НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

На правах рукопису
УДК 004.852

До захисту допущено
В.о. Завідувач кафедри ШІ
_____ О.І. Чумаченко
«___»_____2022 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 122 «Комп'ютерні науки»

на тему: «Алгоритми сортування з використанням нейронних мереж»

Виконала:
студентка 2 курсу, групи КІ-11мп
Павлюк Віра _____

Керівник:
доцент кафедри ММСА,
к.ф.-м.н., доц. Подколзін Г.Б. _____

Рецензент:
Доцент кафедри МАтаТЙ
к.ф.-м.н., доц. Орловський І. В. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Рівень вищої освіти – другий (магістерський)

Спеціальність – 122 «Комп'ютерні науки »

ЗАТВЕРДЖУЮ
В.о. Завідувач кафедри ШІ
_____ О.І. Чумаченко
«___»_____2022 р.

ЗАВДАННЯ
на магістерську дисертацію студентці
Павлюк Вірі

1. Тема дисертації: «Алгоритми сортування з використанням нейронних мереж», науковий керівник роботи Подколзін Гліб Борисович, доцент кафедри ММСА, к.ф.-м.н., доц. затверджено наказом по університету від «3» листопада 2022 р. № 4046-с.
2. Термін подання студентом дисертації 15.12.2022
3. Об'єкт дослідження: Задача сортування числових даних.
4. Предмет дослідження: Моделі нейронних мереж з увагою.
5. Перелік завдань, які потрібно зробити:
    1) здійснити огляд технічної літератури за темою роботи;
    2) дослідити актуальність обраної теми;
    3) ознайомитись із існуючими методами та моделями сортування даних;
    4) здійснити порівняльний аналіз наявних методів, виявити їх переваги та недоліки;

5) розробити та реалізувати систему, що використовує апарат нейрониих мереж, та вирішує задачу сортування числових даних нефіксованої довжини;

6) провести експеримент, що засвідчує працеспроможність запропонованої моделі, виконати аналіз результатів;

7) провести аналіз ринкових можливотсей запуску стартап проекту;

8) розробити концептуальні висновки;

9) підготувати ілюстративний матеріал;

10) оформити пояснювальну записку.

6. Перелік ілюстративного матеріалу.

7. Дата видачі завдання: 1 вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за темою роботи.	01.09.2022 – 14.09.2022	Виконано
2.	Підготовка першого розділу.	14.09.2022 – 16.09.2022	Виконано
3.	Підготовка другого розділу.	16.09.2022 – 23.09.2022	Виконано
4.	Розробка програмного продукту.	01.10.2022 – 20.11.2022	Виконано
5.	Підготовка третього розділу	20.11.2022 – 23.11.2022	Виконано
6.	Підготовка частини стартап-проєкту	23.11.2022 – 25.11.2022	Виконано
9.	Концептуальні висновки. Перспективи розвитку отриманих рішень	25.11.2022 – 28.11.2022	Виконано
10.	Оформлення пояснювальної записки	28.11.2022 – 01.12.2022	Виконано

Студент Віра ПАВЛЮК

Керівник Гліб ПОДКОЛЗІН

РЕФЕРАТ

Дипломна робота містить 92 с., 19 рис., 8 табл., 1 додаток, 30 джерел.

АЛГОРИТМИ СОРТУВАННЯ, НЕЙРОННІ МЕРЕЖІ, ГЛИБОКЕ НАВЧАННЯ, SEQUENCE-TO-SEQUENCE, POINTER NETWORKS, ФУНКЦІЯ УВАГИ, СОРТУВАННЯ ЧИСЕЛ.

Об’єкт дослідження: одновимірні масиви різної довжини, що містять цілі числа.

Мета дослідження: аналіз доцільності та спроможності використання нейронних мереж у задачі сортування даних.

Використані моделі: кодери-декодери послідовність-до-послідовності та мережа вказівників, класичні алгоритми сортування.

Отриманні результати: виявлено, що мережа вказівників краще впоралась з задачею сортування, ніж послідовність-до-послідовності. Запропонована модифікація механізму уваги для мережі вказівників, який покращує результати роботи моделі.

В рамках подальшого дослідження пропонується підвищувати точність моделі, проводити додаткові експерименти з метою мінімізації часу сортування та безпосереднього навчання моделі.

ABSTRACT

The diploma thesis contains 92 p., 19 fig., 8 tabl, 1 appendicy, 30 sources.

SORTING ALGORITHMS, NEURAL NETWORKS, DEEP LEARNING, SEQUENCE-TO-SEQUENCE, POINTER NETWORKS, ATTENTION FUNCTION, SORTING NUMBERS.

Object of research: one-dimensional arrays of different lengths consisting of an integer.

The purpose of the study: analysis of the feasibility and effectiveness of using neural networks in data sorting tasks.

Used models: sequence-to-sequence encoder-decoders and pointer network, classical sorting algorithms.

Findings: The pointer network was found to perform better at the sorting task than sequence-to-sequence. A modification of the attention mechanism for the pointer network is proposed, which improves the performance of the model.

As part of further research, it is suggested to increase the accuracy of the model, to conduct additional experiments in order to minimize the time of sorting and pre-training of the models.

ЗМІСТ

ВСТУП

Сортування є основною операцією багатьох обчислювальних завдань, включаючи розробку НВІС, цифрову обробку сигналів, мережевий зв’язок, керування базою даних і обробку даних, для яких, за оцінками, операції сортування займають понад 25% загального часу обробки. Значення сортування відображено в безлічі методів сортування, які були запропоновані протягом попередніх десятиліть. Головною метою цих методів, незалежно від того, чи вони є послідовними чи паралельними за конструкцією та роботою, є мінімізація часу та вимог до зберігання під час операції сортування. Інструктивні огляди методів послідовного та паралельного сортування наведено в [1] та [2] відповідно.

Основними цілями алгоритму паралельного сортування є мінімізація часу сортування і процесори, упорядковуючи дані елементи в бажаному порядку. Нейронна мережа - це відповідна архітектура в деяких паралельних алгоритмах, оскільки вона може обробляти дані одночасно [4].

У роботах 4, 5, 6 описано класичні моделі адресної пам'яті і асоціативних процесорів як базових компонентів інтелектуальних систем. При цьому перераховані асоціативні операції, що входять до набору операцій, реалізованих в асоціативних процесорах: пошук відповідності, пошук мінімальних/максимальних значень, пошук на основі булевої функції, впорядкований вибір (сортування) 4,5,6. Крім того, відомо, що класичні мережі типу Хеммінга, Хопфілда, мережі MAXNET 7,8,9 можуть виконувати найпростіші асоціативно-логічні операції. Водночас є потреба в ефективних сортувальниках. Це пов'язано з необхідністю обробки великих масивів даних, а точніше реалізувати виведення даних із пам'яті з упорядкуванням елементів масиву за визначеним правилом у реальному часі. Загалом інтерес дає можливість реалізувати процедуру сортування масиву даних за допомогою нейромережевого підходу, що є метою даної роботи.

Завдання сортування елементів масивів даних є класичним, це одне з найважливіших завдань програмування. Проведення експерименту для сортування масиву даних дозволить визначити ефективність нейронних мереж у галузі аналізу та обробки масивів даних. Мета даної роботи побудувати модель для сортування на основі нейронної мережі та порівняти кінцеві результати з класичними алгоритмами, такими як швидке сортування, сортування злиттям, бульбашкове сортування, інтроспективне сортування, тімсорт та сортування купою.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Класичні алгоритми сортування

Алгоритм сортування — це алгоритм, що складається з серії інструкцій, які приймають масив як вхідні дані, виконують певні операції над масивом, який іноді називають списком, і виводять відсортований масив. Алгоритм сортування використовується для перевпорядкування заданого масиву або списку елементів відповідно до оператора порівняння елементів. Оператор порівняння використовується для визначення нового порядку елементів у відповідній структурі даних.

Алгоритмам сортування може знадобитися додатковий простір для порівняння та тимчасового зберігання кількох елементів даних. Ці алгоритми не потребують додаткового простору, а сортування відбувається на місці або, наприклад, у самому масиві. Це називається сортуванням на місці. Бульбашкове сортування є прикладом сортування на місці. Однак у деяких алгоритмах сортування програмі потрібен простір, який перевищує або дорівнює елементам, які сортуються. Сортування, яке займає рівний або більший простір, називається сортуванням не на місці. Сортування злиттям є прикладом сортування не на місці.

Алгоритм сортування на місці використовує постійний простір для отримання результату (змінює лише заданий масив). Він сортує список, лише змінюючи порядок елементів у списку. Наприклад, сортування за допомогою вставки та сортування за вибором є алгоритмами сортування на місці, оскільки вони не використовують додатковий простір для сортування списку, а типова реалізація сортування злиттям не є на місці, також реалізація сортування підрахунку не є наявною. алгоритм сортування місця. тому складність допоміжного простору алгоритмів сортування не на місці збільшується на O(N),

де N — кількість елементів, до яких має бути застосоване сортування, тоді як для алгоритмів на місці вона не збільшується.

Якщо алгоритм сортування після сортування вмісту не змінює послідовність подібного вмісту, у якому він з'являється, це називається стабільним сортуванням. Якщо алгоритм сортування після сортування вмісту змінює послідовність подібного вмісту, у якій він з'являється, це називається нестабільним сортуванням.

Алгоритм сортування називається адаптивним, якщо він використовує переваги вже «відсортованих» елементів у списку, який потрібно відсортувати. Тобто під час сортування, якщо у вихідному списку є вже відсортовані елементи, адаптивні алгоритми врахують це і намагатимуться не перевпорядковувати їх. Неадаптивний алгоритм – це алгоритм, який не враховує вже відсортовані елементи. Вони намагаються змусити кожен окремий елемент бути перевпорядкованим, щоб підтвердити його сортування [6].

Нижче наведено опис кількох найпопулярніших класичних алгоритмів сортування.

1.1.1 Алгоритм швидкого сортування

Алгортитм швидкого сортування (Quicksort) — один із найпопулярніших алгоритмів сортування, який використовує порівняння $n\ \log(n)$ для сортування масиву з n елементів у типовій ситуації. Алгоритм швидкого сортування базується на стратегії «розділяй і володарюй».

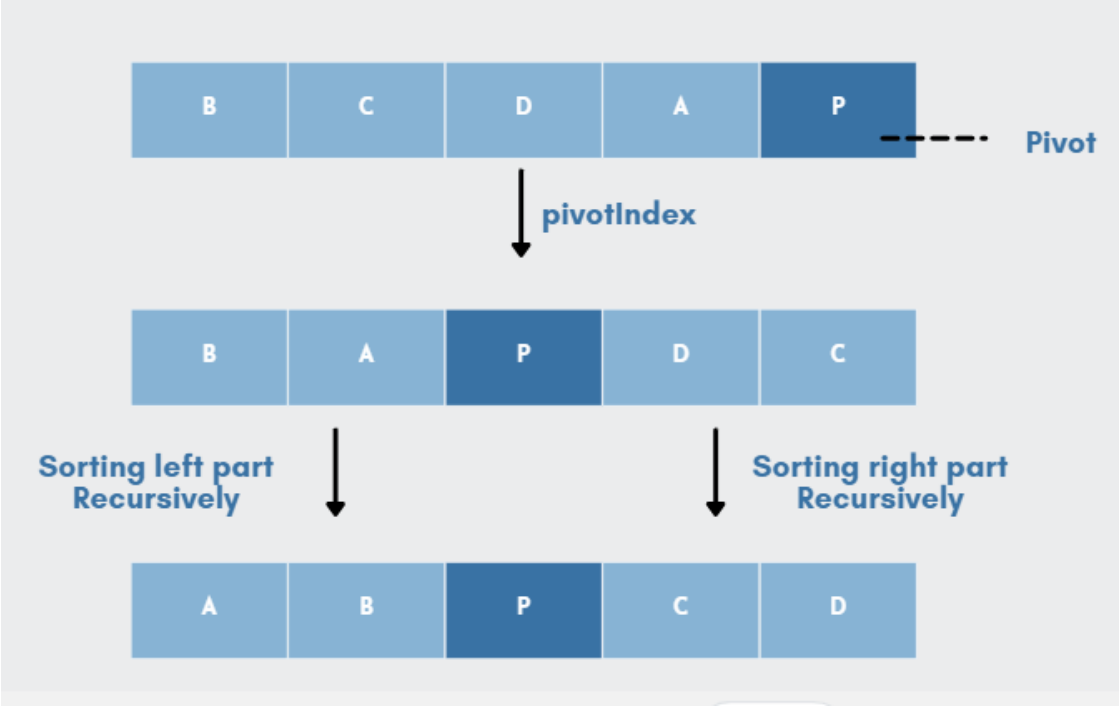


Рисунок 1.1. Огляд алгоритму QuickSort

Швидке сортування — це швидкий алгоритм сортування, який розбиває великий масив даних на менші підмасиви. Це означає, що кожна ітерація працює, розбиваючи вхідні дані на два компоненти, сортуючи їх і потім повторно комбінуючи. Для великих наборів даних цей метод є високоефективним, оскільки його середня та найкраща складність становить $O(n \cdot \log(n))$.

Він був створений Тоні Хоаром у 1961 році і залишається одним із найефективніших алгоритмів сортування загального призначення, доступних сьогодні. Він працює шляхом рекурсивного сортування підсписків по обидва боки від даної опорної точки та динамічного переміщення елементів усередині списку навколо цієї опорної точки.

Таким чином, метод швидкого сортування можна підсумувати в три етапи:

– Вибір: виберіть елемент.
– Розділ: розділіть набір проблем, перемістіть менші частини ліворуч від опори, а більші предмети – праворуч.
– Повторити та об’єднати: повторіть кроки та об’єднайте масиви, які раніше були відсортовані.

Ключові переваги використання алгоритму швидкого сортування:

– Діє швидко і ефективно.
– Він має найкращу часову складність порівняно з іншими алгоритмами сортування.
– Швидке сортування має складність простору $O(n \cdot \log(n))$, що робить його чудовим вибором для ситуацій, коли простір обмежений.

Незважаючи на те, що QuickSort є найшвидшим алгоритмом, він має кілька недоліків:

– Цей метод сортування вважається нестабільним, оскільки він не підтримує початковий порядок пар ключ-значення.
– Коли опорний елемент є найбільшим або найменшим, або коли всі компоненти мають однаковий розмір. Ці найгірші сценарії суттєво впливають на продуктивність швидкого сортування.
– Його важко реалізувати, оскільки це рекурсивний процес, особливо якщо рекурсія недоступна [7].

Складність по часу	Найкраща	$O\ (n\ \log(n))$
	Средня	$O\ (n\ \log(n))$
	Найгірша	$O(n^!)$
Складність по пам’яті	Загальна	$O(n)$

Таблиця 1.1. Характеристика алгоритму швидкого сортування

1.1.2 Алгоритм сортування злиттям

Алгоритм сортування злиттям — це алгоритм сортування, заснований на парадигмі «розділяй і володарюй». У цьому алгоритмі масив спочатку ділиться на дві рівні половини, а потім вони об’єднуються в сортований спосіб.

Він є рекурсивним алгоритмом, який безперервно ділить масив навпіл, доки його неможливо розділити далі. Це означає, що якщо масив стане порожнім або залишиться лише один елемент, поділ припиниться, тобто це базовий випадок для зупинки рекурсії. Якщо масив містить кілька елементів, він ділиться на половини та рекурсивно викликається сортування злиттям для кожної з половин. Нарешті, коли обидві половини відсортовано, застосовується операція злиття. Операція злиття — це процес об’єднання двох менших відсортованих масивів для створення більшого.

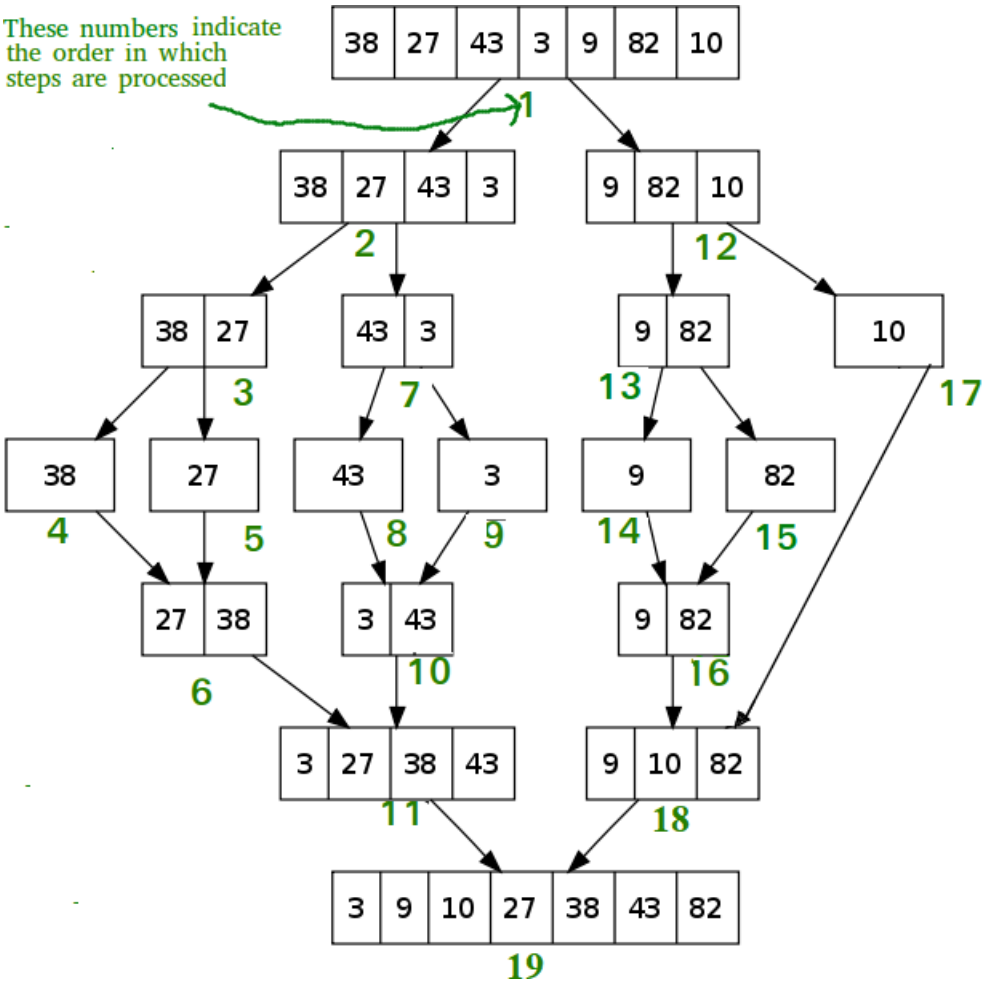


Рисунок 1.2. Рекурсивні кроки сортування злиттям

Сортування злиттям складається з кількох проходів по вхідних даних. Перший прохід об’єднує сегменти розміру 1, другий об’єднує сегменти розміру

2, а i прохід об'єднує сегменти розміру $2^{"\#\$}$. Таким чином, загальна кількість проходів становить $\log(2n)$. Як показало злиття, ми можемо об'єднати два відсортовані сегменти за лінійний час, що означає, що кожен прохід займає $O(n)$ часу. Оскільки є $\log(2n)$ проходів, загальний час обчислення становить $O(\mathrm{n} \log\ n)$.

