф алгоритму (ПГА) глибинної нейронної мережі прямого зв'язку та представляємо потоковий граф алгоритму у формі структурної матриці [10].

Після розбиття і побудови потоковий граф алгоритму глибинної нейронної мережі прямого зв'язку матиме такий вигляд (рис. 5).

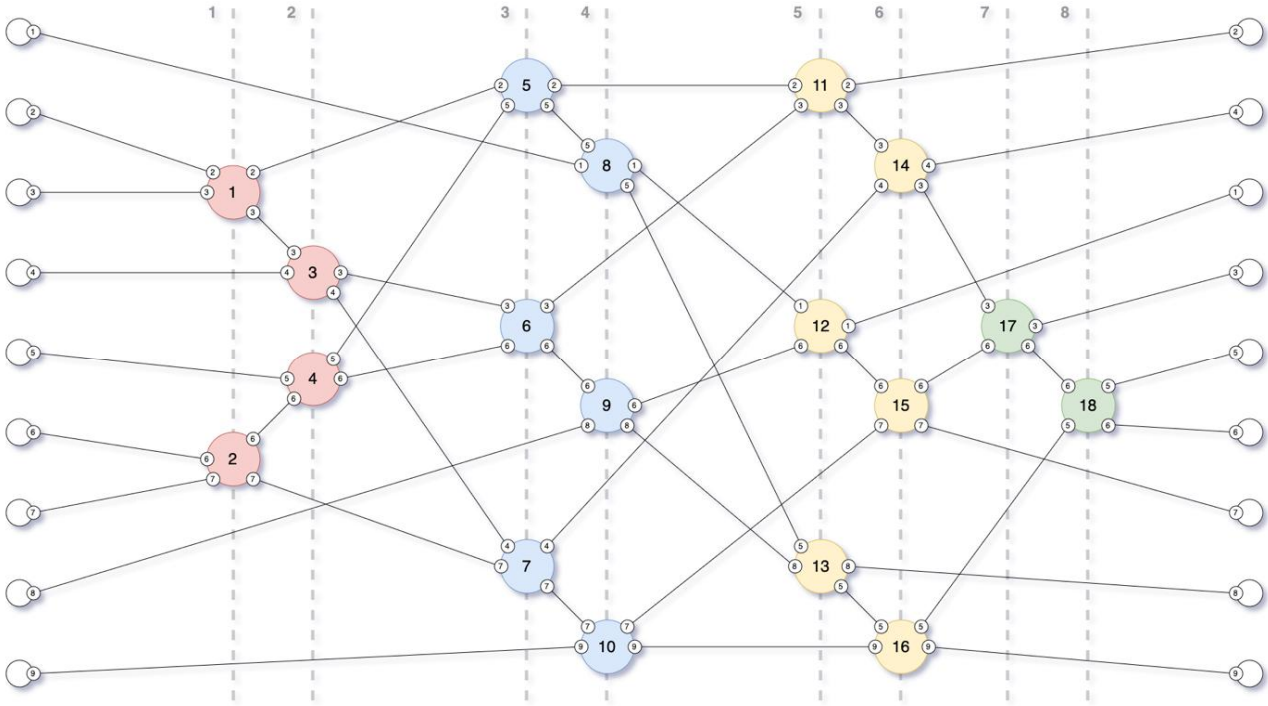

Рис. 5. Потоковий граф алгоритму нейронної мережі

Представлення ПГА у формі структурної матриці (СМ)

Структурна матриця F розмірністю $l \times n$, елементами якої є номери ФО, присвоєні кожній дузі, по якій на ФО надходять операнди, має такий вигляд [5]:

$$F = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & f_{ij} & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ f_{l1} & f_{l2} & \cdots & f_{\ln} \end{pmatrix},$$

де $\forall$ fij $\in N$, $i = 1, l$, $j = 1, n$, l – кількість ярусів (рядків), n – загальна кількість дуг найширшого ярусу, по яких надходять операнди (стовпці).

Заповнюється структурна матриця F за таким правилом: на перетині і-го рядку і j-го стовпця ставиться номер відповідного ФО і-го ярусу, який виконує операцію над операндом, який надходить j-ю дугою ПГА fij = k, де k – належить множині натуральних чисел $k \in N$. Нумерація

починається від 1 і призначається всім наступним ФО підряд в порядку зростання. Решта елементів такої матриці заповнюються нулями.

