НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»
на тему: «Метод передачі даних за допомогою нейронної мережі»

Виконала:
студентка IV курсу, групи КП-51
Граділь Анастасія Валеріївна _____

Керівник:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
Професор кафедри СПіСКС ФПМ, д.т.н.,
Романкевич В.О. _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____ 2018 р.

ЗАВДАННЯ

на дипломну роботу студенту

Граділь Анастасії Валеріївни

1. Тема роботи «Метод передачі даних за допомогою нейронної мережі», керівник роботи Онай Микола Володимирович, к.т.н., доцент, затверджені наказом по університету від «*22*» *травня* 2019 р. № *1331-С*

2. Термін подання студентом роботи «21» червня 2019 р.

3. Вихідні дані до роботи:
   - набір зображень для тестування рішень машинного зору MNIST;
   - набір сигналів ЕКГ;
   - тексти статей Вікіпедії.

4. Зміст роботи:
   - вивчити літературні джерела за тематикою дослідження, в тому числі специфіка роботи з текстовими даними;
   - провести аналіз методів передачі потокових даних;
   - створити та навчити автоенкодери для різних типів даних;
   - розробити клієнт-серверну програму для тестування методу;
   - оцінити швидкість передачі даних та навантаження мережі.

5. Перелік обов'язкового ілюстративного матеріалу:
   - схема роботи методу;
   - принцип роботи згорткових нейронних мереж;
   - особливості обробки тексту;
   - результати порівняння розробленого методу з існуючим.

6. Консультанти розділів роботи

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

7. Дата видачі завдання «31» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за тематикою роботи та збір даних	15.11.2018	
2.	Проведення порівняльного аналізу бібліотек для створення автоенкодерів	10.12.2018	
3.	Підготовка матеріалів першого розділу дипломної роботи	30.12.2018	
4.	Розроблення клієнт-серверної програми для тестування	01.02.2019	
5.	Підготовка матеріалів другого розділу дипломної роботи	20.02.2019	
6.	Створення та навчання автоенкодерів для різних типів та розмірів даних	01.03.2019	
7.	Підготовка матеріалів третього розділу дипломної роботи	30.03.2019	
8.	Перевірка розробленої моделі	20.04.2019	
9.	Підготовка матеріалів четвертого розділу дипломної роботи	20.05.2019	
10.	Підготовка графічної частини дипломної роботи	10.06.2019	
11.	Оформлення дипломної роботи	15.06.2019	

Студент А.В. Граділь

Керівник роботи М.В. Онай

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

MP3 – формат файлу для зберігання аудіо-інформації.

RGB – (скорочено від англ. Red, Green, Blue – червоний, зелений, синій) – адитивна колірна модель, що описує спосіб синтезу кольору, за якою червоне, зелене та синє світло накладаються разом, змішуючись у різноманітні кольори. Широко застосовується в техніці, що відтворює зображення за допомогою випромінення світла.

HSL – (скорочено від англ. Hue, Saturation, Lightness) – колірна модель, в якій будь-який колір визначається трьома характеристиками: кольоровим тоном (англ. Hue), наприклад, синім, червоним, жовтим тощо; насиченістю (англ. Saturation), тобто частиною чистого кольору, без домішки чорної та білої фарб; «світлотою» (англ. Lightness), тобто близькістю до білого кольору.

HSV – колірна модель, заснована на трьох характеристиках кольору: колірному тоні (Hue), насиченості (Saturation) і значенні кольору (Value), який також називають яскравістю (Brightness).

CMYK (скорочено від англ. Cyan, Magenta, Yellow, BlacK color) – субтрактивна колірна модель, використовується у поліграфії, перш за все при багатофарбовому (повноколірному) друці. Вона застосовується у друкарських машинах і кольорових принтерах.

ЕКГ – електрокардіографі́я – метод графічної реєстрації електричних явищ, які виникають у серцевому м'язі під час його діяльності, з поверхні тіла. Таким чином, ЕКГ – це запис коливань різниці потенціалів, які виникають у серці під час його збудження.

PCA – ме́тод головни́х компоне́нт (МГК, англ.principal component analysis) – метод факторного аналізу в статистиці, який використовує ортогональне перетворення множини спостережень з можливо пов'язаними змінними (сутностями, кожна з яких набуває різних числових значень) у множину змінних без лінійної кореляції, які називаються головними компонентами.

CLR – Common Language Runtime – «загальномовне виконуюче середовище» – це компонент пакету Microsoft.NET Framework, віртуальна машина, на якій виконуються всі мови платформи .NET Framework.

LLVM – Low Level Virtual Machine – універсальна система аналізу, трансформації і оптимізації програм, що реалізує віртуальну машину з RISC - подібними інструкціями. Може використовуватися як оптимізуючий компілятор цього байт-коду в машинний код для різних архітектур або для його інтерпретації та JIT-компіляції (для деяких платформ).