Сортування злиттям корисно для сортування пов'язаних списків за час $O(N\ \log(N))$. У випадку зв'язаних списків ситуація інша в основному через різницю в розподілі пам'яті для масивів і зв'язаних списків. На відміну від масивів, вузли пов'язаного списку не можуть бути суміжними в пам'яті. На відміну від масиву, у зв'язаному списку ми можемо вставляти елементи в середину за $O(1)$ додаткового простору та $O(1)$ часу. Тому операцію злиття сортування злиттям можна реалізувати без додаткового місця для пов'язаних списків. У масивах ми можемо виконувати довільний доступ, оскільки елементи є суміжними в пам'яті. Скажімо, у нас є цілий (4-байтовий) масив A і нехай адреса A[0] буде x, тоді для доступу до A[i] ми можемо отримати прямий доступ до пам'яті за адресою $(x\ +\ 4i)$. На відміну від масивів, ми не можемо робити довільний доступ у зв'язаному списку. Швидке сортування вимагає багато такого доступу. У пов'язаному списку для доступу до і-го індексу нам потрібно переміщатися по кожному вузлу від голови до і-го вузла, оскільки у нас немає безперервного блоку пам'яті. Тому накладні витрати збільшуються нашвидке сортування. Сортування злиттям отримує послідовний доступ до даних, і потреба у довільному доступі є низькою [8].

Недоліки сортування злиттям:

- Повільніше порівняно з іншими алгоритмами сортування для менших завдань.
- Алгоритм сортування злиттям вимагає додаткового простору пам'яті $O(n)$ для тимчасового масиву.
- Він проходить весь процес, навіть якщо масив відсортовано.

Складність по часу	Найкраща	$O\ (n\ \log(n))$
	Средня	$O\ (n\ \log(n))$
	Найгірша	$O\ (n\ \log(n))$
Складність по пам’яті	Загальна	$O(n)$
	Додаткова	$O(1)$

Таблиця 1.2. Характеристика алгоритму сортування злиття

1.1.3 Алгоритм сортування бульбашкою

Бульбашкове сортування — це найпростіший алгоритм сортування, який працює шляхом повторної заміни суміжних елементів, якщо вони розташовані в неправильному порядку. Цей алгоритм не підходить для великих наборів даних, оскільки його середня та найгірша часова складність досить висока.

Бульбашкове сортування, яке іноді називають сортуванням із зануренням, — це простий алгоритм сортування, який багаторазово перебирає вхідний список елемент за елементом, порівнюючи поточний елемент із наступним за ним, міняючи їхні значення, якщо потрібно. Ці проходи через список повторюються до тих пір, поки під час проходу не буде виконано жодних замін, що означає, що список стане повністю відсортованим. Алгоритм, який є порівняльним сортуванням, названий на честь того, як більші елементи «випливають» угору списку [20].

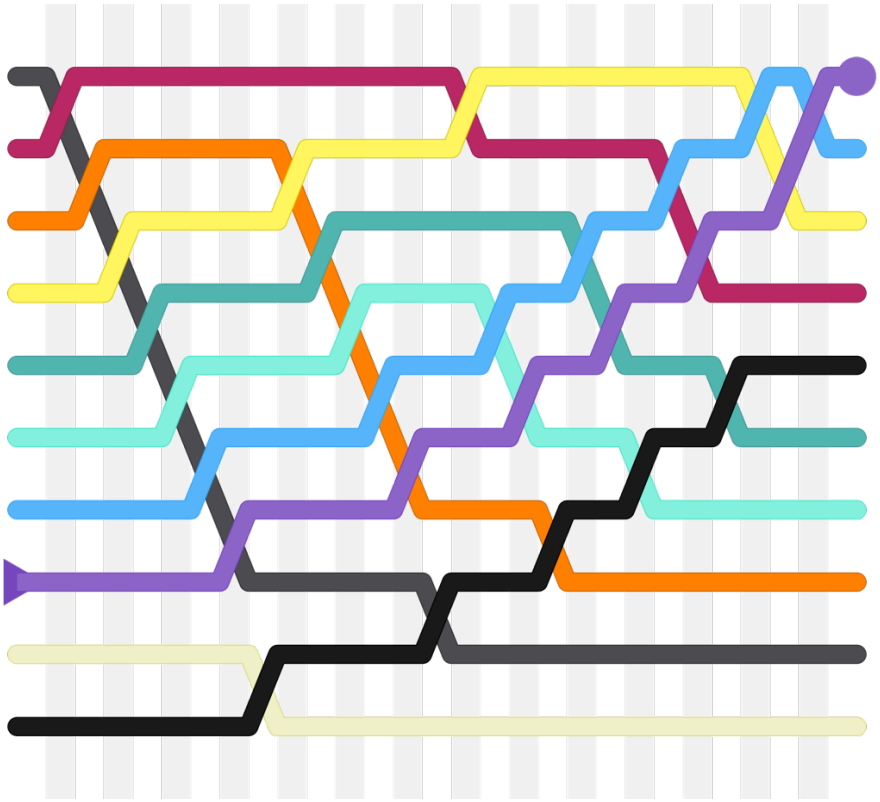


Рисунок 1.3. Статична візуалізація бульбашкового сортування

Бульбашкове сортування займає мінімум часу (порядку n), коли елементи вже відсортовано. Тому найкраще заздалегідь перевірити, чи масив уже відсортований чи ні, щоб уникнути $O(n!)$ часової складності.

Найгірший випадок для бульбашкового сортування виникає, коли елементи масиву розташовані в порядку зменшення. У гіршому випадку загальна кількість ітерацій або проходів, необхідних для сортування даного масиву, становить $(n-1)$. де n — кількість елементів, присутніх у масиві.

Бульбашкове сортування виконує заміну суміжних пар без використання будь-якої основної структури даних. Отже, алгоритм бульбашкового сортування є алгоритмом на місці, а також він є стабільним алгоритмом.

Через свою простоту бульбашкове сортування часто використовується для введення поняття алгоритму сортування. У комп'ютерній графіці він популярний завдяки своїй здатності виявляти крихітну помилку (наприклад, заміну лише двох елементів) у майже відсортованих масивах і виправляти її лише з лінійною складністю ($2n$) [21].

Складність по часу	Найкраща	$O\ (n)$
	Средня	$O\ (n!)$
	Найгірша	$O\ (n!)$
Складність по пам’яті	Загальна	$O(n)$
	Додаткова	$O(1)$

Таблиця 1.2. Характеристика алгоритму сортування злиття

1.1.4 Пірамідальне сортування

Пірамідальне сортування (або сортування купою, HeapSort) — це метод сортування порівнянням, що ґрунтується на такій структурі даних як двійкова купа. Вона схожа на сортування вибором, де спочатку шукаємо максимальний елемент і поміщаємо його в кінець. Далі ми повторюємо ту ж операцію для елементів, що залишилися.

Закінчене двійкове дерево - це двійкове дерево, в якому кожен рівень, за винятком, можливо, останнього, має повний набір вузлів, і все листя розташоване якомога лівіше. Двійкова купа - це закінчене двійкове дерево, в якому елементи зберігаються в особливому порядку: значення в батьківському вузлі більше (або менше) значень його двох дочірніх вузлів. Перший варіант називається max-heap, а другий - min-heap. Купа може бути представлена двійковим деревом чи масивом.

Оскільки двійкова купа це закінчене двійкове дерево, її можна легко уявити у вигляді масиву, а уявлення на основі масиву є ефективним з точки зору витрати пам'яті. Якщо батьківський вузол зберігається в індексі I, лівий дочірній

елемент може бути обчислений як $2I + 1$, а правий дочірній елемент як $2I + 2$ (за умови, що індексування починається з 0).

Алгоритм пірамідального сортування в порядку зростання:

- Побудуйте max-heap із вхідних даних.
- На цьому етапі найбільший елемент зберігається в корені купи. Замініть його на останній елемент купи, а потім зменшіть розмір на 1. Нарешті, перетворіть отримане дерево в max-heap з новим коренем.
- Повторюйте вищезгадані кроки, поки розмір купи більше 1.

Процедура перетворення на купу (далі процедура heapify) може бути застосована до вузла, тільки якщо його дочірні вузли вже перетворені. Таким чином, перетворення має виконуватися знизу нагору.

Переваги heapsort:

- Ефективність – час, необхідний для сортування купи, збільшується логарифмічно, тоді як інші алгоритми можуть зростати експоненціально повільніше, оскільки збільшується кількість елементів для сортування. Цей алгоритм сортування дуже ефективний.
- Використання пам’яті – використання пам’яті є мінімальним, оскільки окрім того, що необхідно для зберігання початкового списку елементів для сортування, для роботи не потрібен додатковий простір пам’яті.
- Простота – його простіше зрозуміти, ніж інші настільки ж ефективні алгоритми сортування, оскільки він не використовує передові концепції інформатики, такі як рекурсія [9].

Алгоритм пірамідального сортування має обмежене застосування, тому що Quicksort (Швидке сортування) та Mergesort (Сортування злиттям) на практиці краще. Проте сама структура даних купи використовується досить часто. Пірамідальне сортування в основному використовується в гібридних алгоритмах, таких як IntroSort. Сортувати майже відсортований (або K-відсортований) масив k найбільших (або найменших) елементів у масиві [10].

Складність по часу	Найкраща	$O\ (n\ \log(n))$
	Средня	$O\ (n\ \log(n))$
	Найгірша	$O\ (n\ \log(n))$
Складність по пам’яті	Загальна	$O(1)$

Таблиця 1.3. Характеристика алгоритму пірамідального сортування

1.1.5 Алгоритм інтросортування

Алгоритм інтросортування (Introsort або інтроспективне сортування) є ефективним алгоритмом сортування на місці, який зазвичай перевершує всі інші алгоритми сортування з точки зору продуктивності. Завдяки своїй високій продуктивності він використовується в декількох стандартних бібліотечних функціях сортування [12].

Introsort — це гібридний алгоритм, винайдений Девідом Массером у 1993 році з метою надання загального алгоритму сортування для стандартної бібліотеки C++. Класична реалізація інтросортування передбачає рекурсивний Quicksort із поверненням до Heapsort у випадку, якщо рівень глибини рекурсії досягає певного максимуму. Максимум залежить від кількості елементів у колекції та зазвичай становить $2\log\ (n)$. Причина цього «резервного варіанту» полягає в тому, що якщо Quicksort не зміг отримати рішення після $2\log\ (n)$ рекурсій для гілки, ймовірно, він потрапив у найгірший випадок і погіршився до $O(\ n^{!}\ )$. Щоб ще більше оптимізувати цей алгоритм, швидка реалізація вводить додатковий крок у кожній рекурсії, де розділ сортується за допомогою InsertionSort, якщо кількість розділів менше 20. Число 20 — це емпіричне число, отримане шляхом спостереження за поведінкою InsertionSort зі списками такого розміру [11].

Вибір правильного алгоритму сортування залежить від випадку, де використовується алгоритм сортування. Уже існує велика кількість алгоритмів сортування, які мають свої плюси та мінуси. Отже, щоб отримати кращий алгоритм сортування, рішення полягає в тому, щоб налаштувати існуючі алгоритми та створити новий алгоритм сортування, який працює краще. Існує багато гібридних алгоритмів, які перевершують загальні алгоритми сортування. Одним з таких є Introsort. Найкращі версії Quicksort конкурентоспроможні як із сортуванням купи, так і сортуванням злиттям на переважній більшості вхідних даних. Рідко Quicksort має найгірший випадок часу роботи $O(N!)$ і використання стека $O(N)$. Як Heapsort, так і Mergesort мають найгірший час роботи $O(N\ \log(N))$, разом із використанням стека $O(1)$ для Heapsort і $O(\log(N))$ для Mergesort відповідно. Крім того, сортування вставкою працює краще, ніж будь-який із наведених вище алгоритмів, якщо набір даних невеликий.

Поєднуючи в собі всі плюси алгоритмів сортування, Introsort поводиться на основі набору даних. Якщо кількість елементів у вхідних даних стає меншою, Introsort виконує сортування за вставкою для вхідних даних. Маючи на увазі найменшу кількість порівнянь (Quicksort), для розбиття масиву шляхом пошуку опорного елемента використовується Quicksort. Як уже згадувалося раніше, найгірший варіант Quicksort базується на двох фазах, і ось як ми можемо їх виправити.

- Вибір опорного елемента: ми можемо використовувати концепцію медіани з 3 або випадкову опорну концепцію або середину як опорну концепцію для пошуку опорного елемента
- Глибина рекурсії під час виконання алгоритму: коли глибина рекурсії стає вищою, Introsort використовує Heapsort, оскільки він має певну верхню межу $O(N\log(N))$.

Чи можна застосовувати Introsort всюди? Якщо дані не поміщаються в масив, Introsort не можна використовувати. Крім того, як Quicksort і Heapsort, Introsort не є стабільним. Якщо потрібне стабільне сортування, Introsort не можна застосувати [13].

Складність по часу	Найкраща	O (n)
	Средня	O (n log n)
	Найгірша	
Складність по пам’яті	Загальна	O(n)
	Додаткова	O(1)

Таблиця 1.4. Характеристика алгоритму інтросортування

1.1.6 Тімсорт

Тімсорт (Timsort) — це адаптивний алгоритм сортування, який потребує $O(n\ \log(n))$ порівнянь для сортування масиву з n елементів. Він був розроблений і реалізований Тімом Пітерсом у 2002 році на мові програмування Python. Це стандартний алгоритм сортування Python з версії 2.3. Він є найшвидшим алгоритмом сортування. TimSort — це алгоритм сортування, заснований на сортуванні вставкою та сортуванні злиттям.

Основний підхід, який використовується в алгоритмі сортування Тіма, такий: спочатку відсортуйте маленькі фрагменти за допомогою сортування вставкою, а потім об’єднайте всі великі фрагменти за допомогою функції злиття сортування злиттям [14]. Ми ділимо масив на блоки, відомі як Run. Ми сортуємо ці прогони за допомогою сортування вставкою один за одним, а потім об’єднуємо ці прогони за допомогою функції об’єднання, яка використовується під час сортування злиттям. Якщо розмір масиву менший, ніж run, масив сортується лише за допомогою сортування вставкою. Розмір серії може варіюватися від 32 до 64 залежно від розміру масиву. Зверніть увагу, що функція

злиття добре працює, коли розмір підмасивів є степенем 2. Ідея базується на тому факті, що сортування вставкою добре працює для малих масивів.

Timsort — це алгоритм сортування, який ефективний для даних реального світу, а не створений в академічній лабораторії. Timsort спочатку аналізує список, який намагається відсортувати, а потім обирає підхід, заснований на аналізі списку. З часу створення алгоритму він використовувався як стандартний алгоритм сортування в Python, Java, платформі Android і в GNU Octave. Велике позначення Тімсорта O — O(n log n) [15].

Складність по часу	Найкраща	O (n)
	Средня	O (n log n)
	Найгірша	
Складність по пам’яті	Загальна	O(n)

Таблиця 1.5. Характеристика алгоритму тімсорт

1.2 Основні визначення та термінологія нейронних мереж

Нейронна мережа — це серія алгоритмів, які намагаються розпізнати базові зв’язки в наборі даних за допомогою процесу, який імітує роботу людського мозку. У цьому сенсі нейронні мережі відносяться до систем нейронів, органічних або штучних за своєю природою.

Нейронні мережі можуть адаптуватися до змін вхідних даних; таким чином мережа генерує найкращий можливий результат без необхідності переробки критеріїв виводу. Концепція нейронних мереж, яка сягає своїм корінням у штучний інтелект, стрімко набирає популярність у розробці торгових систем [17].

Існує три основні компоненти: вхідний рівень, рівень обробки та вихідний рівень. Вхідні дані можуть бути зважені на основі різних критеріїв. Усередині шару обробки, який прихований від очей, є вузли та зв'язки між цими вузлами, які мають бути аналогічними нейронам і синапсам у мозку тварин.