Результат виконання алгоритму для ГА, заданого графічно (рис. 6, *а*), зображено на екранних формах програми (рис. 6): ПГА (рис. 6, *б*); структурна матриця (рис. 6, *в*) [5].

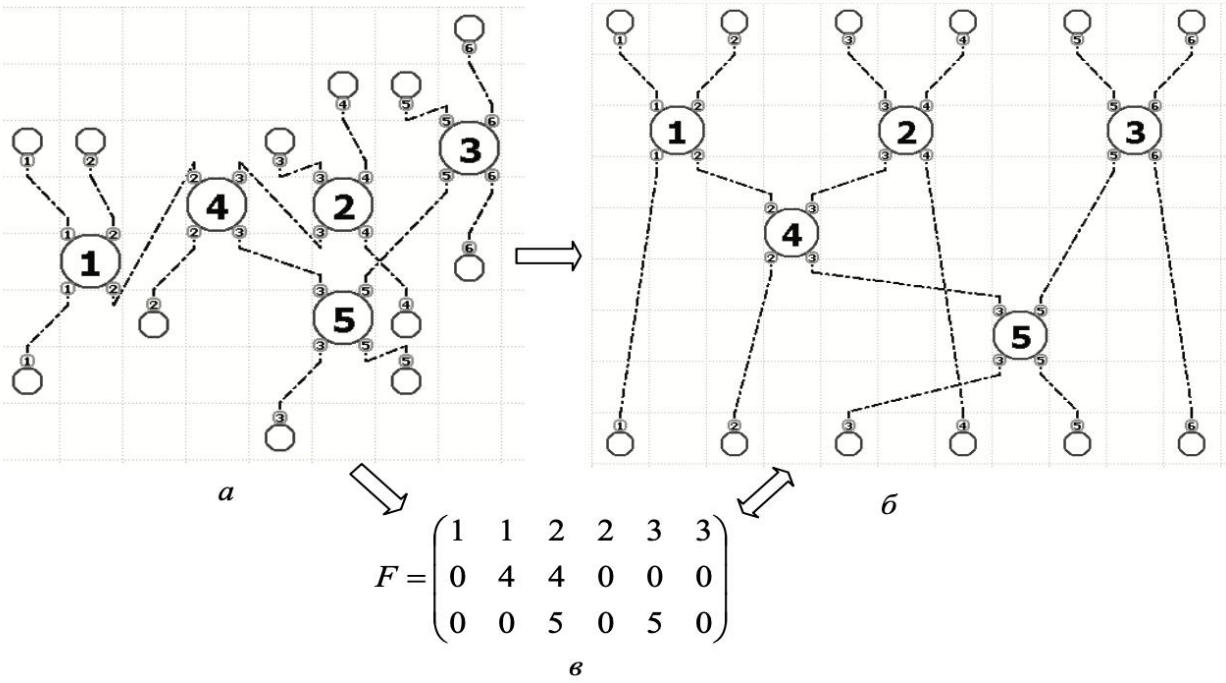

Рис. 6. Побудова ПГА із ГА, заданого графічно, та заповнення структурної матриці: а – ГА; б – ПГА; в – структурна матриця

Таблиця 1

Структурна матриця ПГА нейронної мережі

Дуги										
Яруси	**№**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	1	0	1	1	0	0	2	2	0	0
	2	0	0	3	3	4	4	0	0	0
	3	0	5	6	7	5	6	7	0	0
	4	8	0	0	0	8	9	10	9	10
	5	12	11	11	0	13	12	0	13	0
	6	0	0	14	14	16	15	15	0	16
	7	0	0	17	0	0	17	0	0	0
	8	0	0	0	0	18	18	0	0	0

У табл. 1 наведена структурна матриця потокового графа алгоритму глибинної нейронної мережі прямого зв’язку.

Також для можливості реалізації алгоритму за заданою структурною матрицею необхідно надати опис вершин графа (табл. 2).

Визначення характеристик та показників сформованого ПГА

1) кількість аргументів алгоритму: 9. Кількість стовпців у СМ дорівнює 9.

2) кількість використовуваних у і-й момент часу ПЕ wi для реалізації паралельного алгоритму: 3.

3) кількість використовуваних у і-й момент часу процесорних елементів дорівнює 3.