GNU GPL – GNU General Public License (Загальна публічна ліцензія GNU або Загальна громадська ліцензія GNU) – одна з найпопулярніших ліцензій на вільне програмне забезпечення.

GNU – вільна UNIX-подібна операційна система, що розробляється Проектом GNU.

JVM – віртуальна машина Java (англ. Java Virtual Machine) – набір комп'ютерних програм та структур даних, що використовують модель віртуальної машини для виконання інших комп'ютерних програм чи скриптів.

XML – розши́рювана мо́ва розмі́тки (англ.Extensible Markup Language) – запропонований консорціумом World Wide Web Consortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет.

API – прикладни́й програ́мний інтерфе́йс (інтерфейс програмування застосунків, інтерфейс прикладного програмування) (англ. Application Programming Interface) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено – це набір чітко визначених методів для взаємодії різних компонентів.

ML – машинне навчання (англ. machine learning) – це підгалузь штучного інтелекту в галузі інформатики, яка часто застосовує статистичні прийоми для надання комп'ютерам здатності «навчатися» (тобто, поступово

покращувати продуктивність у певній задачі) з даних, без того, щоби бути програмованими явно.

CNTK – Microsoft Cognitive Toolkit – це стандартизований інструментарій для проектування і розвитку мереж різноманітних видів, застосовує штучний інтелект для роботи з великими обсягами даних шляхом глибокого навчання, використовує внутрішню пам'ять для обробки послідовностей довільної довжини.

Архітектура штучних нейронних мереж – набір шарів різних типів та розмірів та типи їх з’єднання.

JSON – (англ. JavaScript Object Notation) – це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, який може бути прочитаний людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією).

ВСТУП

Актуальність інформації є дужа важливим критерієм успіху у нашому столітті. Вона необхідна не лише у побуті, але і для прийняття правильних важливих рішень на заводах з виробництва матеріалів, на фінансових біржах, на подприемствах з продажу товарів та ін. Інформації стає дедалі більше. Більшість передається по мережі Інтернет.

Наразі існує безліч мережевих протоколів, що допомогають керувати процесом передачі даних по мережі. Активно розвиваються методі шифрування та ущільнення даних. Також, дуже швидко розвивається область нейронних мереж. У даній області ледь не щодня відбуваються нові відкриття. Провідні університети світу лише з 2012 року (5 років назад) почали активно включати цю науку до своїх навчальних програм та готувати спеціалістів даної області.

Тому було прийнято рішення у даній роботі провести дослідження ефективності нейронних мереж для оптимізації процесу передачі даних. А саме досліджується швидкість передачі даних по мережі з попереднім ущільненням за допомогою автоенкодера та подальше відновлення початкових даних на іншому кінці мережі.

Планується розробити різні архітектури автоенкодерів для підтримки різних типів даних: тексту, музики, зображень та відео. А також порівняти такий підхід з роботою звичайних методів, що наразі використовуються у світі.

МЕТА ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

Метою дипломної роботи є розроблення методу передачі даних по мережі за допомогою нейронної мережі.

Науково-практична задача, що розв’язується у даній дипломній роботі, включає наступні завдання:

1. Аналіз існуючих методів передачі даних.
2. Аналіз особливостей обробки різних типів даних.
3. Розроблення методу передачі даних по мережі, що має менше навантаження завдяяки швидкому ущільненню даних за допомогою автоенкодера.
4. Створення та навчання штучних нейронних мереж для даних різних типів та розмірів.
5. Розроблення клієнт-серверної програми для тестування методу.
6. Оцінка навантаження мережі та швидкості передачі даних при використанні розробленого методу та порівняння отриманих результатів з відповідними показниками існуючого методу.

1. АНАЛІЗ ТЕХНОЛОГІЙ

1.1. Протокол потокової передачі даних

Потокова передача даних – технологія для передачі інформації малими порціями, що можуть відтворюватись окремо. Один комп'ютер, сервер, ділить інформацію на послідовні частини. Окремі частини ущільнюються та загортаються у пакети, які містять інформацію про адресата та порядковий номер пакету.Такі пакети передаються по мережі до комп'ютера-клієнта. Там вони розпаковуються, та відтворюються. Особливістю потокової передачі даних є те, що пакети формуються та надсилаються по мірі їх необхідності. Тобто клієнт сам вирішує з якою інтенсивністю будуть надсилатись пакети. Найчастіше використовується для передачі мультимедійної інформації (аудіо, відео) [2].

Окремим типом потокової передачі інформації є передача у реальному часі. Тобто пакети формуються та надсилаються як тільки з'являється інформація, яку слід передати. Очевидно, що передача у реальному часі має свої обмеження. Пакети не можуть передаватись швидше, ніж генеруються нові дані. Також, через поганий зв'зок у мережі може статись ситуація, що один з пакетів буде втрачено. Тоді він не буде відправлений знову. Натомість, буде відправлено наступний пакет з новими даними. Це спричинить відсутність цілісності даних на комп'ютері-клієнті [1].