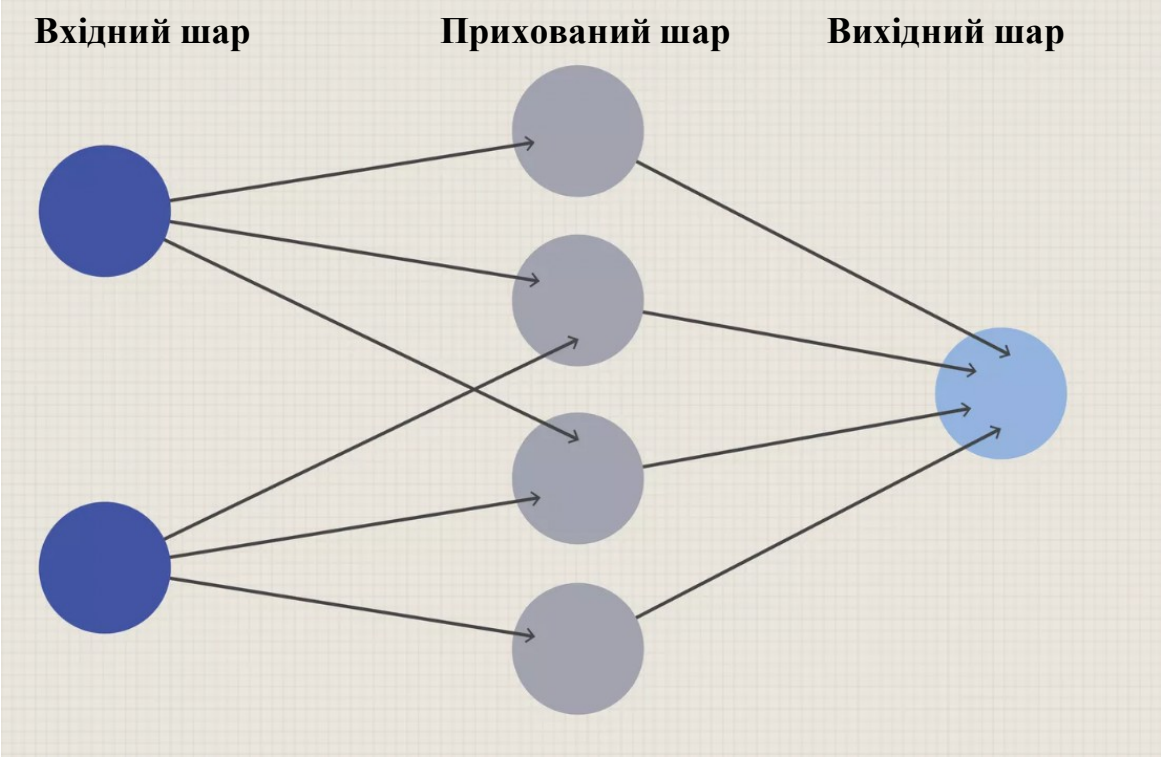


Рисунок 1.3. Проста нейронна мережа

Нейронна мережа працює подібно до нейронної мережі людського мозку. «Нейрон» у нейронній мережі — це математична функція, яка збирає та класифікує інформацію відповідно до певної архітектури. Мережа дуже схожа на статистичні методи, такі як підгонка кривої та регресійний аналіз.

Нейронна мережа містить шари взаємопов'язаних вузлів. Кожен вузол є перцептроном і схожий на множинну лінійну регресію. Перцептрон передає сигнал, створений множинною лінійною регресією, у функцію активації, яка може бути нелінійною [17].

Нейрони можуть розглядатися як елементи обробки (ЕО) у мережі. Посилання або зв'язок – це шлях до даних між двома нейронами. Кожен нейрон посилає імпульси іншим нейронам і приймає імпульси від інших нейронів.

Основна робота нейрона зображена на рис. 1.4, де $x_0, x_1, \ldots, x_n$ – значення вхідних даних, які проходять через посилання, $w_0, w_1, \ldots, w_n$ – це ваги для цих зв'язків, f – функція активації в кожному нейроні, y – значення вихідних даних, а θ є константою, яка представляє порогове значення. Усі зв'язки зважені, так що значення даних множаться на вагу посилання, яке вони передали [16].

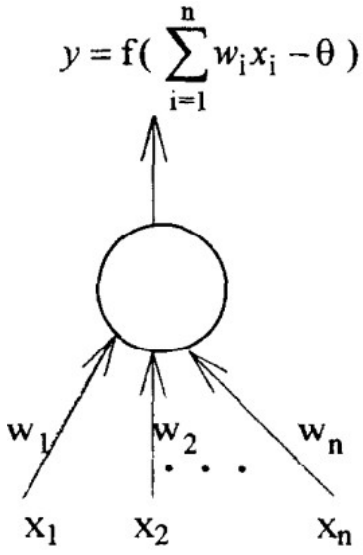


Рисунок 1.4. Робота нейрона.

Рисунок 1.5. Графи різних типів нейронних мереж

Розглянемо переваги та недоліки, які підштовхнули до використання нейронних мереж у задачі сортування.

Нейронні мережі, можуть працювати безперервно та є більш ефективними, ніж простіші аналітичні моделі. Нейронні мережі також можна запрограмувати на навчання на попередніх виходах для визначення майбутніх результатів на основі подібності до попередніх вхідних даних.

Нейронні мережі, які використовують хмарні онлайн-сервіси, також мають переваги зменшення ризиків порівняно з системами, які покладаються на апаратне забезпечення локальних технологій. Крім того, нейронні мережі часто

можуть виконувати кілька завдань одночасно (або, принаймні, розподіляти завдання, які повинні виконуватися модульними мережами одночасно).

Нарешті, нейронні мережі постійно розширюються в нові програми. Хоча ранні теоретичні нейронні мережі були дуже обмежені в застосуванні в різних галузях, сьогодні нейронні мережі використовуються в медицині, науці, фінансах, сільському господарстві та безпеці.

Хоча складність нейронних мереж є сильною стороною, це може означати, що на розробку певного алгоритму для конкретного завдання потрібні місяці (якщо не більше). Крім того, може бути важко помітити будь-які помилки або недоліки в процесі, особливо якщо результати є оцінками або теоретичними діапазонами.

Нейронні мережі також може бути важко перевірити. Деякі процеси нейронної мережі можуть виглядати як «чорна скринька», де вводяться вхідні дані, мережі виконують складні процеси та повідомляють про результати. Людям також може бути важко проаналізувати слабкі сторони в процесі обчислення або навчання мережі, якщо мережі бракує загальної прозорості щодо того, як модель вивчає попередні дії [17].

1.3 Глибоке навчання

Глибоке навчання — це підмножина машинного навчання, яке, по суті, є нейронною мережею з трьома або більше рівнями. Ці нейронні мережі намагаються змоделювати поведінку людського мозку — хоча й далеко не збігаються з його можливостями — дозволяючи йому «навчатися» на великих обсягах даних. Хоча нейронна мережа з одним шаром все ще може робити приблизні прогнози, додаткові приховані шари можуть допомогти оптимізувати та покращити точність.

Глибоке навчання керує багатьма додатками та службами штучного інтелекту, які покращують автоматизацію, виконуючи аналітичні та фізичні завдання без втручання людини. Технологія глибокого навчання лежить в основі повсякденних продуктів і послуг (таких як цифрові помічники, пульти дистанційного керування телевізором із голосовою підтримкою та виявлення шахрайства з кредитними картками), а також нових технологій (таких як безпілотні автомобілі).

1.3.1 Відмінність від класичного машинного навчання

Глибоке навчання відрізняється від класичного машинного навчання типом даних, з якими воно працює, і методами, за допомогою яких воно навчається.

Алгоритми машинного навчання використовують структуровані дані з мітками для прогнозування, тобто конкретні функції визначаються на основі вхідних даних для моделі та організовуються в таблиці. Це не обов’язково означає, що він не використовує неструктуровані дані; це просто означає, що якщо так, то зазвичай проходить певну попередню обробку, щоб організувати його в структурований формат.

Глибоке навчання усуває частину попередньої обробки даних, яка зазвичай пов’язана з машинним навчанням. Ці алгоритми можуть завантажувати й обробляти неструктуровані дані, як-от текст і зображення, і автоматизувати виділення функцій, усуваючи певну залежність від експертів-людей. Наприклад, скажімо, у нас є набір фотографій різних домашніх тварин, і ми хочемо класифікувати їх за категоріями «кіт», «собака», «хом’як» тощо. Алгоритми глибокого навчання можуть визначити, які особливості (наприклад, вуха) є найважливішими, щоб відрізнити кожну тварину від іншої. У машинному навчанні ця ієрархія функцій встановлюється вручну людиною-експертом.

Потім, за допомогою процесів градієнтного спуску та зворотного поширення, алгоритм глибокого навчання налаштовується та підлаштовується для точності, дозволяючи робити прогнози з підвищеною точністю.

Моделі машинного та глибокого навчання також здатні до різних типів навчання, які зазвичай класифікуються як контрольоване навчання, неконтрольоване навчання та навчання з підкріпленням. Контрольоване навчання використовує мічені набори даних для класифікації або прогнозування; це вимагає певного втручання людини, щоб правильно позначити вхідні дані. Навпаки, неконтрольоване навчання не потребує мічених наборів даних, натомість воно виявляє шаблони в даних, кластеризуючи їх за будь-якими відмінними характеристиками. Навчання з підкріпленням – це процес, у якому модель вчиться ставати точнішою для виконання дії в середовищі на основі зворотного зв’язку, щоб максимізувати винагороду.

1.3.2 Принципи роботи

Нейронні мережі глибокого навчання, або штучні нейронні мережі, намагаються імітувати людський мозок за допомогою комбінації вхідних даних, ваг і зміщення. Ці елементи працюють разом, щоб точно розпізнавати, класифікувати та описувати об’єкти в даних.

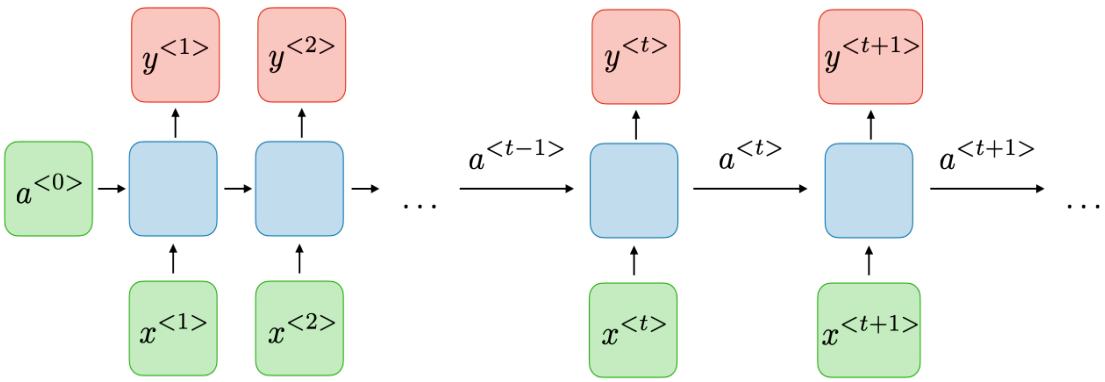
Глибокі нейронні мережі складаються з кількох шарів взаємопов’язаних вузлів, кожен з яких будується на попередньому шарі для уточнення й оптимізації прогнозу або категоризації. Цей хід обчислень через мережу називається прямим поширенням. Вхідний і вихідний шари глибокої нейронної мережі називаються видимими шарами. Вхідний рівень – це місце, де модель глибокого навчання отримує дані для обробки, а вихідний рівень – це місце, де робиться остаточний прогноз або класифікація.

Інший процес, який називається зворотним поширенням, використовує такі алгоритми, як градієнтний спуск, для обчислення помилок у передбаченнях, а потім коригує ваги та зміщення функції, переміщаючись назад між шарами, щоб навчити модель. Разом пряме та зворотне поширення дозволяють нейронній мережі робити прогнози та відповідно виправляти будь-які помилки. З часом алгоритм стає точнішим.

Вище описано найпростіший тип глибокої нейронної мережі в найпростіших термінах. Однак алгоритми глибокого навчання неймовірно складні, і існують різні типи нейронних мереж для вирішення конкретних проблем або наборів даних [22].

1.4 Рекурентна нейронна мережа

Архітектура традиційної RNN – рекурентні нейронні мережі, також відомі як RNN, є класом нейронних мереж, які дозволяють використовувати попередні виходи як вхідні, маючи приховані стани. Зазвичай архітектура виглядає так, як зображено на рисунку 1.6.



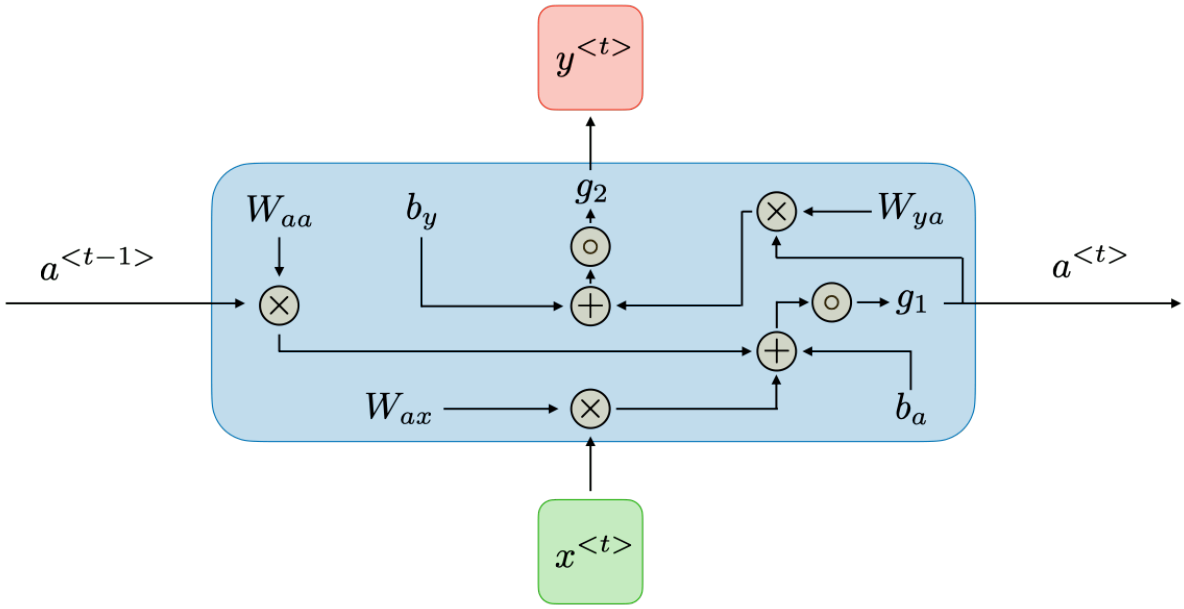


Рисунок 1.6. Архітектура рекурентної нейронної мережі

Для кожного часового кроку t активація $a^{<t>}$ та вихід $y^{<t>}$ виражаються таким чином:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a), \tag{1.1}$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y), \tag{1.2}$$

де $W_{aa}, W_{ax}, W_{ya}, b_a, b_y$ – коефіцієнти, що поділяють тимчасову та g_1, g_2 функцію активації [23].

Перевагами рекурентної нейронної мережі є:

– Можливість обробки вхідних даних будь-якої довжини.
– Розмір моделі не збільшується разом із розміром вхідних даних.
– Обчислення враховують історичну інформацію.
– Вага розподіляється в часі.

У випадку рекурентної нейронної мережі функція втрат f усіх часових кроків визначається на основі втрат на кожному часовому кроці наступним чином:

$$f(\mathfrak{X}, y) = \sum_{t-\$}^{y} f(\mathfrak{X}^{\&t}, y^{\&t}), \tag{1.3}$$

Зворотне поширення здійснюється в кожен момент часу. На кроці часу T похідна втрати f відносно вагової матриці W виражається таким чином:

$$\frac{6f^T}{6W} = \sum_{t-\$}^{T} \frac{6f^T}{6W}\mathrm{F}_{(t)}, \tag{1.4}$$

Найпоширеніші функції активації, які використовуються в модулях RNN, описані на рисунку 1.7 [23].

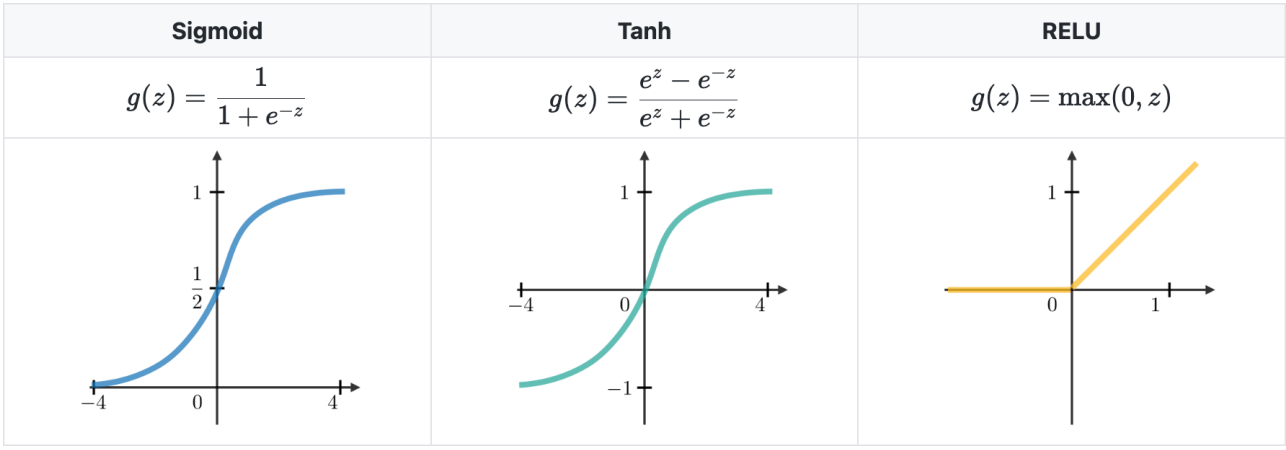


Рисунок 1.7. Найпоширеніші функції активації.

В рамках задачі сортування ми розглядаємо традиційний тип рекурентної нейронної мережі один-до-одного (One-To-One). Архітектуру цього типу зображено на рисунку 1.8.

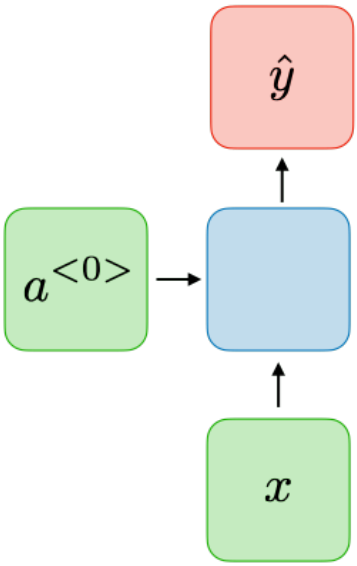


Рисунок 1.8. Архітектура RNN One-To-One

Висновки

Алгоритм сортування — це алгоритм, що складається з серії інструкцій, які приймають масив як вхідні дані, виконують певні операції над масивом, який іноді називають списком, і виводять відсортований масив.

Алгоритми сортування можна класифікувати за такими групами: сортування на місці (якщо алгоритм не використовує додаткового простору), стабільне або нестабільне сортування (якщо алгоритм після сортування вмісту не змінює послідовність подібного вмісту), адаптивне або неадаптивне сортування (якщо алгоритм використовує переваги вже «відсортованих» елементів у списку).

Ми розглянули найпопулярніші класичні алгоритми для сортування, так як: швидке сортування, сортування бульбашкою, сортування злиттям, пірамідальне сортування, інтросортування та тімсорт.