4) час виконання послідовного алгоритму T1: 18.
5) кількість значущих елементів СМ дорівнює 18.
6) час виконання паралельного алгоритму Tp: 8.
7) кількість ярусів ПГА та кількість рядків СМ дорівнює 8.
8) кількість необхідних ПЕ p для максимального розпаралелювання алгоритму: 3.
9) така кількість використовується на рівнях – 3, 4, 5, 6.
10) прискорення Sp = 18 / 8 = 2,25.
11) ефективність Ep = 2.25 / 3 = 0,75 = 75 %.

Таблиця 2

Опис вершин

Вершина	
Ідентифікатор вершини	**Тип вершини**
1	Вхід НМ * вага
2	Вхід НМ * вага
3	регістр
4	регістр
5	Вхід НМ * вага
6	Вхід НМ * вага
7	Вхід НМ * вага
8	регістр
9	регістр
10	регістр
11	Вхід НМ * вага
12	Вхід НМ * вага
13	Вхід НМ * вага
14	Вхід НМ * вага
15	Вхід НМ * вага
16	Вхід НМ * вага
17	Вхід НМ * вага
18	Вхід НМ * вага

Аналіз завантаженості ПЕ, зменшення ширини ПГА та перебудова ПГА з врахуванням можливої зміни кількості використовуваних ПЕ

Матриця завантаженості ППГА виглядає так (табл. 3).

Таблиця 3

Матриця завантаженості ПЕ

1	1	0	0	2	2
0	3	3	4	4	0
5	6	7	5	6	7
8	8	9	10	9	10
12	11	11	13	12	13
14	14	16	15	15	16
0	17	0	0	17	0
0	0	18	18	0	0

Загальна кількість вузлів, необхідна для виконання цього алгоритму роботи НМ, дорівнює кількості вузлів на найбільшому шарі. У цьому випадку це 6.

Коефіцієнт завантаження визначається відношенням загальної кількості ПЕ до кількості ПЕ, необхідних для виконання НМ.

- Кількість ПЕ – 3
- Загальна кількість – 18
- Коефіцієнт завантаженості процесорних елементів – 18 / 3 = 6

Завантаженість ПГА на 3–5 рівнях дорівнює 3. Отже, такий алгоритм погано виконуватиметься на ОМ з кількістю процесорних елементів 2.

Щоб збільшити ефективність цього алгоритму на ОМ з такою кількістю процесорних елементів, потрібно зменшити ширину ПГА.

Можлива зміна кількості ПЕ – 2. Для отримання такого результату, ПГА повинна виглядати так (рис. 7).