Для регулювання потокової передачі даних було створено мережеві протоколи. Найпоширенішими протоколами прикладного рівня моделі TCP/IP є: RTSP, RTCP, RTP. Розглянемо їх детальніше.

RTP (Real-time Transport Protocol) – протокол передачі даних у реальному часі. Він ідентифікує тип та номер пакету, встановлює мітку синхронізації. В пакетах також зберігається інформація про відправника, сеанс та характер вмісту пакету (наприклад тип кодування). На основі даних, що передаються таким пакетом, одержувач синхронізує різні потоки

(наприклад, звук та відео) та послідовно відтворює отримані дані. Протокол працює на основі протоколу транспортного рівня UDP.

RTCP (RTP Control Protocol) – протокол, що призначений для раегування на зміни у мережі. Функцією протоколу є контроль якості послуг та перевірка навантаженості системи. Пакети даного протоколу є багатоадресними, тож сервер може отримувати відповіді відразу від декількох клієнтів. У відповідях може міститись інформація про різноманітні проблеми зв'язку, такі як втрата пакетів або нерівномірна швидкість передачі інформації. На основі отриманої інформації відбувається масштабування сеансу шляхом налаштування інтенсивності відправки пакетів.

RTSP (Real Time Streaming Protocol) – потоковий протокол реального часу. Даний протокол не ущільнює дані та не визначає методи інкапсуляції даних. Він не залежить від конкретних транспортних протоколів. Треба відмітити, що передача самих даних не є завданням даного протоколу. Натомість, для передачі даних часто використовуються RTP у поєднанні з RTCP. Такий підхід дозволяє контролювати швидкість передачі. Призначенням самого протоколу є встановлення зв'язку та управління сеансами[3]. Для своєї роботи протокол використовує набір команд:

- OPTIONS – запит методів, що підтримуються сервером;
- DESCRIBE – запит на отримання опису контенту;
- SETUP – встановлює медіа потік, передаючи url-адреси потоку та номери портів для прийому RTP та RTCP даних. Має бути виконаний перед командою PLAY;
- PLAY – запит на початок відтворення потоку. Якщо момент початку (мітка часу) не вказано, відтворення починається з початку, або з моменту останньої зупинки;
- PAUSE – запит на тимчасову зупинку потоку;
- RECORD – запит на збереження даних. Можна вказати час початку запису та час завершення або один з цих параметрів;

- ANNOUNCE – може бути відправлений від клієнта до сервера та навпаки. В обох випадках ініціює оновлення інформації про потік;
- TEARDOWN – запит на завершення сеансу;
- REDIRECT – використовується, щоб повідомити клієнта про необхідність підключитись до іншого сервера. Якщо клієнт бажає працювати з поточним сервером, йому слід надіслати запити TEARDOWN та SETUP до поточного сеансу;
- GET_PARAMETER – запит на отримання параметрів потоку. Іноді використовується для перевірки затримки передачі даних по мережі;
- SET_PARAMETER – запит на встановлення нових параметрів для поточного потоку.

Наведені вище протоколи прикладного рівня найчастіше працюють на основі протоколу UDP.

UDP (User Datagram Protocol) – протокол транспортного рівня стеку TCP/IP. Він відповідає за загортання повідомлень у пакети (датаграми) та їх відправку по мережі. Однак, даний протокол не дає гарантії доставки. Навіть у разі помилки або втрати пакету протокол вважає свою роботу виконаною. Тоді відповідальність за повторну відправку даних бере протокол вищого рівня. Такий підхід є ефективним при передачі даних у реальному часі. Адже при неякісному мережевому з'єднанні багаточисельні спроби повторної відправки пакету можуть спричинити кілька неприємних ситуацій. По-перше, додаткове навантаження на систему. По-друге, інші пакети, що містять актуальну інформацію очікують відправки у черзі. Тобто останні відправлені пакети не несуть актуальної інформації. Тож при втраті пакету, UDP просто відправить наступний пакет. Зі сторони клієнта, звісно, частина даних буде втрачена, але система залишиться у стабільному стані та подальша інформація буде відправлятись вчасно.

TCP (Transmission Control Protocol) – протокол транспортного рівня. На відміну від UDP, він бере на себе відповідальність за встановлення з'єднання та доставку повідомлень. Такий протокол доцільно

використовувати у сервісах доставки повідомлень, системах обліку та інших системах, де важливо, щоб інформація була доставлена правильно. При відправці першого повідомлення протокол ініціює встановлення з'єднання і лише потім надсилає повідомлення. Коли одна зі сторін (клієнт або сервер) розуміє, що повідомлень більше не буде, протокол відповідає за коректне розірвання з'єднання.