Нейронна мережа — це серія алгоритмів, які намагаються розпізнати базові зв’язки в наборі даних за допомогою процесу, який імітує роботу людського мозку.

Глибоке навчання — це підмножина машинного навчання, яке, по суті, є нейронною мережею з трьома або більше рівнями. Рекурентна нейронна мережа відноситься до моделей глибокого навчання. Архітектура традиційної рекурентні нейронні мережі дозволяє використовувати попередні виходи як вхідні, маючи приховані стани. В рамках задачі сортування ми розглядаємо традиційний тип рекурентної нейронної мережі один-до-одного

РОЗДІЛ 2 НЕЙРОННІ МЕРЕЖІ У СОРТУВАННІ ДАННИХ

2.1 Постановка задачі

Мета даної роботи побудувати модель для сортування на основі нейронної мережі та порівняти кінцеві результати з класичними алгоритмами, описаними в попередньому розділі. Останнім часом нейронні мережі набули великої популярності та широкого використання в різних областях. Цікаво дослідити доцільність використання нейронних мереж у сортуванні даних. Чи зможе алгоритм сортування на основі нейронної мережі перевершити за складністю по часу або по пам’яті класичні алгоритми сортування.

На відміну від інших проблем машинного навчання, ця вже вирішується класичними методами (наприклад, швидке сортування). І, звісно, дані, які ми можемо згенерувати для передачі в модель машинного навчання, щоб навчитися сортувати, нескінченні, тож що робить цю проблему цікавою, це те, як мало даних моделі машинного навчання знадобиться, щоб навчитися сортувати.

Завдання сортування елементів масивів даних є класичним, це одне з найважливіших завдань програмування. У класичному вигляді в її реалізації використовують точні алгоритми. Проведення експерименту для сортування масиву даних дозволить визначити ефективність нейронних мереж у галузі аналізу та обробки масивів даних.

До того ж, нейронні мережі часто розглядаються як просто математичні функції, які не мають реального значення: ми знаємо вхідні та вихідні дані нейронної мережі, але не можемо зрозуміти міркування, які використовуються для визначення виходу. Таким чином, це був би новий приклад, коли нейронна мережа пояснюється сама собою, і він показав би, що нейронні мережі можуть вивчити ефективні алгоритми сортування. Ми також маємо потенціал вивчити ще більш ефективні алгоритми сортування або алгоритми наближеного сортування, які є приблизно правильними, але швидшими за відомі алгоритми.

2.2 Опис моделей

Однією з найбільш корисних властивостей, яку ми можемо мати, це здатність для моделі вивчати алгоритм, який узагальнює довші вхідні послідовності. Зокрема, фактична архітектура моделі легко потребуватиме здатності узагальнювати до більш довгих послідовностей, а алгоритм навчання, який її навчає, повинен буде відповідним чином упорядкувати, щоб модель не підганялася до коротших послідовностей, які вона бачить під час навчання.

Ця проблема, очевидно, вимагає від нас вивчення функції, яка відображає послідовності в послідовності, і очевидним способом виконати це буде використання архітектури RNN кодера-декодера, яка використовується для проблем перекладу. Можна зробити краще, використовуючи загальний варіант цієї архітектури: збільшення за допомогою механізму уваги.

Було вирішено використати в даному дослідженні моделі послідовність-до-послідовності (sequence-to-sequence) та мережі вказівників (pointer networks). Далі буде наведено детальніший опис кожної моделі.

2.2.1 Модель послідовність-до-послідовності

Моделі sequence-to-sequence (скорочено Seq2Seq) — це моделі глибокого навчання, які досягли значного успіху в таких завданнях, як машинний переклад, підсумовування тексту та створення підписів до зображень. Google Translate почав використовувати таку модель у виробництві наприкінці 2016 року. Ці моделі вперше були описані у роботах [24] та [25].

Ця модель приймає послідовність елементів (слів, літер, часових рядів тощо) і виводить іншу послідовність елементів.

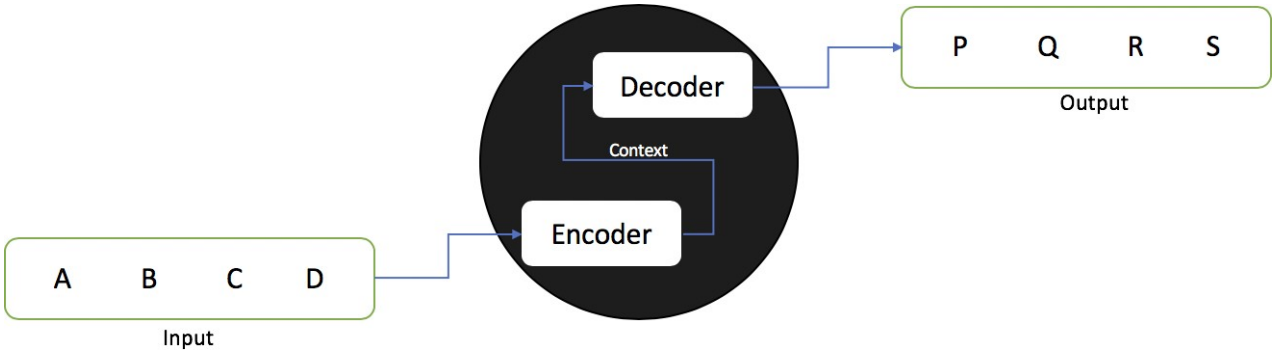


Рисунок 2.1. Модель Seq2Seq

Модель складається з кодера і декодера. Кодер фіксує контекст вхідної послідовності у вигляді прихованого вектора стану та надсилає його в декодер, який потім створює вихідну послідовність. Оскільки завдання базується на послідовності, і кодер, і декодер, як правило, використовують певну форму RNN, LSTM, GRU тощо. Прихований вектор стану може мати будь-який розмір, хоча в більшості випадків він приймається як ступінь 2 і велике число (256, 512, 1024), яке може певним чином представляти складність повної послідовності, а також домену.

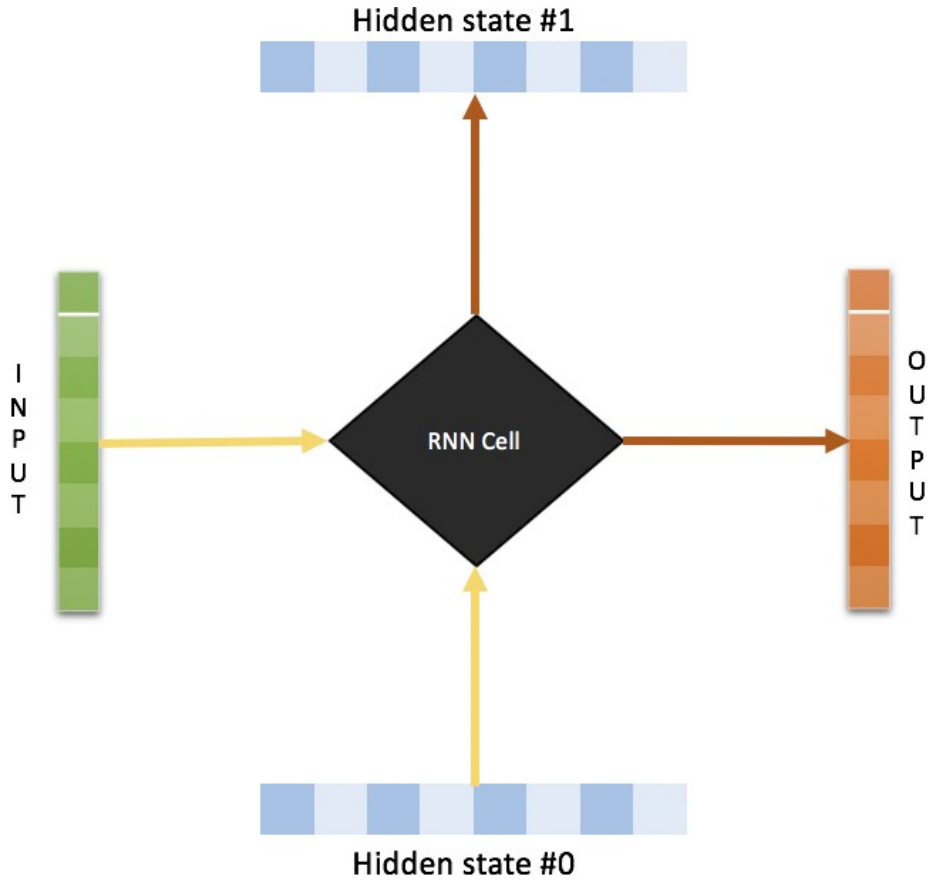


Рисунок 2.2. Осередок RNN

RNN за дизайном приймають два входи, поточний приклад, який вони бачать, і представлення попереднього входу. Таким чином, вихід на кроці часу t залежить від поточного входу, а також входу на момент часу t-1. Саме тому вони працюють краще, коли ставляться до послідовних завдань. Послідовна інформація зберігається в прихованому стані мережі та використовується в наступному випадку.

Кодер, що складається з RNN, приймає послідовність як вхідні дані та генерує остаточне вбудовування в кінці послідовності. Потім це надсилається до декодера, який потім використовує його для передбачення послідовності, і після кожного наступного передбачення він використовує попередній прихований стан, щоб передбачити наступний екземпляр послідовності [26].

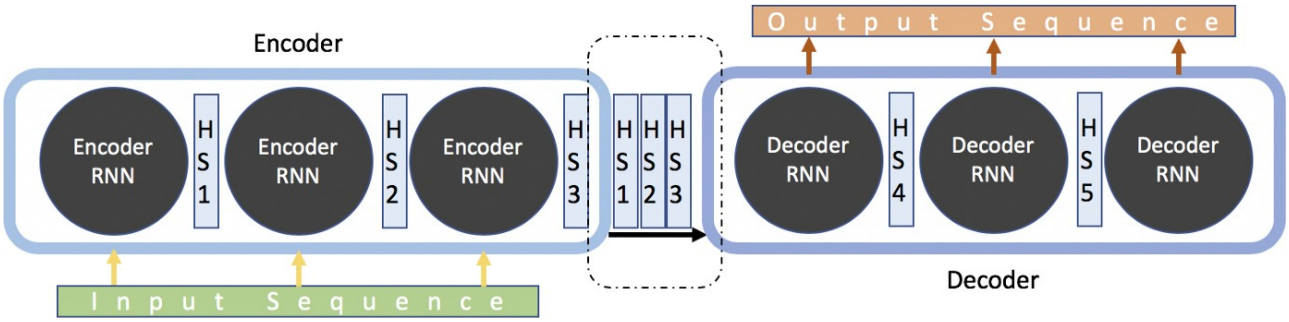


Рисунок 2.3. Модель кодера-декодера для моделювання Seq2Seq

На відміну від багатьох проблем послідовність-до-послідовності, вихід моделі фактично буде елементами входу. Багато проблем послідовності є певним перетворенням між доменами, і, отже, очевидно, виходи відповідатимуть входам приблизно один до одного, залежно від домену. Але при сортуванні виходи відповідають входам точно один до одного, тобто вихід є перестановкою вхідних даних.

2.2.2 Мережа вказівників

Мережі вказівників (Pointer network) є різновидом моделі послідовності до послідовності з увагою. Замість перекладу однієї послідовності в іншу вони видають послідовність покажчиків на елементи вхідної серії. Основне використання цього — упорядкування елементів послідовності або набору змінної довжини.

Мережі покажчиків ефективно створюють механізм уваги для змінної кількості токенів. Враховуючи послідовність токенів (t_3, $t_\$$, ... , $t_{n\#\$}$), мережа вказівників обслуговує вхідний словник кандидатів для наступного токена t_n. Кожен із цих кандидатів на вхід має пов’язаний вектор вбудовування, створений кодувальником. Подібним чином, кожен з маркерів послідовності має власні вбудовані декодери. На відміну від інших моделей передбачення послідовності,

мережі покажчиків побудовані таким чином, що кількість кандидатів на вхід може змінюватися під час висновку.

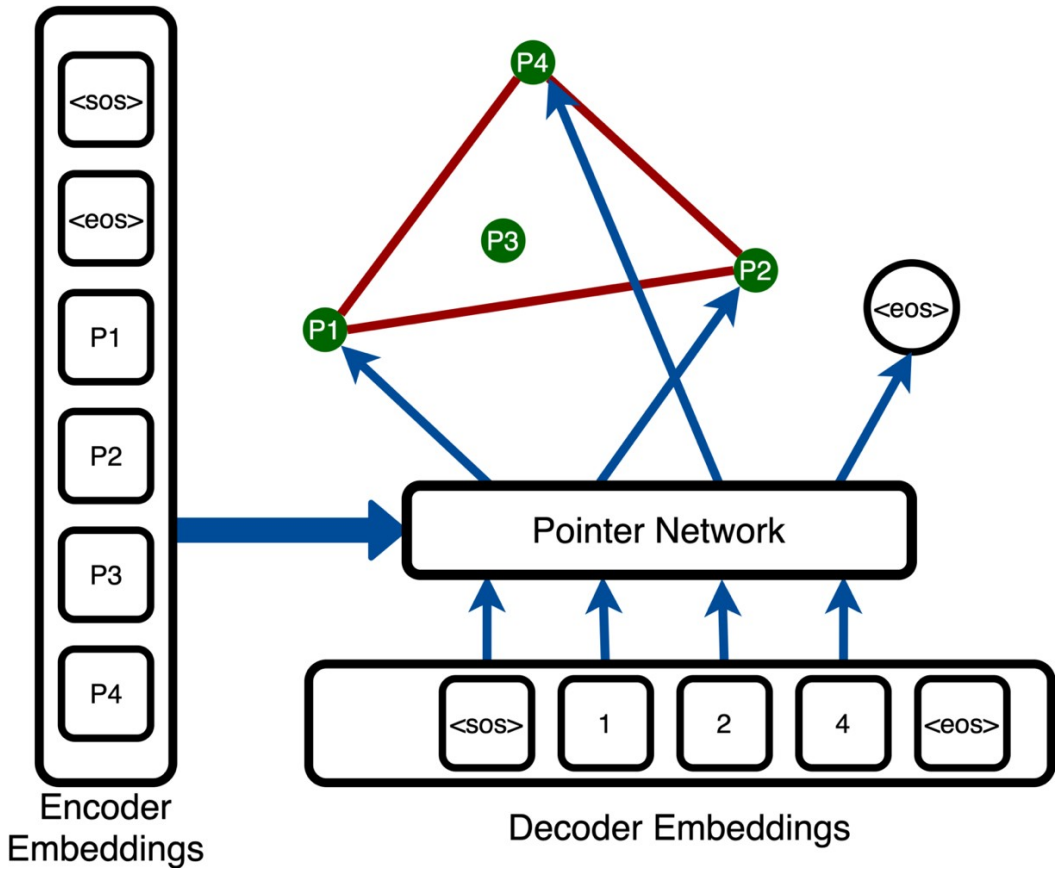


Рисунок 2.4. Архітектура мережі вказівників

Ключовим нововведенням мережі покажчиків є механізм уваги з ретельно розробленою структурою. Пара формул 2.1 нижче виражає ядро логіки мережі вказівників. Спочатку він складається з двох вагових коефіцієнтів, які можна вивчати, помножених на вбудовування кодера та вбудовування декодера, відповідно, після чого слідує нелінійність і множення на інший вивчений ваговий коефіцієнт v, який зменшує розмірність функції. Цей рівень по суті поєднує в собі функції введення та цільового вбудовування, щоб створити дводольний граф активацій, що відповідає парам вхід-ціль. Нарешті, шар *softmax* виражає умовну ймовірність у вимірі, що відповідає нашим введенням. Те, що нам залишається (для кожного рядка в пакеті), — це рядок імовірностей між вхідними токенами для будь-якої позиції у вихідній послідовності [30].

У мережі вказівників ці ваги/маски уваги не використовуються далі для обчислення вектора контексту для наступного часового кроку. Ці ваги вважаються покажчиками на вхідну послідовність. Вхідний часовий крок, що має найвищу вагу, вважається вихідним для цього часового кроку декодера [29].

$$u_j^{"} = v^T \tanh\#W_1 e_j + W_2 d_i\$ \qquad j \in (1, \dots, n)$$
$$p(C_i | C_\$, \dots, C_{i\#\$}, P) = softmax(u^i) \tag{2.1}$$

З рівняння 2.1 можна побачити, що операція *softmax* над *u* далі не використовується для обчислення вектора контексту для подачі як інформації на поточний крок декодера. Вихідні дані операції *softmax* вказують на вхідний маркер із максимальним значенням.

Проста та елегантна архітектура мережі вказівників (Pointer network, PTR-Net) вирішує тонку складність у проблемах передбачення послідовності. Скажімо, ми хочемо передбачити послідовність індексів над вхідною послідовністю. Що ми робимо, якщо довжина цього введення є змінною? З попередніми методами розмір словника (тобто індекси вхідної послідовності) повинен бути фіксованим апріорі. Це добре для таких проблем, як створення речень, де вхідний словник є набором відомих символів. З іншого боку, комбінаторні проблеми — це те, де входять мережі вказівників. Мережі вказівників дозволяють нам розв’язувати задачі комбінаторної оптимізації, де метою є послідовність індексів вхідних даних, які визначаються під час висновку, а не під час навчання. Стаття про вказівні мережі охоплює три таких завдання: плоскі опуклі оболонки, тріангуляцію Делоне та симетричну плоску проблему комівояжера [30].

Як згадувалося раніше, мережі вказівників застосовуються до комбінаторних задач, таких як вирішення плоских опуклих оболонок. Мережі покажчиків також з’являються в різноманітних останніх програмах. PolyGen [27], унікальна генеративна модель для тривимірних сіток використовує мережу

вказівників для призначення вершин даній грані, призначаючи по одній грані за раз, доки не буде побудована топологія сітки. Мережа вказівників навіть використовується для вибору помітних речень, які добре підсумовують документ у [28].

2.3 Огляд літератури

Чимало досліджень було проведено з метою аналізу здатності нейронних мереж до якісного сортування даних. Основна мета досліджень полягає в тому, щоб розширити продуктивність сортування класичних алгоритмів. Існує багато цікавих ідей, які можна застосувати до цієї проблеми.