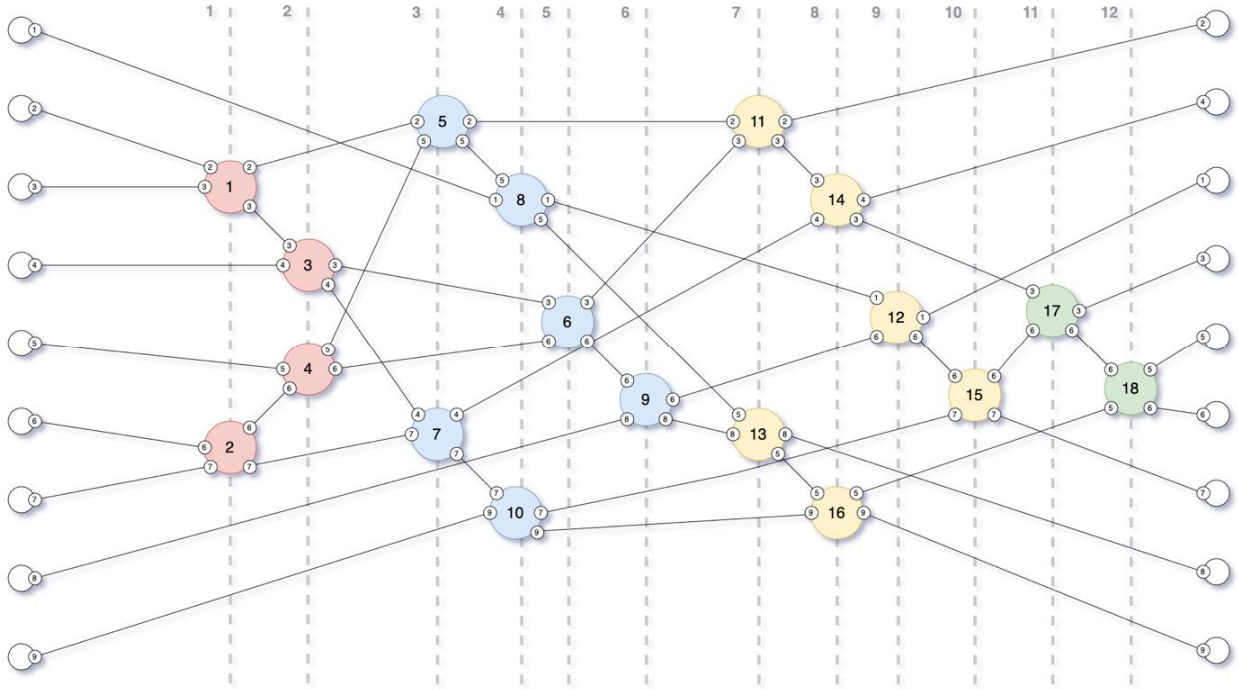

Рис. 7. Потоковий граф алгоритму зі зменшеною шириною графа

1) кількість аргументів алгоритму – 9.
2) кількість стовпців у СМ дорівнює 9.
3) кількість використовуваних у і-й момент часу ПЕ wi для реалізації паралельного алгоритму – 2.
4) кількість використовуваних у і-й момент часу процесорних елементів дорівнює 2.
5) час виконання послідовного алгоритму Tl: 18.
6) кількість значущих елементів СМ дорівнює 18.
7) час виконання паралельного алгоритму Tp: 12.
8) кількість ярусів ПГА та кількість рядків СМ дорівнює 12.
9) кількість необхідних ПЕ p для максимального розпаралелювання алгоритму – 2.
10) така кількість використовується на рівнях – 3–10
11) прискорення Sp = 18 / 12 = 1,5.
12) ефективність Ep = 1,5 / 2 = 0,75 = 75 %.

Матриця завантаженості буде виглядати так (табл. 4).

Таблиця 4

Матриця завантаженості ПЕ

1	1	2	2
3	3	4	4
5	7	5	7
8	8	10	10
0	6	6	0
0	9	9	0
11	11	13	13
14	14	16	16
0	12	12	0
0	15	15	0
17	0	0	17
0	18	18	0

Коефіцієнт завантаження визначають відношенням загальної кількості ПЕ до кількості ПЕ, необхідних для виконання НМ.

- Кількість ПЕ – 2
- Загальна кількість – 18
- Коефіцієнт завантаженості процесорних елементів – 18 / 2 = 9

Результати дослідження

У ході дослідження було розроблено та успішно впроваджено методику зменшення ширини потокового графа алгоритму глибинної нейронної мережі прямого зв'язку (DFF). Основні результати дослідження містять:

1) зменшення кількості процесорних елементів (ПЕ): завдяки зменшенню ширини потокового графа вдалося скоротити кількість ПЕ, необхідних для виконання алгоритму нейронної мережі з 3 до 2. Це дозволило підвищити ефективність використання обчислювальних ресурсів, особливо на пристроях з обмеженими апаратними можливостями, таких як одноплатні мікрокомп'ютери;

2) збереження продуктивності алгоритму: незважаючи на зменшення кількості ПЕ, ефективність роботи алгоритму залишилася на високому рівні, зберігаючи продуктивність і точність нейронної мережі. Прискорення виконання паралельного алгоритму було досягнуто зі збереженням коефіцієнта ефективності на рівні 75 %;

3) покращення розпаралелювання: зменшення ширини потокового графа сприяло кращому розподілу обчислювальних ресурсів між різними ярусами графа, що дозволило оптимізувати розпаралелювання алгоритму.

Загалом результати дослідження демонструють успішність запропонованого підходу для оптимізації нейронних мереж шляхом зменшення ширини потокового графа та зниження вимог до обчислювальних ресурсів без втрати точності. Це відкриває нові можливості для застосування нейронних мереж у системах з обмеженими апаратними ресурсами.

Висновки

У результаті проведеного дослідження було розроблено ефективну методику зменшення ширини потокового графа алгоритму глибинної нейронної мережі прямого зв'язку. Основною метою було зменшення кількості процесорних елементів (ПЕ), необхідних для виконання алгоритму, без значних втрат у точності та продуктивності моделі. Запропонований підхід показав високий потенціал для оптимізації обчислень на пристроях з обмеженими ресурсами, таких як одноплатні комп'ютери, що використовуються в системах Інтернету речей (IoT) і вбудованих системах [11]. Оптимізація потокового графа дозволила зменшити кількість процесорних елементів, необхідних для роботи нейронної мережі, що призвело до більш ефективного використання обмежених обчислювальних ресурсів. Незважаючи на скорочення кількості ПЕ, вдалося зберегти високу точність

і продуктивність нейронної мережі. В майбутньому цю методику можна удосконалити шляхом поєднання з іншими методами оптимізації, такими як проріджування ваг або використання менш складних функцій активації. Запропонований підхід є важливим кроком до створення більш ефективних та енергоекономних систем на основі глибинних нейронних мереж.