Методи відправки даних у реальному часі відрізняються не лише мережевими протоколами, але і схемами маршрутизації даних. Розглянемо також деякі з них.

Unicast (одноцільова передача) – схема, за якою один пакет створюється та надсилається для одного адресата. Навіть коли кілька адресатів отримують одну інформацію, процес створення пакетів відбуватиметься для кожного окремо.

Multicast (групова передача) – схема, при якій пакети даних відправляються відразу декільком адресатам. Використання цієї схеми є особливо ефективним при передачі даних у реальному часі, адже одночасно деяка кількість клієнтів отримують одні і ті самі дані. Саме тому доцільно спаковувати інформацію у пакети один раз та розсилати його усім, а не робити це окремо для кожного клієнта. Такий підхід економить і час і пам'ять сервера.

Broadcast (широкомовна передача) – використовує спеціальну IP-адресу, щоб надіслати один потік усім клієнтам. Такий потік буде прийматися всіма увімкненими пристроями мережі незалежно від їх бажання. Тож така схема частіше використовується для передачі службової інформації мережевого рівня.

1.2. Аналіз згорткових нейронніх мереж для різних типів даних

Згорткова нейронна мережа (Convolutional Neural Network) – архітектура штучних нейронних мереж, призначена для розпізнавання образів. На відміну від багатошарового персептрона, нейронні мережі даного типу врховують «топологію» інформації [4].

Відмінність згорткової мережі від звичайної легко пояснити на прикладі розпізнавання картинки. Персепторон отримує забраження, як одномірний масив пікселів. Пікселі, що йдуть один за одним по вертикалі розташовані поряд. А от пікселі праворуч та ліворуч знаходиться дуже далеко в такому масиві. При такій архітектурі штучної нейронної мережі складно розпізнати горизонтальну лінію.

Зовсім інша ситуація складається при розпізнаванні такого ж зображення згортковою нейронною мережею. Оскільки вона розглядає зображення, як двомірний масив (саме так, як бачить його людина). При передачі інформації з одного пікселя, враховується інформація усіх сусідніх. При чому кожен сусідній піксель має різну вагу. Одже, лінію будь-якого напрямку буде легко розпізнати.

Згорткові нейронні мережі застосовуються для різних типів задач. Найчастіше для класифікації та реграсії. Дані можуть мати кілька каналів інформації, що формально збільшує їх розмірність, але фактично розглядається як погляд на одне і те ж з різних боків. Для кожного типу вхідної інформації підбирається особлива архітектура.

Розглянемо одномірні згорткові мережі. Такий тип нейронних мереж застосовується для даних, що легко подаються у вігляді одномірних масивів. До таких даних можна віднести будь-які часові ряди з одним числовим значення у кожен момент часу. Наприклад, сигнали ЕКГ мають лише один канал – одне значення у момент часу, а от аудіо-записи мають мінімум два канали – частота звукової хвилі та її амлітуда (такі параметри використовуються, наприклад, для збереження аудіо-файлу у форматі MP3).

При проведенні аналізу таких даних для розпізнавання патернів використовуються одномірні ядра згортки – вектор з непорною кількістю елементів. Ядро (або декілька ядер) проходить по значенням кожного з каналів, слугуючи ніби вагами між частинкою нейроннів вхідного шару та одним нейроном наступного, та виділяє відповідні ознаки, такі як збільшення або зменшення значень, короткочасні піки, хвилеподібна зміна

значень та інше. Одне ядро відслідковує одну ознаку та формує нову послідовність числових значень. Утворена послідовність підлягає обробці pooling шаром для виділення найбільш суттєвих патернів, а потім новими ядрами наступного шару, де виділяються вже більш складні патерни. Потім виділені характеристики збираються у спільний вектор та аналізуються звичайною нейронною мережею [6].