2.3.1 Сортування з урахуванням розподілу даних на основі нейронної мережі

Сортування широко використовується в багатьох обчислювальних завданнях, наприклад у базі даних додатків і роботи з обробки великих даних. Вчені у даному досліджені представили NN-sort, метод сортування на основі нейронної мережі та розподілу даних. NN-sort використовує для сортування модель, навчену на історичних даних для сортування майбутніх даних. NN-sort використовує кілька ітерацій для зменшення конфліктів під час процесу сортування, які спостерігаються як основним вузьким місцем продуктивності у використанні моделей DNN для розв’яння задачі на сортування.

Також у дослідженні надано комплексний аналіз алгоритму NN-sort, включаючи його межу складності, вартість моделі для опису, як знайти потрібний баланс між різними факторами, такими як точність моделі та

ефективність сортування. Експериментальні результати демонструють, що NN-sort перевершує традиційні алгоритми сортування до 10,9x. Дотримуючись цієї нитки дослідження, вчені вивчають, як ефективно застосувати такий підхід до програм, щоб покращити продуктивність їхньої фази сортування, що зрештою може принести користь для загальної ефективності програми або системи [18].

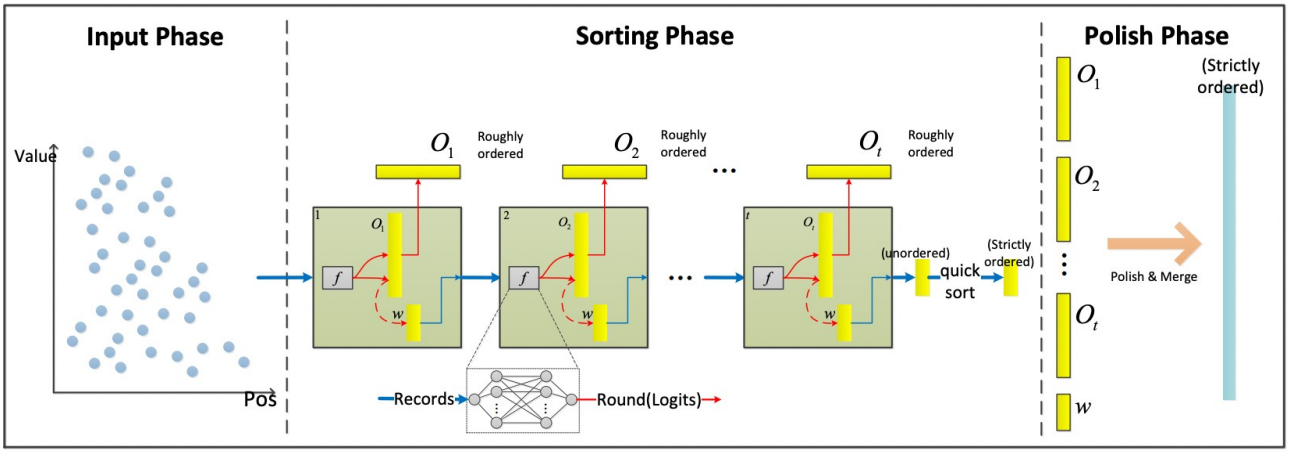


Рисунок 2.5. Архітектура NN-sort

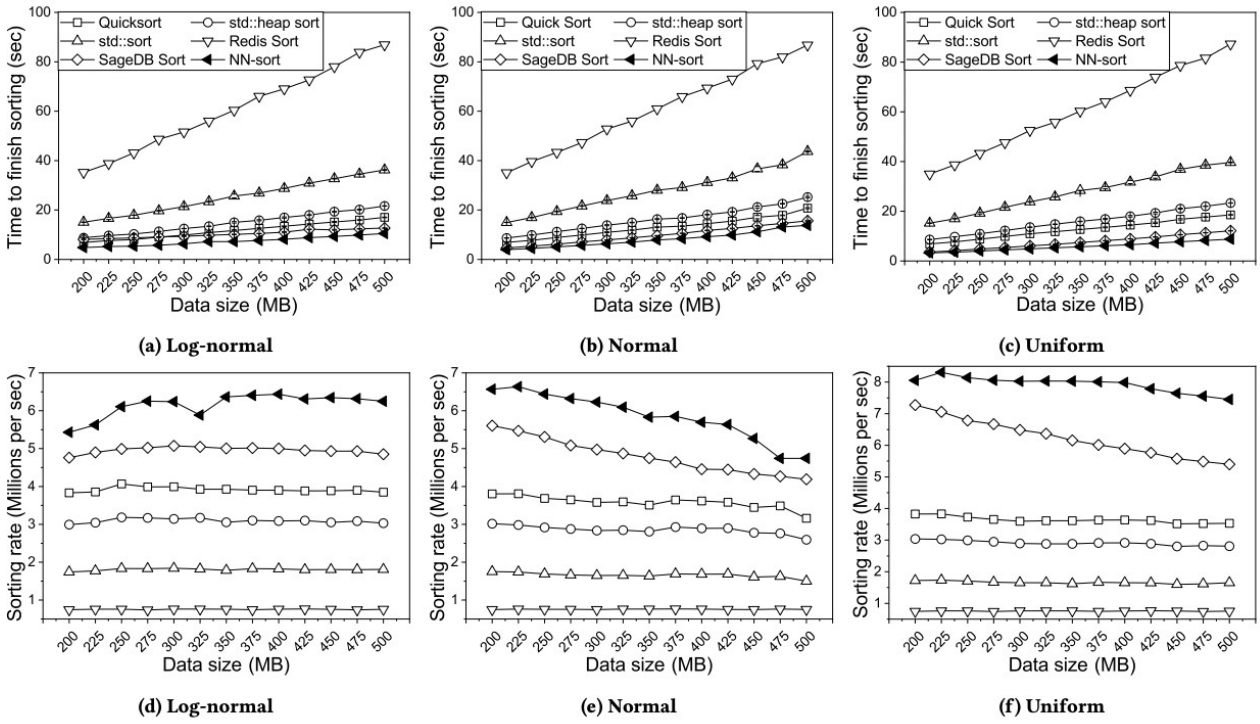


Рисунок 2.6. Продуктивність NN- sort на наборі даних з різними розподілами

2.3.2 Виявлення та емулювання алгоритмів сортування за зображеннями слідів їх виконання

Експерименти показують, що нейронні мережі здатні моделювати механізми, що лежать в основі простих алгоритмів, якщо їх сліди достатньо надаються як дані. Більше того, продемонстровано, що програми можна розглядати як зображення, що відображають просторові візерунки, так і послідовні інструкції на мові, специфічному для предметної області. Підхід перевірено на трьох типах дедалі складніших завданнях: виявлення, розпізнавання та емуляція.

Слід зазначити, що хоча три різні установки проблеми мають однаковий простір станів, вони концептуально відрізняються один від одного. Вони показують, як той самий об’єкт (простір станів алгоритму) демонструє характеристики з різним ступенем складності: статичні (виявлення), часові (розпізнавання в послідовності) і генеративні (емуляція) середовищ.

Структура та експерименти, представлені для емуляції зображень слідів виконання, дають результати, які в деяких випадках є більш точними, ніж отримані за допомогою більш складних моделей машинного навчання, що призводить до висновку, що спостереження за поведінкою алгоритму вздовж шкали часу виконання може подолати бар’єри в індукції програми які залишаються складними навіть із дуже складними моделями машинного навчання. Однак ці моделі показують додаткові методи, які можна використовувати для покращення результатів підходу трасування виконання. Тому у звіті акцентується увага на результатах у напрямку постановки проблеми. Наші висновки показують, що ефективні моделі нейронних мереж для цієї області повинні підкорятися принципам формулювання проблеми (введення-виведення, трасування виконання), конкретних концепцій вирішення проблем

(наприклад, вказівники, зв’язки, увага, пам’ять, операції) і дизайну (послідовний, моделі з подачею вперед). Впровадження цих принципів у модель машинного навчання може потім узагальнити здатність вивчати програми з більш зашумленими або складнішиими слідами виконання [16].

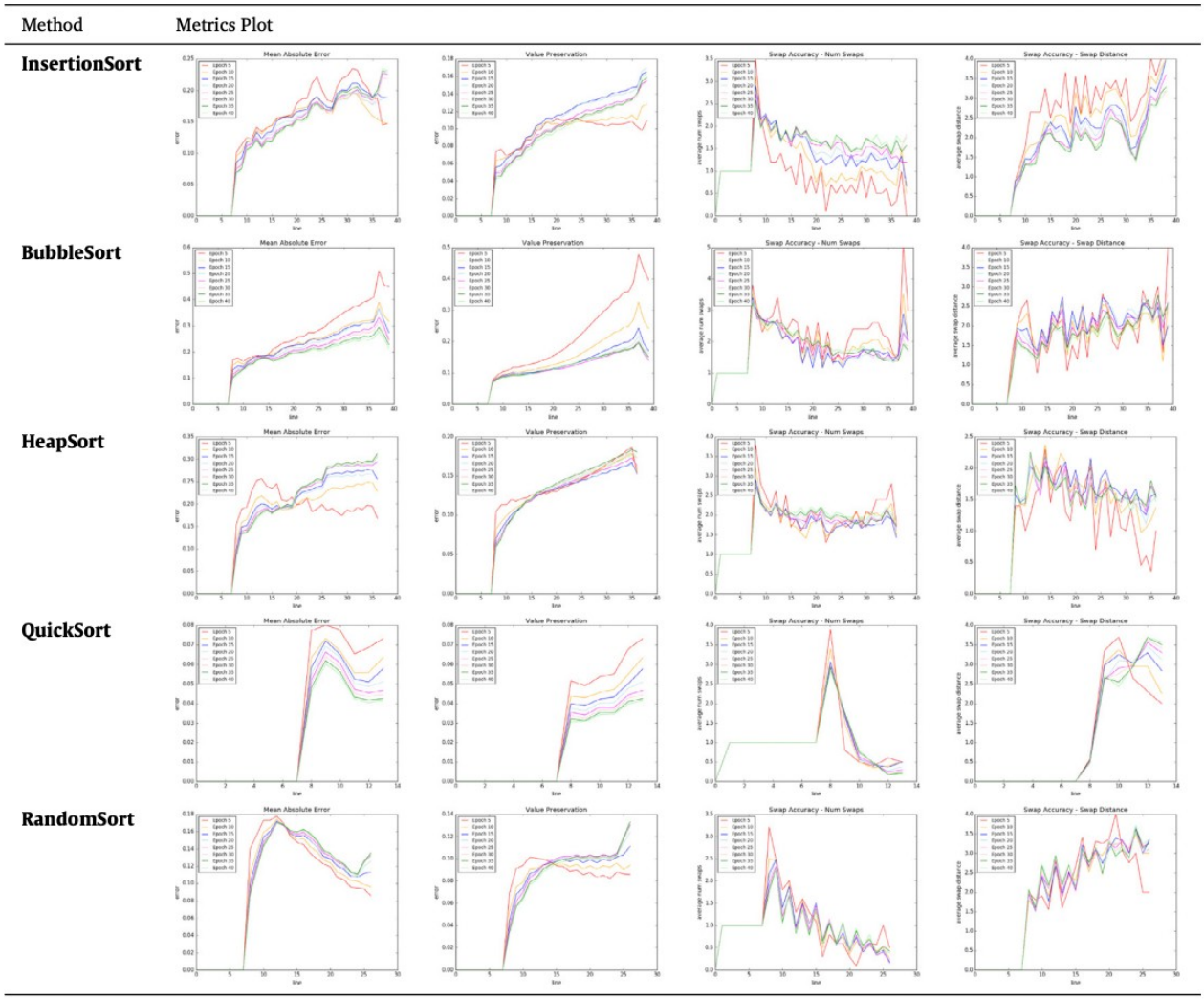


Рисунок 2.7. Показники помилок для емуляції за кількістю епох, навчених для SAE.

2.3.3 Нова штучна нейронна мережа для сортування

Було запропоновано нову техніку сортування на основі штучної нейронної мережі, яка здатна сортувати задану послідовність реальних елементів у монотонному (за спаданням чи зростанням) порядку. Хоча використовувана штучна нейронна мережа походить від штучної нейронної мережі теорії гармонії з точки зору конструктивних характеристик, її режим роботи було змінено в таких аспектах: ні функція гармонійного консенсусу, ні імітований відпал не використовуються, функції активації спрощені, виконується детерміноване оновлення активації, відбувається поступова дезактивація активованих вузлів і забезпечується автоматичний критерій завершення операції. Ці модифікації роблять запропоновану техніку простою в конструкції, а також прозорою та швидкою в роботі [19].

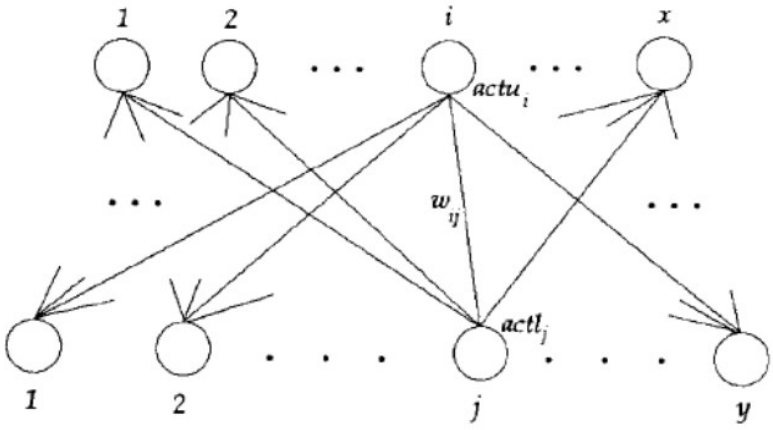


Рисунок 2.8. Загальна структура HT ANN

Висновки

Моделі sequence-to-sequence — це моделі глибокого навчання, які досягли значного успіху в таких завданнях, як машинний переклад, підсумовування тексту та створення підписів до зображень. Модель складається з кодера і

декодера. Кодер фіксує контекст вхідної послідовності у вигляді прихованого вектора стану та надсилає його в декодер, який потім створює вихідну послідовність.

Мережі вказівників (Pointer network) є різновидом моделі послідовності до послідовності з увагою. Замість перекладу однієї послідовності в іншу вони видають послідовність покажчиків на елементи вхідної серії. Основне використання цього — упорядкування елементів послідовності або набору змінної довжини.

Чимало досліджень було проведено з метою аналізу здатності нейронних мереж до якісного сортування даних. Наприклад, вчені у дослідженні «Сортування з урахуванням розподілу даних на основі нейронної мережі» представили NN-sort, метод сортування на основі нейронної мережі та розподілу даних. NN-sort використовує для сортування модель, навчену на історичних даних для сортування майбутніх даних.

РОЗДІЛ 3 РОЗРОБКА МОДЕЛЕЙ ДЛЯ СОРТУВАННЯ ЧИСЛОВИХ ДАНИХ

Для дослідження поставленої задачі було проведено програмну реалізацію розглянутих вище моделей, а також наведено модифікацію моделі вказівників. Зібрано значення метрик для порівняння оцінок моделей. Налаштування параметрів моделей для покращення результатів.

3.1 Модифікація моделі вказівників

Оригінальна архітектура моделі вказівників передбачає побудову кодеру-декодеру на основі LSTM. Довга короткочасна пам’ять у короткому LSTM – це особливий вид RNN, здатний вивчати довгострокові послідовності. Вони були введені Шмідхубером і Хохрайтером у 1997 році. Вони спеціально розроблені для уникнення довгострокових проблем залежності. Запам'ятовування довгих послідовностей протягом тривалого періоду часу - це його спосіб роботи. Ці комірки використовують ворота для регулювання інформації, яку потрібно зберігати або відкидати під час роботи циклу перед передачею довгострокової та короткострокової інформації до наступної комірки. Загалом існує три шлюзи, які LSTM використовує: вхідні шлюзи, шлюзи забуття та вихідні шлюзи.

Рідше за LSTM використовується GRU. Робочий процес Gated Recurrent Unit, коротко GRU, такий самий, як і RNN, але різниця полягає в роботі та воротах, пов’язаних з кожним підрозділом GRU. Щоб вирішити проблему, з якою стикається стандартний RNN, GRU включає два механізми роботи шлюзу, які називаються шлюзом оновлення та шлюзом скидання. Спочатку вступає в дію шлюз скидання, він зберігає відповідну інформацію з минулого кроку в новому вмісті пам’яті. Потім він множить вхідний вектор і прихований стан на їхні ваги. Далі він обчислює поелементне множення між воротами скидання та

попередньо прихованим кратним станом. Після підсумовування вищезазначених кроків застосовується нелінійна функція активації та генерується наступна послідовність.

Основна відмінність GRU полягає у тому, що вони не мають внутрішньої пам'яті, вони не мають вихідного шлюза, який присутній у LSTM. GRU використовує менше параметрів навчання і, отже, використовує менше пам'яті та виконується швидше, ніж LSTM. Тож, ми вирішили модифікувати модель, застосувуючи GRU для реалізації кодера-декодера.

Також була проведена зміна механізму уваги в декодері з метою досягнення кращих результатів. В оригінальній версій моделі вказівників використовується адитивна увага. Адитивна увага реалізується наступним чином

$$u_{\mathbf{j}}^{"} = v^T \tanh\#W_1 e_j + W_2 d_i\$, \ j \in (1, \ldots, \mathrm{n}) \tag{3.1}$$

де e_8 та d_{i} приховані стани кодера та декодера, а v, $W_\$$, W_2 є параметрами моделі, які можна вивчати.

На противагу адитивній увазі, мультиплікативна увага працює швидше. Адитивна та мультиплікативна увага схожі за складністю, хоча на практиці мультиплікативна увага є швидшою та більш ефективною, оскільки її можна реалізувати ефективніше за допомогою множення матриці. Обчислення мультиплікативної уваги відбувається наступним чином.

$$u_{\mathbf{j}}^{"} = v^T \tanh\#W[e_j, d_{\ i}]\$, \ j \in (1, \ldots, \mathrm{n}) \tag{3.2}$$

3.2 Структура програмного забезпечення

Для реалізації було обрано мову програмування Python, яка найчастіше використовується для задач машинного навчання та містить чимало бібліотек для роботи з нейронними мережами. В нашій роботі використовується бібліотека pytorch для реалізації моделей, numpy для різних обчислень, time для розрахунку часу сортування та matplotlib для графічного представлення певних результатів.