Список літератури

1. *Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning Efficient Convolutional Networks through Network Slimming. CoRR abs/1708.06519, (2017). DOI: 10.48550/arXiv.1708.06519*
2. *Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. CoRR abs/1703.09039, (2017). DOI: 10.48550/arXiv.1703.09039*
3. *Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. 2016. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. CoRR abs/1607.03250, (2016). DOI: 10.48550/arXiv.1607.03250*
4. *Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. CoRR abs/1506.02626, (2015). DOI: 10.48550/arXiv.1506.02626*
5. *Мельник А. О., Яковлєва І. Д. і Ющенко В. Ю. 2010. ПОБУДОВА ТА МАТРИЧНЕ ПОДАННЯ ПОТОКОВОГО ГРАФА АЛГОРИТМУ. Вісник Вінницького політехнічного інституту. 3 (Листоп. 2010), 93–99. URL: https://visnyk.vntu.edu.ua/index.php/visnyk/article/view/757*
6. *Мельник А. О., Мицко Ю. Є. 2012. ВИКОНАННЯ ПОДАНИХ ПОТОКОВИМ ГРАФОМ АЛГОРИТМІВ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ GPGPU. Вісник Національного університету "Львівська політехніка", (745), 124–130. URL: https://ena.lpnu.ua/handle/ntb/20127*
7. *Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1–9. DOI: 10.1109/CVPR.2015.7298594*
8. *Maher G. M., Abdolrasol S. M. Suhail Hussain, Taha Selim Ustun, Mahidur R. Sarker, Mahammad A. Hannan, Ramizi Mohamed, Jamal Abd Ali, Saad Mekhilef, and Abdalrhman Milad. 2021. Artificial Neural Networks Based Optimization Techniques: A Review. Electronics 10, 21 (2021). DOI: 10.3390/electronics10212689*
9. *Feed-forward propagation from scratch in Python. Online resource. URI: https://subscription. packtpub.com/book/data/9781789346640/1/ch01lvl1sec05/feed-forward-propagation-from-scratch-in-python*
10. *Фастюк Є., Гузинець Н. 2024. ОПТИМІЗАЦІЯ АЛГОРИТМУ РОБОТИ НЕЙРОННОЇ МЕРЕЖІ ЗА РАХУНОК ЗМЕНШЕННЯ ШИРИНИ ПОТОКОВОГО ГРАФА АЛГОРИТМУ. Матеріали конференцій МЦНД, (31.05.2024; Черкаси, Україна), 214–216. DOI: 10.62731/mcnd-31.05.2024.006*
11. *Fastiuk Y., Bachynskyy R., Huzynets N. 2021. Methods of Vehicle Recognition and Detecting Traffic Rules Violations on Motion Picture Based on OpenCV Framework. Advances in Cyber-Physical Systems, 6(2), 105–111. DOI: 10.23939/acps2021.02.105*

OPTIMIZATION OF THE ALGORITHM FLOW GRAPH WIDTH IN NEURAL NETWORKS TO REDUCE THE USE OF PROCESSOR ELEMENTS ON SINGLE-BOARD COMPUTERS

Y. Fastiuk, N. Huzynets

Lviv Polytechnic National University,
Electronic Computational Machines Department
E-mail: yevhen.i.fastiuk@lpnu.ua, nataliia.v.huzynets@lpnu.ua

The article presents a method for optimizing the algorithm flow graph of a deep neural network to reduce the number of processor elements (PE) required for executing the algorithm on single-board computers. The proposed approach is based on the use of a structural matrix to optimize the neural network architecture without loss of performance. The research demonstrated that by reducing the width of the graph, the number of processor elements was reduced from 3 to 2, while maintaining network performance at 75 % efficiency. This approach is significant as it expands the potential applications of neural networks in embedded systems and IoT, enhancing the efficiency of computational resource utilization on devices with limited computational capabilities, ensuring effective use of resources.

Keywords: Neural network, algorithm flow graph, algorithm optimization, IoT.