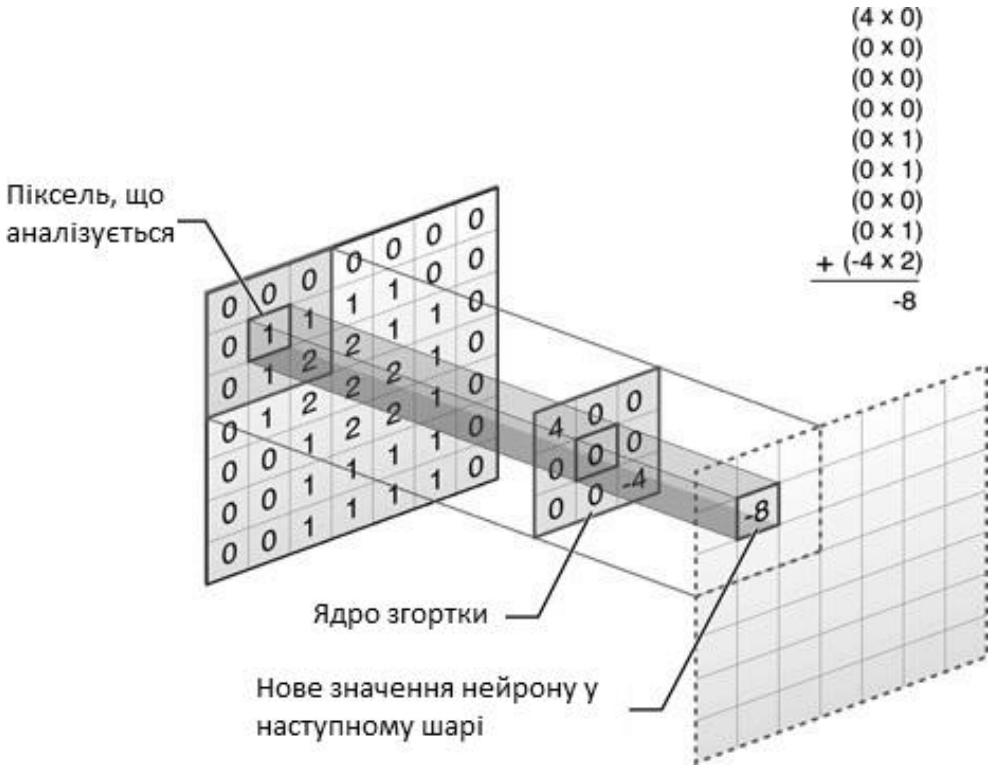


Рис. 1.1. Принцип роботи двомірної згортки

Двомірні згоркові мережі, як привило, призначені для обробки зображень. Їх доцільно використовувати у випадку, коли для даних має принципове значення їх розміщення на площині [5]. Зображення зазвичай мають три канали: RGB, HSL, HSV, а іноді і більше, як у випадку з CMYK. Звичайно, у випадках рантгенівських або просто чорно-білих знімків канал всього один. Ядра двомірної згортки для такого типу даних мають вигляд квадратної матриці. Центр матриці співставляється по черзі з кожним (або з кожним n-им) значенням. Далі відбувається, як і при роботі звичайної нейронної мережі, поелементне множення матриць, та сумевання

результатів. Тільки, на відміну від роботи звичайної штучної нейронної мережі, до уваги береться лише невелика частина вхідних даних та відсутня функція активації (рис. 1.1).