Програмний продукт складається з кількох модулів. Encoder.py містить стандартну реалізацію кодера. Encoder_gru.py містить модифіковану реалізацію кодера. Модулі ptr_decoder.py та attn_decoder.py містять реалізацію декодерів для моделей Pointer network та Sequence-To-Sequence відповідно. Для генерації навчальних та тестових даних було написано скрипт data.py. Для навчання та передбачення розроблено модулі train.py та evaluate.py. Також наведено допоміжні модулі: функції для візуалізації – visual.py, функції для обчислення метрик – metrics.py, інші допоміжні функції – utils.py. Для легшого проведення експериментів, реалізовано модулі з необхідними налаштуваннями, зібрані в директорію experiments.

3.3 Навчання та оцінка

Згенеровані дані для навчання моделей містять одновимірні масиви чисел різної довжини. Мінімальна довжина масива береться за 2, а максимальна – 100. В рамках даного експерименту розглядаються цілі числа. Тестові дані для оцінки моделей згенеровані аналогічним чином. Експериментальним шляхом було досліджено якої кількості даних вистачить моделям для навчання та адекватного сортування. Таким чином виявили, що для вирішення задачі достатньо взяти 10000 навчальних екземплярів. За меншої кількості спостерігається гірша робота сортування.

Під час навчання на кожній ітерації обчислюється функція втрат негативний логарифм правдоподібності втрати (NLLLOSS), приклад значень якої для однієї епохи зображено на рисунку 3.1 та рисунку 3.2.

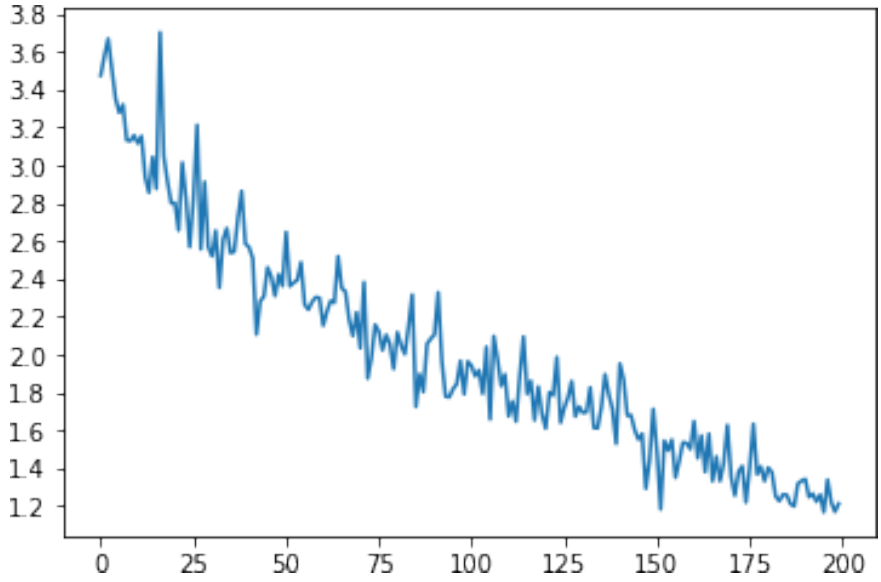


Рисунок 3.1. Значення NLLLOSS для першої епохи для моделі Pointer Network

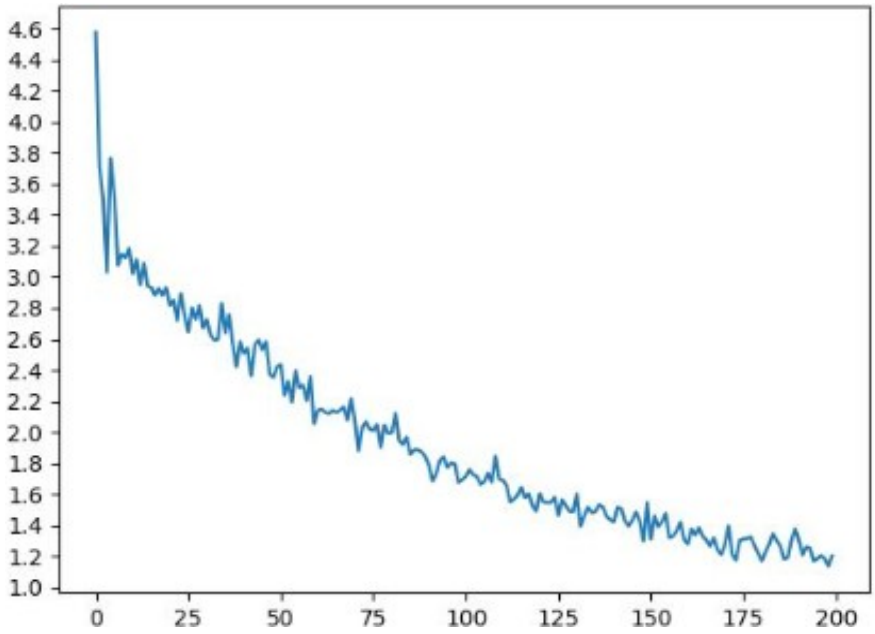


Рисунок 3.2. Значення NLLLOSS для першої епохи для моделі Seq2Seq

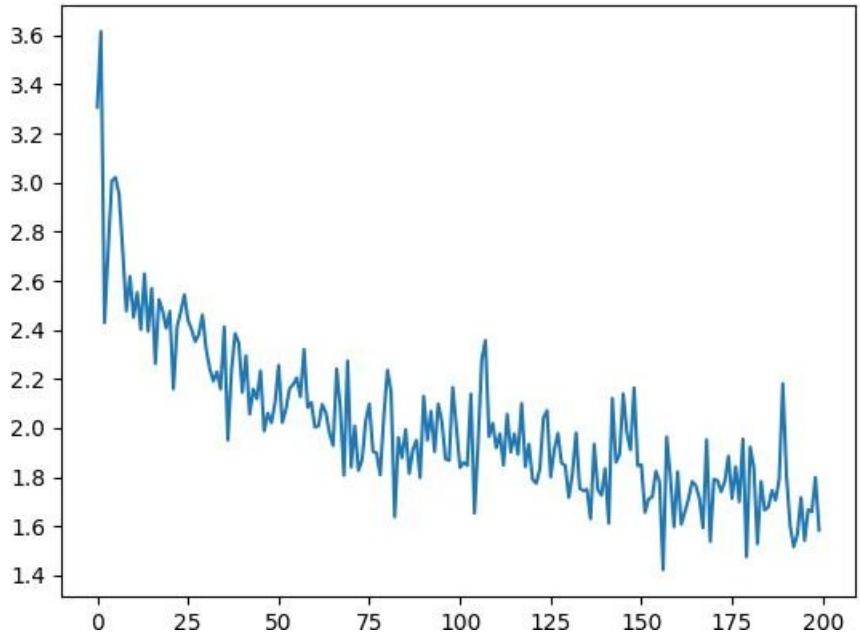


Рисунок 3.3. Значення NLLLOSS для першої епохи для модифікованої моделі Pointer Networks

Результати тестування роботи моделей наведені у таблиці нижче (Табл. 3.1).

Таблиця 3.1. Робота сортування тестових даних моделей Sequence-To-Sequence та Pointer Networks

Оригінальний масив	Sequence-To-Sequence	Pointer Networks	Modified Pointer Networks
[6, 23]	[6, 23]	[6, 23]	[6, 23]
[24, 7, 19, 28]	[7, 19, 24, 28]	[7, 19, 24, 28]	[7, 19, 24, 28]
[22, 12, 12, 29, 5, 1, 24]	[5, 12, 22, 24, 29]	[1, 5, 12, 22, 24, 29]	[1, 5, 12, 22, 24, 29]
[8, 1, 10, 22, 19, 17, 6, 11]	[6, 8, 10, 11, 17, 19, 22]	[1, 6, 8, 10, 11, 17, 19, 22]	[1, 6, 8, 10, 11, 17, 19, 22]
[18, 11, 30, 15, 26, 11, 23, 4]	[4, 11, 11, 15, 18, 23, 26, 30]	[4, 11, 15, 18, 23, 26, 30]	[4, 11, 11, 15, 18, 23, 26, 30]

[24, 2, 13, 21, 5, 6, 1]	[2, 5, 6, 13, 21, 24]	[2, 5, 6, 13, 21, 24]	[1, 2, 5, 6, 13, 21, 24]
[25, 17, 5, 1, 8, 18, 20, 18]	[5, 8, 17, 18, 20, 25]	[1, 5, 8, 17, 18, 18, 20, 25]	[1, 5, 8, 17, 18, 18, 20, 25]

Як видно з таблиці, модель періодично губить повторюванні значення (рядки 3 та 7 таблиці 3.1) та найменше значення 1. Тим не менш, немає порушення порядку сортування. Ми вирішили зробити модифікацію оригінального механізму уваги в моделі Pointer Networks, запропонованого авторами, з метою покращення результатів роботи. В таблиці 2 ми наводимо результати метрик, які дозволяють порівняти моделі. Метрика Permutation повертає долю, перестановок. Метрика Nondecreasing повертає долю елементів розташованих у порядку неспадання. Та, нарешті, метрика Accuracy повертає відсоток правильно відсортованих масивів.

Таблиця 3.2. Значення метрик Permutation, Nondecreasing та Accuracy

Метод / Метрика	Sequence-To-Sequence	Pointer Networks	Modified Pointer Networks
Permutation	0.875	0.901	0.949
Nondecreasing	0.192	0.034	0.011
Accuracy	88.53%	92.81%	95.48%

Оскільки однією з причин проведення даного дослідження було бажання мінімізувати час сортування даних великої розмірності, ми провели порівняння роботи моделі Pointer Network з класичними алгоритмами сортування. Результати часу сортування для масиву довжиною 1000000 елементів наведені у таблиці нижче (Табл. 3.3).

Таблиця 3.3. Порівняння часу сортування

Алгоритм	Modified Pointer Network	Pointer Network	Bubble Sort	Quick Sort	Merge Sort	Heap sort	IntroSort	TimSort
Час	21 s	25 s	1 h 58 m	1 s	2.5 s	5 s	3.4 s	3.2 s

Висновки

Оскільки сортування чисел по суті відображає послідовність кластерів у послідовність такої самої розмірності, ми вибрали для дослідження моделі Sequence-To-Sequence та Pointer Networks. Перевагою моделей є здатність сприймати на вхід послідовності нефіксованої розмірності, саме тому їх використовують для проблеми перекладу.

Було проведено модифікацію оригінальної моделі вказівників з метою покращення продуктивності. Для цього було використано GRU для реалізації кодера моделі та змінено оригінальний механізм адитивної уваги на мультиплікативну. Ці дії дозволили покращити результати, на що вказують запропоновані метрики.

Для дослідження було використано в якості навчальних даних списки цілочисельних значень різної розмірності. В результаті оцінки роботи моделей було виявлено, що вони виконують сортування чисел, проте періодично допускають похибку, зменшуючи довжину послідовності. Запропоновані метрики показали, що модель Modified Pointer Networks краще впоралась з цією задачею.

Планується проводити дослідження і надалі, не зупиняючись на цьому результаті. Найактуальніший бік розвитку – це мінімізація часу сортування моделі.

РОЗДІЛ 4 РОЗРОБКА ВЛАСНОГО СТАРТАП ПРОЕКТУ

Зараз у сучасному світі існує велика потреба в системах здатних оптимально сортувати великі об’єми даних. Існують класичні алгоритми, які виконують сортування числових даних, проте вони досягають своїх меж. Через це з’явилась ідея залучення глибокого навчання для вирішення даної задачі.

Отже, проблема, яку вирішує стартап, є актуальною і, потенційно, може стати успішним на ринку та бути основним вибором для переважної кількості цільової аудиторії.

4.1 План розробки стартапу та масштабування його на ринок

Наведемо план розробки стартапу та виведення його на ринок.

Спочатку треба провести маркетинговий аналіз, який включає в себе:

- конкурентний аналіз, щоб зрозуміти, якими методами вирішення проблем вже користуються люди;
- формування ідеї самого проекту та виділення цільової аудиторії;
- розробити стратегію виведення товару на ринок, базуючись на аналізі ринкового середовища.

Наступним кроком являється організація самого стартапу. На цьому етапі мають бути:

- складений весь план та побудований таймлайн розробки та запуску продукту;
- запланований обсяг виробництва та оцінений потенційний обсяг ресурсу, який буде потрібен для виконання плану;
- розраховані витрати, необхідні для реалізації проекту, та витрати на запуск проекту.

Далі необхідно виконати фінансово-економічний аналіз та оцінити ризики стартап-проекту, в межах якого:

- визначити обсяг інвестиційних втрат;
- розрахувати основні фінансово-економічні показники проекту (собівартість, ціну продукту/послуги, податковий збір та чистий прибуток) та визначити показники інвестиційної привабливості проекту (рентабельність продажів, період окупності проекту);
- визначити основні ризики проекту та способи для їх запобігання.

Фінальним кроком являється розробка заходів з комерціалізації продукту. Цей крок являється важливим для масштабування та збільшення розмірів продукту.

Для того, щоб залучити інвесторів та знайти різні способи фінансування проекту, необхідно:

- провести дослідження на предмет інтересів потенційних інвесторів та бізнесів;
- скласти інвестиційну пропозицію, яка включає в себе як опис самого продукту та його теперішні розміри, так і можливі шляхи розширення та розвитку;
- обрати канали комунікації із потенційно зацікавленими персонами.

Далі наведемо результати виконання кожного з описаних кроків.

4.2 Опис ідеї стартап-проекту

Стартап-проект полягає у вирішенні проблеми сортування числових даних методами глибокого навчання. Суть продукту стартапу полягає у тому, що система приймає на вхід одновимірні масиви різної довжини та повертає сортовану послідовність.

У таблиці 4.1 наведена інформаційна карта стартапу.

Таблиця 4.1 – Інформаційна карта стартап-проекту

Назва проекту	SortWithRNN
Автори проекту	Павлюк Віра
Коротка анотація	Додаток буде сортувати одновимірні масиви різної довжини, що містять цілі числа
Термін реалізації проекту	12 місяців
Необхідні ресурси	Приміщення з комп'ютерами, доступом до Інтернету, доступ до електромережі Програмне забезпечення для розробки, хмарне сховище для даних, антивірусні програми Фінансові кошти на оплату заробітної плати виконавцям на термін 12 місяців, а також на такі витрати як: оренда приміщення, комунальні послуги, оренда хмарного сховища тощо
Опис проблеми, яку вирішує проект	Продукт вирішує задачу сортування послідовності чисел нефіксованої довжини
Головні цілі та завдання проекту	Метою проєкту є створення моделі, яка буде сортувати дані великої розмірності
Очікувані результати	Привернення технологічних компаній до нашого стартапу, досягнення мети раціонального сортування даних

4.3 Технологічний аудит ідеї проекту

Тепер можна розібрати ідею стартапу та провести конкурентний аналіз. У таблиці 4.2 наведений опис ідеї стартапу.

Таблиця 4.2 – Опис ідеї стартапу

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Основна ідея полягає у створенні системи, яка буде приймати на вхід послідовність чисел нефіксованої довжини та повертатиме сортовану послідовність	Сортування великих об’ємів числових даних	Користувач зможе отримувати сортовані дані
	Гібридна система сортування для подальшої мінімізації часу роботи системи	Система буде оновлюватись і якість сортування числових даних буде покращуватись

Далі проведемо порівняльний аналіз конкурентів проекту та наведемо результати у таблиці 4.3.

Таблиця 4.3 – Порівняльний аналіз конкурентів проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W	N	S
		Власний проект	NN- sort	HT ANN			
1	Якість сортування даних	Надає доволі хороший результат	Свій власний алгоритм	Свій власний алгоритм			+
2	Доступність по ціні	Безкоштовний	Немає у вільному доступі	Немає у вільному доступі		+	
3	Персоналізація генерації	Присутня	Відсутній	Відсутнія		+	

Далі аналізуємо реальність технічно здійснити ідею проекту (таблиця 4.4).

Таблиця 4.4 – Технологічна здійсненність продукту

№ п/п	Ідея проекту	Технології і реалізації	Наявність технологій	Доступність технологій
1	Створення комплексної системи, яка буде приймати на вхід послідовність нефіксованої довжини, що містить цілі числа та повертатиме сортовану послідовність.	Використання мови програмування Python	Наявні	Доступні

Кінець таблиці 4.4

№ п/п	Ідея проекту	Технології і реалізації	Наявність технологій	Доступність технологій
2		Використання мови програмування C++	Не наявні, необхідні допрацювання	Доступні
3		Використання мови програмування Rust	Наявні, необхідні допрацювання	Доступні
Обрана технологія реалізації ідеї проекту: Python				

4.4 Аналіз ринкових можливостей запуску стартап-проекту

Далі проведемо попередній аналіз ринку для запуску стартап-проекту (таблиця 4.5).

Таблиця 4.5 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники ринку (найменування)	Характеристика
1	Кількість головних гравців, од	2
2	Загальний обсяг продаж, грн/ум.од	5000
3	Динаміка ринку (якісна оцінка)	Нестабільна, мінлива
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні

Кінець таблиці 4.5.

№ п/п	Показники ринку (найменування)	Характеристика
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	8%

Тепер проведемо характеристику потенційних клієнтів, які можуть бути зацікавлені в проекті (таблиця 4.6).

Таблиця 4.6 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреби, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Зменшення часу сортування	Компанії, що оперують великими об’ємами даних	Цікавить сортування даних за мінімальний час	Швидке сортування
2	Вирішення задачі сортування методами нейронних мереж	Розробники в сфері машинного навчання	Вирішення відомої задачі не стандартними методами	Новий напрямок дослідження

Обрахуємо фактори загроз (таблиця 4.7) та можливостей (таблиця 4.8). Проаналізуємо загрози, щоб зрозуміти можливі перешкоди при запуску продукт

на ринок. Фактори можливостей же треба обрахувати, щоб знати усі сприятливі умови та по можливості ними скористатися.

Таблиця 4.7 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	Хоча ринок є відкритим і неосвоїним, на ньому вже є кілька великих гравців, які відомі на ринку, які вже мають свою цільову групу покупців	Знайти точки додаткової цінності для користувача
2	Ціна збуту	Конкуренти можуть коштувати менше через нижчу якість	Сфокусуватися на якості роботи застосунку та продумати маркетингову стратегію
3	Якість аналізу	Оскільки задача вже вирішена стандартними методами, моделі можуть програвати класичним алгоритмам через свою точність та складність.	Мати достатній штаб і ресурси, для побудови різних моделей та покращення характеристик.

Таблиця 4.8 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Універсальнісь	Продукт не залежить від апаратної платформи як у більшості конкурентів	Зробити акцент під час маркетингу, продовжувати розвиток як окремого продукту
2	Простота у використанні	Від користувача треба лише завантажити файл з масивами числових даних	Реалізувати зручний інтерфейс для завантаження
3	Якість та гарантії	Надавати найбільш якісні послуги та сервіси	Пропонувати моделі з найкращими результатами, а також надавати усю необхідну технічну підтримку
4	Безкоштовний сервіс при MVP	Максимально швидко набрати базу своїх клієнтів та заявити про себе на ринку	Розгорнути широкий маркетинг, а також активно боротися за клієнтів конкурентів

Далі розглянемо питання конкуренції, а саме визначимо її тип та рівень (таблиця 4.9).

Таблиця 4.9 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	У чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції: недосконала конкуренція	Представлено мало продуктів та експертів	Зробити максимальним збут застосунку
2. За рівнем конкурентної боротьби: міжнародний	Наявні проекти, розроблені та можуть бути доступні у всьому світі	Розширити цільову аудиторію, розробити інтерфейс на різних мовах
3. За галузевою ознакою: внутрішньогалузева	Можуть працювати з різними галузями	Покращити персоналізацію
4. Конкуренція за видами товарів: товарно-родова	Конкуренція з аналізами інших систем та експертів	Підтримувати та покращувати якість існуючих функцій
5. За характером конкурентних переваг: нецінова	Різні компанії пропонують різну якість	Розробляти якісніші алгоритми і моделі
6. За інтенсивністю: марочна	Вже представлені компанії із сильним брендом	Предметно створити комунікаційну стратегію для вибудови свого бренду

Далі необхідно виконаємо аналіз конкуренції за моделлю 5 сил конкуренції Майкла Портера (таблиця 4.10).

Таблиця 4.10 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти у галузі	Потенційні конкуренти	Постачальники	Клієнти	Товарозамінники
	Інші існуючі системи та продукти	Якість, ціни, кількість користувачів, капіталовкладення	Фактори сили постачальників	Контроль якості, порівняння цін	Сила бренду, якість, ціна, масштаби
Висновки	Конкуренція з невеликою інтенсивністю, а також підігрітий ринок	Можливості входження на ринок, нові потенційні конкуренти	Постачальники відсутні	Клієнти не диктують умови роботи на ринку	Товарозамінники відсутні

Маючи результати аналізу конкуренції (таблиця 4.10), характеристики ідеї стартап-проекту (таблиця 4.5), характеристики потенційних клієнтів і їх вимоги до продукту (таблиця 4.6) та фактори ринкового середовища (таблиці 4.7 і 4.8) було сформульовано та обґрунтовано перелік факторів конкурентоспроможності (таблиця 4.11).

Таблиця 4.11 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Універсальність	Продукт не залежить від апаратної платформи як у більшості конкурентів
2	Простота у використанні	Від користувача треба лише завантажити файл зображення
3	Якість та гарантії	Надавати найбільш якісні послуги та сервіси
4	Безкоштовний сервіс при MVP	Максимально швидко набрати базу своїх клієнтів та заявити про себе на ринку

Тепер можна провести аналіз сильних та слабких сторін продукту (таблиця 4.12).

Таблиця 4.12 – Порівняльний аналіз сильних та слабких сторін системи

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів						
			-3	-2	-1	0	1	2	3
1	Універсальність	20	+						
2	Простота у використанні	16			+				
3	Якість та гарантії	10		+					
4	Безкоштовний сервіс при MVP	17			+				

Далі проведемо SWOT-аналіз продукту (таблиця 4.13).

Таблиця 4.13 – SWOT-аналіз стартап-проекту

Сильні сторони Універсальність Простота у використанні Якість та гарантії Безкоштовний сервіс при MVP	Слабкі сторони Відсутність сильного бренду Не сформована база клієнтів Не підключені альтернативні канали маркетингу
Можливості Покращення системи Мінімізація часових витрат Розширення спектру вхідних данних	Загрози Нові системи та експерти Збут Зовнішні обставини

Завдяки SWOT-аналізу, ми змогли визначити сильні та слабкі сторони, можливості та загрози, пов'язані з конкуренцією, плануванням стартап-проекту. Далі спроєктуємо альтернативну ринкову поведінку для інтеграції стартап-проекту на ринок та приблизний час реалізації системного комплексу, з урахуванням потенційних проектів, що можуть бути виведені на ринок та наведемо результати у таблиці 4.14.

Таблиця 4.14 – Альтернативи ринкового впровадження стартап проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Вихід на ринок з нижче якістю	50%	5 місяці
2	Пропонувати одразу платне використання	20%	7 місяців
3	Представлення користувачам системи без інтерфейсу	70%	8 місяці

У даному пункті був проведений детальний аналіз ринку та продукту. Також відповідно до результатів проведеного конкурентного аналізу, визначених факторів ринку та його сприятливість, описання ідеї та характеристик стартап-проекту, робимо висновок висновок, що існують дуже сприятливі умови для виходу продукту на ринок.

4.5 Розроблення ринкової стратегії стартап-проекту

Для розробки ринкової стратегії продукту, у першу чергу, необхідно проаналізувати цільову аудиторію проекту (таблиця 4.15).

Таблиця 4.15 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит умежах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу усегмент
1	Персональні користувачі	Низька	10%	Середня	Середня
2	Великі бізнеси	Висока	40%	Середня	Середня
3	Малі та середні бізнеси	Висока	40%	Середня	Середня
4	Держава	Низька	10%	Низька	Низька
Які цільові групи обрано: 2, 3					

Маючи аналіз цільових груп, далі визначимо базову стратегію розвитку продукту (таблиця 4.16).

Таблиця 4.16 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	2 та 3	Диференційованого маркетингу	Масштабування та максимізація	Оптимальних витрат

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (таблиці 4.17, 4.18).

Таблиця 4.17 – Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих уконкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
Ні	Так	Ні	Виклику лідера

Таблиця 4.18 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту (три ключових)
Універсальність Простота у використанні Якість результатів	Оптимальних витрат	Універсальність Простота у використанні Якість та гарантії Безкоштовне використання при MVP	Система, яка оптимальніше за можливі альтернативи сортує великі масштаби даних різної розмірності. Система з простим інтерфейсом

4.6 Розроблення маркетингової програми стартап-проекту

Після проведеного комплексного аналізу, можемо повноцінно описати ключові переваги концепції потенційного товару (таблиця 4.19) та побудувати концепцію маркетингових комунікацій (таблиця 4.20).

Таблиця 4.19 – Ключові переваги концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Якісне сортування	Якісне сортування даних	Постійне покращення та перенавчання моделей, які зможуть краще вирішувати задачу
2	Універсальність	Система не залежить від довжини вхідної послідовності	Така система може використовуватись для різної розмірності даних
3	Простий інтерфейс	Система дуже проста у використанні	Система із інтуїтивно зрозумілим інтерфейсом, який вимагає всього лише завантаження файлу з даними

Таблиця 4.20 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Пошук спеціалізованих систем	Публікація в спеціалізованих виданнях, журналах	Точність Якість Універсальність	Поєднати повідомлення про те, що це якісна система, яка є незалежною	Таргетована реклама на цільову аудиторію
2	Пошук доступного та дешевого продукту	Згадки в інтернет статтях, форуми	Простота Безкоштовне використання MVP	Вселити довіру у бренд та продукт	Реклама у лідерів думок Таргетована реклама на цільову аудиторію

4.7 Висновки

Даний розділ був присвячений дослідженню стартап-проекту. В якості такого була представлена система для сортування даних.

У рамках розділу було досліджено розробку стратегій виходу на ринок та маркетинг-стратегії для цього. Зокрема, даний ринок являється сприятливим з невеликою кількістю представлених компаній конкурентів. Оскільки вони дають лише частину функцій, а запропонована система є універсальною та доступною, то у стартап-проекту є всі шанси стати монополістами на ринку.

Також були опрацьовані сильні та слабкі сторони проекту, SWOT аналіз, аналіз конкурентів та цільової аудиторії. На основі всіх досліджень був сформований концепт маркетингової стратегії для обраних цільових аудиторій.

ВИСНОВКИ

Сортування є фундаментальною операцією в обчислювальній техніці. Проблема сортування не є новою та має багато різних рішень за допомогою класичних алгоритмів сортування, які базуються на послідовному попарному порівнянні та перестановці елементів. Проте класичні алгоритми не є оптимальними у випадку великої розмірності даних. Це може стати серйозною проблемою при розробці складних рішень.

Тим не менш нейронні мережі, завдяки своїй архітектурі, здатні виконувати паралельні обчислення, що дозволяє оптимізувати витрати часу та навантаження на процесор під час сортування.

У даній роботі було розглянуто використання нейронних мереж у задачі сортування числових даних. Для дослідження було обрано моделі послідовність-до-послідовності та мережу вказівників, через їх архітектуру. Ці моделі дозволяють подавати на вхід послідовності нефіксованої довжини. Модель вказівників, завдяки своїй архітектурі, вирішила проблему, коли розмір вихідного словника залежить від вхідних даних.

В роботі була запропонована модифікація моделі вказівників, запропонованої авторами у 2017 році. Було змінено механізм побудови кодеру-декодеру, що дозволило покращити продуктивність моделі. А також змінено оригінальний механізм уваги на користь мультиплікативної уваги, що покращило результати моделі. Запропоновані оцінки якості показали, що модифікована модель вказівників дає кращі результати за стандартні реалізації. Також модель краще показала себе за часом сортування, ніж оригінальна імплементація.

Планується проводити дослідження і надалі, не зупиняючись на цьому результаті. Найактуальніший бік розвитку – це мінімізація часу сортування моделі. Також є мета розширити словник вхідних даних та розглянути сортування багатовимірних масивів.

Практичне значення отриманих результатів полягає у використанні даної моделі підприємствами та поодинокиими розробниками у програмних забезпеченнях для сортування даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. S. Chen, G. J. Gibson, and C. F. N. Cowan, “Adaptive channel equalization using a polynomial perceptron structure,” Proc. Inst. Elect.Eng., vol. 137, pt. 1, pp. 257–264, Oct. 1990.
2. S. Chen, G. J. Gibson, C. F. N. Cowan, and P. M. Grant, “Adaptive equalization of finite nonlinear channels using multilayer perceptrons, ”Signal Process., vol. 20, pp. 107–119, 1990.
3. Kohonen, T., [Associative memory: A system – theoretical approach], Springer-Verlag, Berlin Heidelberg (1978).
4. Kohonen, T., [Content – Addressable Memories], Springer-Verlag, Berlin Heidelberg (1980).
5. Foster, C., [Content addressable parallel processors], Litton Education Publishing Inc, New York (1976).
6. *Data Structure - Sorting Techniques*. (n.d.). Retrieved October 28, 2022, from https://www.tutorialspoint.com/data_structures_algorithms/sorting_algorithms.htm
7. Patel, H. (2020, March 10). *An Overview of QuickSort Algorithm*. towardsdatascience.com.
8. GeeksforGeeks. (2022, September 23). *Merge Sort Algorithm*. https://www.geeksforgeeks.org/merge-sort/?ref=lbp
9. Строчков, В. (2019, July 15). *Пирамидальная сортировка (HeapSort)*. Хабр. https://habr.com/ru/company/otus/blog/460087/
10. GeeksforGeeks. (2022a, September 22). *Heap Sort*. https://www.geeksforgeeks.org/heap-sort/
11. *Introsort*. (n.d.). Retrieved October 28, 2022, from https://aquarchitect.github.io/swift-algorithm-club/Introsort/

12. *Алгоритм интросортировки — обзор и реализация на C++*. (n.d.). Retrieved October 28, 2022, from https://www.techiedelight.com/ru/introsort-algorithm/
13. GeeksforGeeks. (2021, November 25). *IntroSort or Introspective sort*. https://www.geeksforgeeks.org/introsort-or-introspective-sort/
14. *Tim Sort - javatpoint*. (n.d.). www.javatpoint.com. Retrieved October 28, 2022, from https://www.javatpoint.com/tim-sort
15. *Timsort — the fastest sorting algorithm you've never heard of*. (2018, June 26). HackerNoon. https://hackernoon.com/timsort-the-fastest-sorting-algorithm-youve-never-heard-of-36b28417f399
16. https://doi.org/10.1016/j.infsof.2020.106350 Received 13 October 2019; Received in revised form 27 April 2020; Accepted 17 May 2020 Available online 24 May 2020
17. *What Is a Neural Network?* (2022, September 21). Investopedia. https://www.investopedia.com/terms/n/neuralnetwork.asp
18. Xiaoke Zhu*, Taining Cheng, Jing He, Shaowen Yao⋆, Wei Zhou⋆, Qi Zhang*, and Ling Liu. 2020. NN-sort: Neural Network based Data Distribution-aware Sorting. In Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY . ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/1122445.1122456
19. Tambouratzis, Tatiana. (1999). A novel artificial neural network for sorting. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society. 29. 271-5. 10.1109/3477.752799.
20. Cortesi, Aldo (27 April 2007). "Visualising Sorting Algorithms". Retrieved 16 March 2017.
21. *Bubble sort algorithm*. GeeksforGeeks. (2022, October 28). Retrieved October 28, 2022, from https://www.geeksforgeeks.org/bubble-sort/
22. Education, I. C. (2022, March 30). *Deep Learning*. https://www.ibm.com/cloud/learn/deep-learning

23. *CS 230 - Recurrent Neural Networks Cheatsheet*. (n.d.-c). https://stanford.edu/%7Eshervine/teaching/cs-230/cheatsheet-recurrent-neural-networks
24. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734, October 25-29, 2014, Doha, Qatar. c 2014 Association for Computational Linguistics
25. Sutskever, I., Vinyals, O., & V. Le, Q. (2014). *Sequence to sequence learning with Neural Networks*. Retrieved November 15, 2022, from https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf
26. Dugar, Pranay. "Attention — Seq2Seq Models." *Medium*, 24 Nov. 2019, towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263. Accessed 16 Nov. 2022.
27. Nash, Charlie, et al. "PolyGen: An Autoregressive Generative Model of 3D Meshes." *Arxiv*, 23 Feb. 2020, arxiv.org/pdf/2002.10880.pdf. Accessed 18 Nov. 2022.
28. Chen, Yen-Chun, and Mohit Bansal. "Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting." *Arxiv*, 28 May 2013, arxiv.org/pdf/2002.10880.pdf. Accessed 18 Nov. 2022.
29. Singh, Abhishek. "Understanding Pointer Networks." *Medium*, 3 Feb. 2021, towardsdatascience.com/understanding-pointer-networks-81fbbc1ddbc8. Accessed 14 Nov. 2022.
30. McGough, Mason. "Pointer Networks with Transformers." *Medium*, 23 June 2021, towardsdatascience.com/pointer-networks-with-transformers-1a01d83f7543. Accessed 22 Nov. 2022.

ДОДАТОК А. ЛІСТИНГ ПРОГРАМНОГО КОДУ

1. Файл train.py

```
import random
import time
import numpy as np
import torch
import torch.nn as nn

from .utils import device, SOS_token, EOS_token, time_since, save_checkpoint,
load_checkpoint, RANDOM_SEED
from .visual import show_plot


def train(encoder, decoder, optim, optim_params, weight_init, grad_clip, is_ptr,
training_pairs, n_epochs,
          teacher_force_ratio, print_every, plot_every, save_every, checkpoint_dir):
    np.random.seed(RANDOM_SEED), torch.manual_seed(RANDOM_SEED)
    encoder.train(), decoder.train()
    encoder_optim = optim(encoder.parameters(), **optim_params)
    decoder_optim = optim(decoder.parameters(), **optim_params)

    checkpoint = load_checkpoint(checkpoint_dir)
    if checkpoint:
        start_epoch = checkpoint["epoch"]
        first_iter = checkpoint["iter"]
        plot_losses = checkpoint["plot_losses"]
        print_loss_total = checkpoint["print_loss_total"]
        plot_loss_total = checkpoint["plot_loss_total"]
        encoder.load_state_dict(checkpoint["encoder"])
        decoder.load_state_dict(checkpoint["decoder"])
        encoder_optim.load_state_dict(checkpoint["encoder_optim"])
        decoder_optim.load_state_dict(checkpoint["decoder_optim"])
    else:
        start_epoch = 0
        first_iter = 0
        plot_losses = []
        print_loss_total = 0 # Reset every print_every
        plot_loss_total = 0 # Reset every plot_every
        encoder.apply(weight_init) # initialize weights
        decoder.apply(weight_init)  # initialize weights

    criterion = nn.NLLLoss()

    size, n_iters = len(training_pairs), n_epochs * len(training_pairs)
    current_iter = start_epoch * size + first_iter
    start = time.time()
```

```
    for epoch in range(start_epoch, n_epochs):
        np.random.shuffle(training_pairs)
        start_iter = first_iter if epoch == start_epoch else 0
        for i in range(start_iter, size):
            loss = train_step(training_pairs[i], encoder, decoder, encoder_optim,
decoder_optim, is_ptr, criterion,
                              teacher_force_ratio, grad_clip)
            print_loss_total += loss
            plot_loss_total += loss
            current_iter += 1

            if current_iter % print_every == 0:
                print_loss_avg, print_loss_total = print_loss_total / print_every, 0
                print('%s (epoch: %d iter: %d %d%%) %.4f' % (time_since(start,
current_iter / n_iters),
                                                             epoch, i + 1,
                                                             current_iter / n_iters *
100,
                                                             print_loss_avg))

            if current_iter % plot_every == 0:
                plot_loss_avg, plot_loss_total = plot_loss_total / plot_every, 0
                plot_losses.append(plot_loss_avg)

            if current_iter % save_every == 0:
                if i + 1 < size:
                    save_epoch = epoch
                    save_iter = i + 1
                else:
                    save_epoch = epoch + 1
                    save_iter = 0
                save_checkpoint({
                    "epoch": save_epoch,
                    "iter": save_iter,
                    "plot_losses": plot_losses,
                    "print_loss_total": print_loss_total,
                    "plot_loss_total": plot_loss_total,
                    "encoder": encoder.state_dict(),
                    "decoder": decoder.state_dict(),
                    "encoder_optim": encoder_optim.state_dict(),
                    "decoder_optim": decoder_optim.state_dict(),
                }, checkpoint_dir)

        show_plot(plot_losses, save=True, name=checkpoint_dir + 'plot_'+ str(epoch))



def train_step(training_pair, encoder, decoder, encoder_optim, decoder_optim, is_ptr,
criterion, teacher_force_ratio,
               grad_clip):
    encoder_hidden = encoder.init_hidden()
    encoder_optim.zero_grad(), decoder_optim.zero_grad()

    loss = 0
```

```
    input_tensor, target_tensor = training_pair
    input_length, target_length = input_tensor.size(0), target_tensor.size(0)

    encoder_outputs = torch.zeros(input_length, encoder.hidden_dim, device=device)

    for i in range(input_length):
        encoder_output, encoder_hidden = encoder(input_tensor[i], encoder_hidden)
        encoder_outputs[i] = encoder_output[0, 0]

    decoder_input, decoder_hidden = torch.tensor([[SOS_token]], device=device),
encoder_hidden

    teacher_force = random.random() < teacher_force_ratio

    for i in range(target_length):
        args = (decoder_input, decoder_hidden, encoder_outputs)
        if is_ptr:
            args += (input_tensor,)
        decoder_output, decoder_hidden, _ = decoder(*args)
        if not teacher_force:
            topv, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze().detach()
        else:
            decoder_input = target_tensor[i]
        loss += criterion(decoder_output, target_tensor[i])

        if not teacher_force and decoder_input.item() == EOS_token:
            break

    loss.backward()
    nn.utils.clip_grad_norm_(encoder.parameters(), grad_clip),
nn.utils.clip_grad_norm(decoder.parameters(), grad_clip)

    encoder_optim.step()
    decoder_optim.step()

    return loss.item() / target_length
```