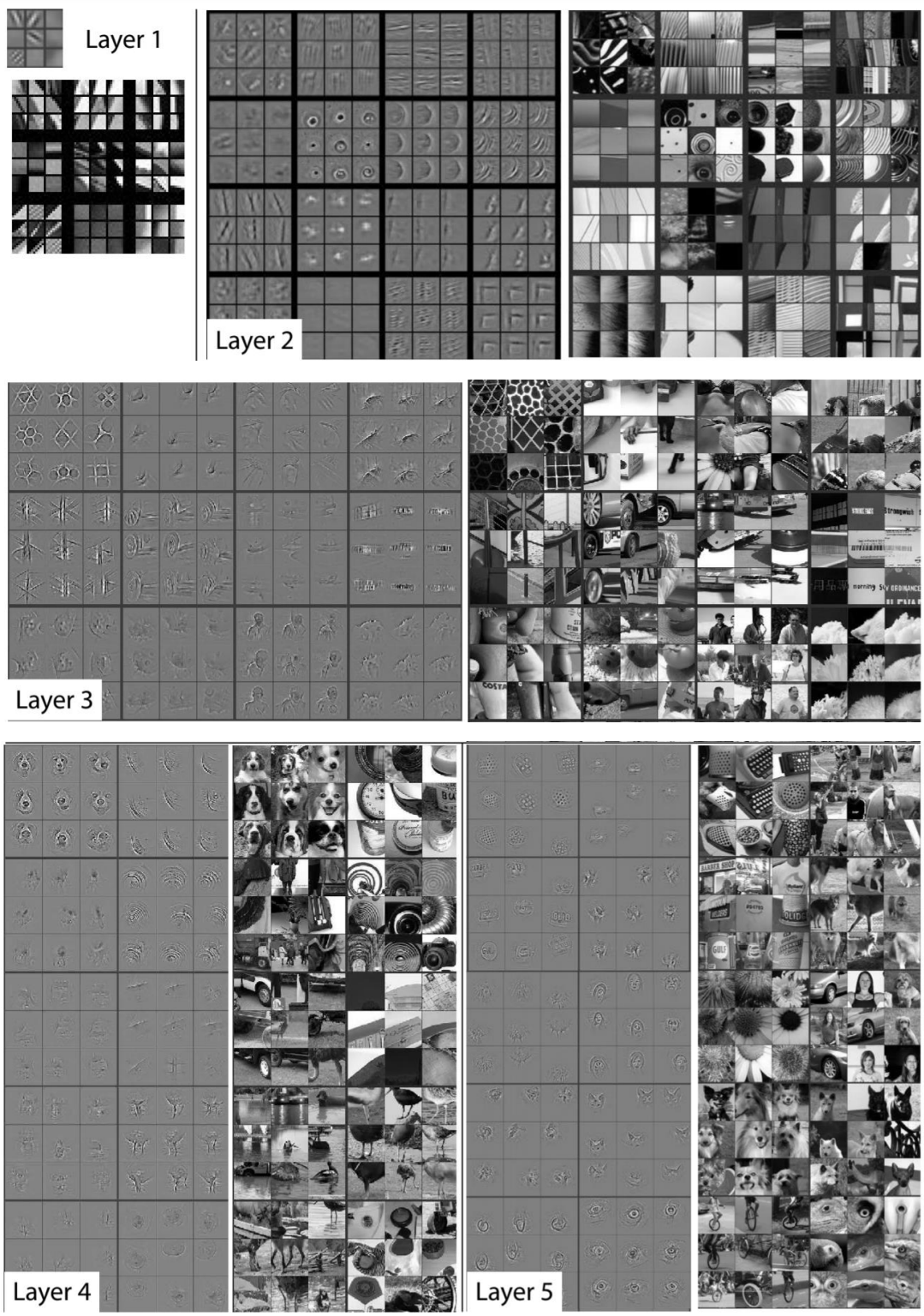


Рис. 1.2. Візуалізація патернів, які розпізнає згорткова нейронна мережа на перших п'яти шарах

Перший шар згортки дає змогу виділити лише прямі лінії будь-якого напрямку. На другому згортковому шарі найронна мережа може розпізнавати складніші патерні, такі як кола, хвилясті та ламані лінії. Трьох шарів достатьо, щоб розпізнати силует людини. А вже на 4-5 шарах виділяються патерни, які можуть розпізнати обличчя (рис. 1.2).

Тривимірні згорткові нейронні мережі використовуються не часто через великі обчислювальні витрати. Однак, вони є незамінними для даних, що відображають об’ємні об’єкти або двовимірні об’єкти у часі. Прикладами таких даних є знімки комп’ютерної томограми, звичайне відео або ж знімки для аналізу грунту та пошуку корисних копалин.

Як і у попередніх випадках, повністю зберігається структура даних, що дає змогу відслідкувати зміни у різних зрізах (рис. 1.3). Тобто при аназі тривимірних даних можна легко віднайти 3D об’єкти, а при аналізі відео відслідкувати не лише зорові образи, а й характер їх руху.

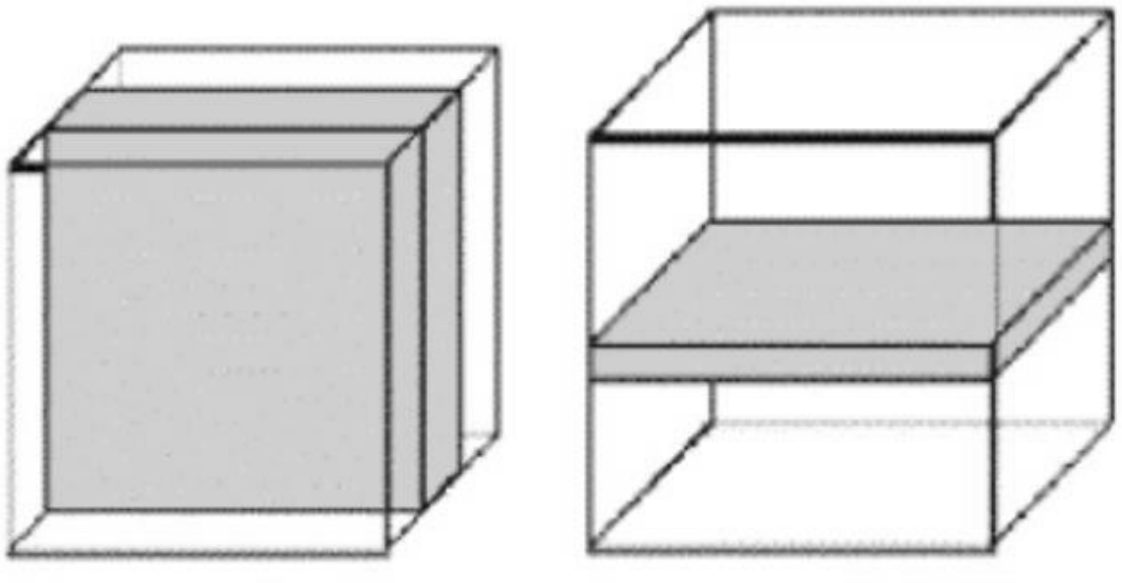


Рис. 1.3. Зрізи по осі *X* та *Z*

Ядро згортки у таких нейронних мережах має вигляд кубу – тривимірного тензору [7].

Звичайно, якщо потрібно буде аналізувати рух тривимірних об’єктів у часі, слід буде використовувати чотиривимірні нейронні мережі. Але наразі це не реалізовано у відкритих інструментах для побудови згорткових нейронних мереж.

Окремо слід виділити обробку тексту. Очевидно, що для роботи нейронної мережі дані мають бути представлені у цифровому форматі. Для представлення тексту у числовому форматі його відображають у деякий векторний простір. Звичайно, можна було б формувати звичайний унітарний код (one hot encoding). Однак, при такому підході слова, що мають однакове значення, матимуть зовсім різне числове представення. Це може створити великі проблеми для нейронної мережі.

Кажучи більш абстракто, геометричне відношеня між векторами слів має відображати семантичний зв’язок між відповідними словами. Тобто, векторне представлення слів має відображати людську мову у геометричний простір. Наприклад, від правильно сформованого простору векторних представлень слід оччікувати, що синоніми будуть відображені у схожі вектори. Тобто геометрична відстань (L2-відстань) між будь-якими двома векторами буде залежати від семантичної відстані між відповідними словами.

Окрім відстані сіж словами, бажано щоб і окремі напрямки у сформованому векторному просторі мали сенс, як зображено на рис. 1.4.

Для кожної мови необхідно будувати власний простір, адже мови не ізоморфні – багато залежить від культури та контексту. Для створення такого простору використовують Embedding шар. Він працює як словник, що відображає код слова у векторний простір. Крім того, що вектор є більш інформативним представленням слова для нейронної мережі, він також зазвичає є набагато меншим ніж початковий код.

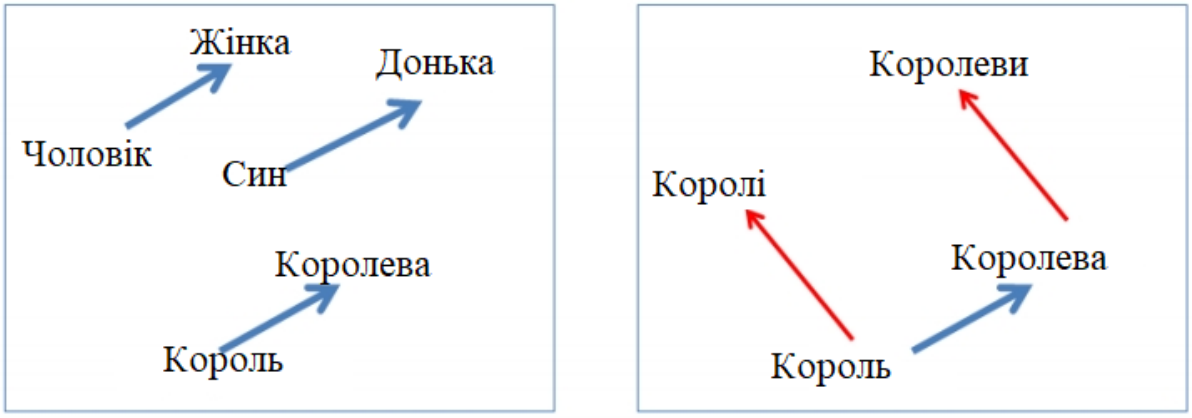


Рис. 1.4. Двомірна площина з векторними представленнями слів, де видно напрямок «від чоловічої статі до жіночої» та «від однини до множини»

Після того, як отримано векторне представлення слів у тексті, можна використовувати одномірну згорткову нейронну мережу. При чому необхідно враховувати, що окремо взята координата вектора не буде несе в собі специфічну інформацію. Тобто при однакових значеннях двох координат, вони нестимуть різну інформацію, навідміну від значень сигналів або пікселів. Тож при використанні згорткової нейронної мережі слід використовувати такий розмір ядра згортки, який відповідає розмірності векторного представлення слова. Тоді кожне ядро зможе виділити певну властивість кожного оремого слова на першому згортковому шарі. А потім оперувати з ними у повнозв'язних шарах.

1.3. Висновки

У першій частині даного розділу наведена основна інформація про протоколи потокової передачі даних прикладного та транспортного рівнів, їх особливості, переваги та недоліки. Наведено вагомі аргументи щодо доцільності використання протоколу UDP у якості транспортного протоколу для потокової передачі даних у реальному часі та його недоліки у інших випадках. Також, розглянуто різні методи маршрутизації та ситуації, коли їх доцільніше використовувати.

У другій частині даного розділу розглянуто особливості конфігурації згорткового шару для аналізу даних з різною структурою. Наведено

приклади даних з розглянутими типами структур та принцип їх аналізу за допомогою конкретної архітектури згорткової нейронної мережі. Окремо розглянуто спосіб роботи з текстовими даними за допомогою Embeddind шару та специфіку подальшої роботи з такими даними.