2. Файл evaluate.py

```
import torch
from .utils import device, SOS_token, EOS_token, MAX_LENGTH, RANDOM_SEED
import time

def evaluate(encoder, decoder, input_tensor, is_ptr, max_length=MAX_LENGTH):
    torch.manual_seed(RANDOM_SEED)
    encoder.eval(), decoder.eval()
```

```
    with torch.no_grad():
        input_length = input_tensor.size()[0]
        encoder_hidden = encoder.init_hidden()

        encoder_outputs = torch.zeros(input_length, encoder.hidden_dim, device=device)

        for i in range(input_length):
            encoder_output, encoder_hidden = encoder(input_tensor[i], encoder_hidden)
            encoder_outputs[i] += encoder_output[0, 0]

        decoder_input, decoder_hidden = torch.tensor([[SOS_token]], device=device),
encoder_hidden

        decoded_output = []
        start = time.time()
        for i in range(max_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(*(decoder_input,
decoder_hidden, encoder_outputs, input_tensor,))
            topv, topi = decoder_output.data.topk(1)
            if topi.item() == EOS_token:
                decoded_output.append('<EOS>')
                break
            else:
                decoded_output.append(topi.item())
            decoder_input = topi.squeeze().detach()
        total_time = time.time() - start
        return decoded_output, total_time
```

3. Файл generate.py

```
import os
import numpy as np
from .config.generate import get_config
from .utils import RANDOM_SEED


def generate(name, size, max_val, min_length, max_length, ewc):
    np.random.seed(RANDOM_SEED)
    if not os.path.isdir("data"):
        os.mkdir("data")
    with open("data/" + name + ".txt", mode='w') as file:
        file.write("|".join([str(size), str(max_val), str(max_length)]) + "\n")
        if ewc:
            for length in range(2, max_length + 1):
                for i in range(size):
                    if i % 10000 == 0:
                        print(length, i)
```

```
                    lst = list(np.random.randint(2, max_val, length))
                    srt = sorted(lst)
                    file.write(str(lst) + "|" + str(srt) + "\n")
        for i in range(size):
            if i % 10000 == 0:
                print(i)
            lst = list(np.random.randint(2, max_val, np.random.randint(min_length,
max_length)))
            srt = sorted(lst)
            file.write(str(lst) + "|" + str(srt) + "\n")


if __name__ == "__main__":
    args, unparsed = get_config()
    generate(args.name, args.size, args.max_val, args.min_length, args.max_length,
args.ewc)
```

4. Файл metrics.py

```
def is_permutation(a, b):
    if len(a) != len(b):
        return False
    member = [True for _ in range(len(a))]
    for item in a:
        found = False
        for i in range(len(b)):
            if item == b[i] and member[i]:
                member[i], found = False, True
                break
        if not found:
            return False
    return not any(member)


def nondecreasing(a):
    incorrect = 0
    for i in range(len(a) - 1):
        if a[i] > a[i+1]:
            incorrect += 1
    return incorrect


if __name__ == "__main__":
    from tests.metrics import run_tests
    run_tests()
```

5. Файл utils.py

```
import math
import os
import time
import re
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

SOS_token = 0
EOS_token = 1

MAX_LENGTH = 16
RANDOM_SEED = 0


def set_max_length(max_length):
    global MAX_LENGTH
    MAX_LENGTH = max_length


def as_minutes(s):
    h, m = math.floor(s / 3600), math.floor(s / 60)
    m, s = m - h * 60, s - m * 60
    return '%dh %dm %ds' % (h, m, s)


def time_since(since, percent):
    now = time.time()
    s = now - since
    es = s / percent
    rs = es - s
    return '%s (- %s)' % (as_minutes(s), as_minutes(rs))


def save_checkpoint(state, model):
    if not os.path.isdir('checkpoints'):
        os.mkdir('checkpoints')

    epoch, iteration = state["epoch"], state["iter"]
    root_dir = os.path.join("checkpoints", model)
    file_name = os.path.join(root_dir, "e" + str(epoch) + "i" + str(iteration) +
".ckpt")

    if not os.path.isdir(root_dir):
        os.mkdir(root_dir)
    torch.save(state, file_name)
```

```
def load_checkpoint(model):
    root_dir = os.path.join("checkpoints", model)
    if os.path.isdir(root_dir):
        max_epoch, max_iteration = -1, -1
        argmax_file = ""
        for file_name in os.listdir(root_dir):
            try:
                pattern = re.compile(r"""e(?P<epoch>[\d]*)
                                         i(?P<iter>[\d]*)
                                         \.ckpt""", re.VERBOSE)
                match = pattern.match(file_name)
                epoch, iteration = int(match.group("epoch")), int(match.group("iter"))
                if epoch > max_epoch and iteration > max_iteration:
                    max_epoch, max_iteration = epoch, iteration
                    argmax_file = file_name
            except (ValueError, AttributeError):
                print("A file other than a checkpoint appears to be in the checkpoints
folder; please remove it")
        if max_epoch >= 0 and max_iteration >= 0:
            print("Loading checkpoint file...")
            return torch.load(os.path.join(root_dir, argmax_file))
```

6. Файл visual.py

```
import os
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker


def show_plot(points, save=False, name="plot"):
    plt.figure()
    fig, ax = plt.subplots()
    loc = ticker.MultipleLocator(base=0.2)
    ax.yaxis.set_major_locator(loc)
    plt.plot(points)
    if save:
        if not os.path.isdir("imgs"):
            os.mkdir("imgs")
        plt.savefig("imgs/" + name + ".jpg")
    plt.show()
```

7. Файл encoder.py

```
import torch
import torch.nn as nn

from ..utils import device

class Encoder(nn.Module):
    def __init__(self, input_dim,
                 embedding_dim,
                 hidden_dim,
                 num_layers=1):
        super(Encoder, self)._init_()
        self.hidden_dim = hidden_dim

        self.embedding = nn.Embedding(num_embeddings=input_dim,
                                      embedding_dim=embedding_dim)
        self.gru = nn.GRU(input_size=embedding_dim,
                          hidden_size=hidden_dim,
                          num_layers=num_layers)
        # self.lstm = nn.LSTM(input_size=embedding_dim,
        #                     hidden_size=hidden_dim,
        #                     num_layers=num_layers)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output, hidden = self.gru(embedded, hidden)
        return output, hidden

    def init_hidden(self):
        return (torch.zeros(1, 1, self.hidden_dim, device=device), torch.zeros(1, 1,
self.hidden_dim, device=device))
```

8. Файл ptr_decoder.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from ..utils import device


class PtrDecoder(nn.Module):
    def   init  (self, output_dim,
```

```
                embedding_dim,
                hidden_dim,
                num_layers=1,
                dropout=0.1):
        super(PtrDecoder, self).__init__()
        self.output_dim = output_dim
        self.hidden_dim = hidden_dim

        self.embedding = nn.Embedding(num_embeddings=self.output_dim,
                                      embedding_dim=embedding_dim)
        self.dropout = nn.Dropout(dropout)
        self.gru = nn.GRU(input_size=embedding_dim,
                          hidden_size=self.hidden_dim,
                          num_layers=num_layers)

        self.attn = nn.Linear(in_features=self.hidden_dim * 2,
                              out_features=self.hidden_dim)
        self.out = nn.Linear(in_features=self.hidden_dim,
                             out_features=1)

    def forward(self, input, hidden, encoder_outputs, encoder_inputs):
        length = encoder_outputs.shape[0] # the length of the input sequence
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)
        output, hidden = self.gru(embedded, hidden)
        stacked_hidden = torch.squeeze(torch.stack([hidden[0] for _ in range(length)]))
        context = torch.cat((encoder_outputs, stacked_hidden), dim=1)
        attn_weights = F.log_softmax(self.out(torch.tanh(self.attn(context))), dim=0)

        output = torch.full((self.output_dim,), float("-inf"))
        for i in range(len(attn_weights)):
            output[encoder_inputs[i]] = attn_weights[i]

        return torch.unsqueeze(output, 0), hidden, attn_weights

    def init_hidden(self):
        return torch.zeros(1, 1, self.hidden_dim, device=device)
```

9. Файл attn_decoder.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from ..utils import device, MAX_LENGTH
class AttnDecoder(nn.Module):
    def   init  (self, output_dim,
```

```
                embedding_dim,
                hidden_dim,
                num_layers=1,
                dropout=0.1,
                max_length=MAX_LENGTH):
        super(AttnDecoder, self)._init_()
        self.output_dim = output_dim
        self.hidden_dim = hidden_dim
        self.max_length = max_length
        self.embedding = nn.Embedding(num_embeddings=self.output_dim,
                                      embedding_dim=embedding_dim)
        self.attn = nn.Linear(in_features=self.hidden_dim + embedding_dim,
                              out_features=self.max_length)
        self.attn_combine = nn.Linear(in_features=self.hidden_dim + embedding_dim,
                                      out_features=self.hidden_dim)
        self.dropout = nn.Dropout(dropout)
        self.lstm = nn.LSTM(input_size=self.hidden_dim,
                            hidden_size=self.hidden_dim,
                            num_layers=num_layers)
        self.out = nn.Linear(in_features=self.hidden_dim,
                             out_features=self.output_dim)

    def forward(self, input, hidden, encoder_outputs):
        encoder_outputs = F.pad(encoder_outputs, (0, 0, 0, self.max_length -
len(encoder_outputs)))
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)
        attn_weights = F.softmax(self.attn(torch.cat((embedded[0],
torch.squeeze(hidden[0], dim=0)), 1)), dim=1)
        attn_applied = torch.bmm(attn_weights.unsqueeze(0),
encoder_outputs.unsqueeze(0))
        attn_combine = self.attn_combine(torch.cat((embedded[0], attn_applied[0]),
1)).unsqueeze(0)
        output, hidden = self.lstm(F.relu(attn_combine), hidden)
        output = F.log_softmax(self.out(output[0]), dim=1)
        return output, hidden, attn_weights

    def init_hidden(self):
        return torch.zeros(1, 1, self.hidden_dim, device=device)
```

10. Файл experiments/train.py

```
import torch.nn as nn
from torch import optim

from ..data import read_data, tensors_from_pair
from ..models.attn_decoder import AttnDecoder
```

```
from ..models.encoder import Encoder
from ..models.ptr_decoder import PtrDecoder
from ..train import train
from ..utils import device, set_max_length


def weight_init(module):
    """
    Initialize weights of <module>. Applied recursivly over model weights via .apply()
    """
    for parameter in module.parameters():
        nn.init.uniform_(parameter, -0.08, 0.08)


def run(is_ptr=True, checkpoint_dir='ptr/vanilla', n_epochs = 1, grad_clip = 2, dim =
256):
    """
    Run the experiment.
    """
    name = "train"
    max_val, max_length, pairs = read_data(name)

    hidden_dim = embedding_dim = dim
    teacher_force_ratio = 0.5
    optimizer = optim.Adam
    optimizer_params = {}

    set_max_length(max_length)
    training_pairs = [tensors_from_pair(pair) for pair in pairs]

    data_dim = max_val + 1
    encoder = Encoder(input_dim=data_dim,
                      embedding_dim=embedding_dim,
                      hidden_dim=hidden_dim)
    if is_ptr:
        decoder = PtrDecoder(output_dim=data_dim,
                             embedding_dim=embedding_dim,
                             hidden_dim=hidden_dim)
    else:
        decoder = AttnDecoder(output_dim=data_dim,
                              embedding_dim=embedding_dim,
                              hidden_dim=hidden_dim)

    train(encoder=encoder,
          decoder=decoder,
          optim=optimizer,
          optim_params=optimizer_params,
          weight_init=weight_init,
          grad_clip=grad_clip,
          is_ptr=is_ptr,
          training_pairs=training_pairs,
          n_epochs=n_epochs,
          teacher_force_ratio=teacher_force_ratio,
          print_every=150,
```

```
          plot_every=50,
          save_every=100,
          checkpoint_dir=checkpoint_dir)
```

11. Файл experiments/evaluate.py

```
import numpy as np
import time

from ..data import read_data, tensors_from_pair
from ..evaluate import evaluate
from ..models.attn_decoder import AttnDecoder
from ..models.encoder import Encoder
from ..models.ptr_decoder import PtrDecoder
from ..utils import load_checkpoint, device, RANDOM_SEED, time_since, as_minutes


def run(is_ptr=True, checkpoint_dir='ptr/vanila', dim = 256, calc_time=False):
    np.random.seed(RANDOM_SEED)
    max_val, max_length, pairs = read_data(name="test")
    np.random.shuffle(pairs)
    training_pairs = [tensors_from_pair(pair) for pair in pairs]

    data_dim = max_val + 1
    hidden_dim = embedding_dim = dim

    encoder = Encoder(input_dim=data_dim,
                      embedding_dim=embedding_dim,
                      hidden_dim=hidden_dim).to(device)
    if is_ptr:
        decoder = PtrDecoder(output_dim=data_dim,
                             embedding_dim=embedding_dim,
                             hidden_dim=hidden_dim).to(device)
    else:
        decoder = AttnDecoder(output_dim=data_dim,
                              embedding_dim=embedding_dim,
                              hidden_dim=hidden_dim).to(device)

    checkpoint = load_checkpoint(checkpoint_dir)
    if checkpoint:
        encoder.load_state_dict(checkpoint["encoder"])
        decoder.load_state_dict(checkpoint["decoder"])
    else:
        print("Count not find checkpoint file.")

    len_pairs = len(pairs)
    if (calc_time):
```

```
        time_arr = []
        for i in range(len_pairs):
            input_tensor, target_tensor = training_pairs[i]
            output_tensor, time = evaluate(encoder=encoder,
                                           decoder=decoder,
                                           input_tensor=input_tensor,
                                           is_ptr=is_ptr)
            time_arr.append(time)
        print('Total time: %s' % (as_minutes(sum(time_arr))))

    else:
        for i in range(len_pairs):
            input_tensor, target_tensor = training_pairs[i]
            if (len(list(np.asarray(input_tensor.data).squeeze())) < 10):
                output_tensor, time = evaluate(encoder=encoder,
                                               decoder=decoder,
                                               input_tensor=input_tensor,
                                               is_ptr=is_ptr)
                print('Input: ', list(np.asarray(input_tensor.data).squeeze()))
                print('Output: ', output_tensor[:-1])
                print()
```

12. Файл experiments/report_metrics.py

```
import numpy as np

from ..data import read_data, tensors_from_pair
from ..evaluate import evaluate
from ..models.attn_decoder import AttnDecoder
from ..models.encoder import Encoder
from ..models.ptr_decoder import PtrDecoder
from ..utils import load_checkpoint, device, RANDOM_SEED
from ..metrics import is_permutation, nondecreasing


def run(is_ptr=True, checkpoint_dir='ptr/vanilla', dim=256):
    np.random.seed(RANDOM_SEED)
    max_val, max_length, pairs = read_data(name="test")
    np.random.shuffle(pairs)
    training_pairs = [tensors_from_pair(pair) for pair in pairs]

    data_dim = max_val + 1
    hidden_dim = embedding_dim = dim

    encoder = Encoder(input_dim=data_dim,
                      embedding_dim=embedding_dim,
                      hidden_dim=hidden_dim).to(device)
    if is_ptr:
```

```
        decoder = PtrDecoder(output_dim=data_dim,
                             embedding_dim=embedding_dim,
                             hidden_dim=hidden_dim).to(device)
    else:
        decoder = AttnDecoder(output_dim=data_dim,
                              embedding_dim=embedding_dim,
                              hidden_dim=hidden_dim).to(device)

    checkpoint = load_checkpoint(checkpoint_dir)
    if checkpoint:
        encoder.load_state_dict(checkpoint["encoder"])
        decoder.load_state_dict(checkpoint["decoder"])
    else:
        print("Count not find checkpoint file.")

    permutation_count, nondecreasing_count = 0, 0
    for i in range(len(training_pairs)):
        input_tensor, target_tensor = training_pairs[i]
        output_tensor = evaluate(encoder=encoder,
                                 decoder=decoder,
                                 input_tensor=training_pairs[i][0],
                                 is_ptr=is_ptr)
        target, output = list(np.asarray(input_tensor.data).squeeze()), output_tensor[:-
1]
        if is_permutation(target, output):
            permutation_count += 1
        if nondecreasing(output) == 0:
            nondecreasing_count += 1
    print("Permutation: %s" % (permutation_count / len(training_pairs)))
    print("Nondecreasing: %s" % (nondecreasing_count / len(training_pairs)))
```