2. РОЗРОБЛЕННЯ МЕТОДУ ПЕРЕДАЧІ ДАНИХ ЗА ДОПОМОГОЮ НЕЙРОННОЇ МЕРЕЖІ

2.1. Основні етапи методу передачі даних за допомогою нейронної мережі

Розглянемо основні етапи передачі даних, між двома кінцевими пристроями:

1. Завантаження даних до оперативної пам’яті.
2. Ущільнення даних (опціонально).
3. Шифрування даних (опціонально).
4. Відправка по мережі.
5. Прийняття та розшифровка даних.
6. Відновлення даних (якщо було здійснено ущільнення).

Особливістю запропонованого методу є те, що за ущільнення та часткове шифрування даних відповідає нейронна мережа.

На першому етапі один з кінцевіх пристроїв (надалі П1) завантажує необхідні для передачі дані до оперативної пам’яті. Цей етап окремо виділено, оскільки наявність даних у оперативній пам’яті є основною умовою для подальшої їх обробки.

На другому етапі дані проходять крізь згорткові шари нейронної мережі (енкодер), як зображено на рис. 2.1. В результаті формується вектор ознак даних, за яким можна їх відновити. Отриманий вектор може бути в сотні разів менший, ніж початкові дані. Його розмір залежить від конкретних даних та їх типу.

Дані не можуть бути проінтепретовані без другої частини нейронної мережі – декодеру. Але, їх необхідно зашифрувати, оскільки будь-хто може прочитати дані, маючи декодер. Тому, на третьому етапі здійснюється шифрування отриманого вектору ознак.

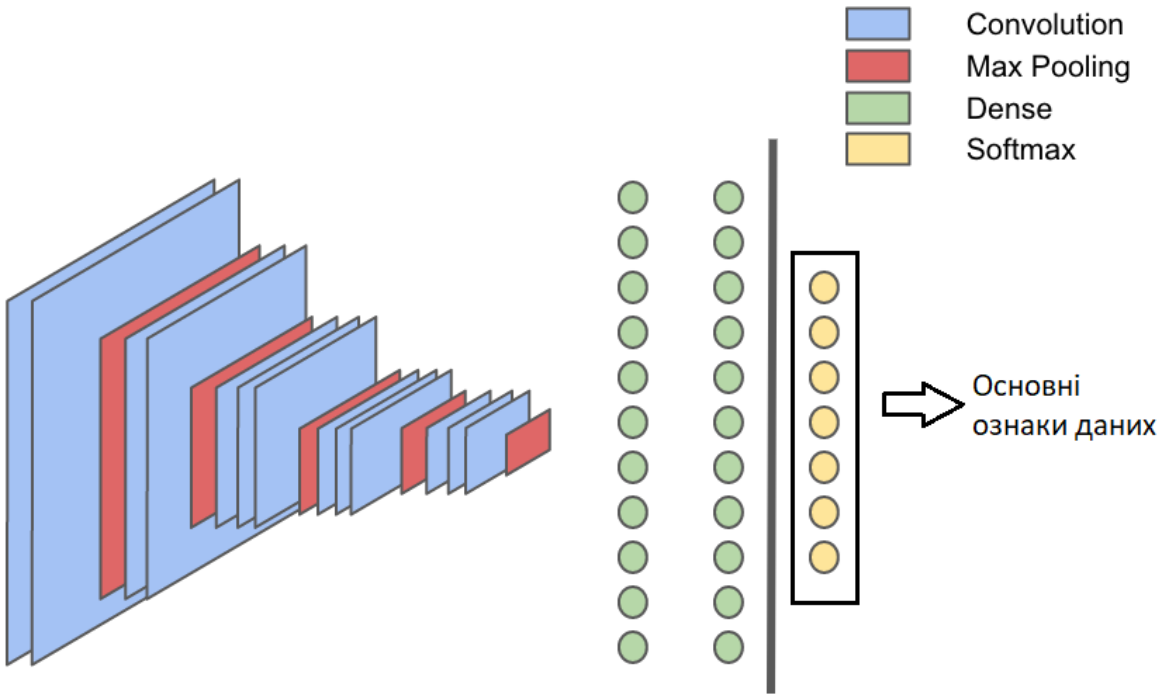


Рис. 2.1. Приклад 2D згоркової нейронної мережі для формування вектору ознак

На червертому етапі відбувається передача зашифрованих даних від П1 до іншого кінцевого пристрою (П2). Протоколи передачі даних залишаються незмінними та залежать від конкретного випадку.

На п’ятому етапі П2 отримує дані та розшифровує їх. Розпаковка мережевих пакетів та перевірка їх цілісності відбувається на рівні мережевих протоколів. Алгоритм шифрування обирається конкретною системою довільним чином.

На початку шостого етапу П2 має закодовані дані – вектор ознак та декодер. Розгорткові шари декодеру відновлюють початкові дані за основними ознаками. Таким чином, П2 отримує відновлені дані.

Тобто, метод передачі даних за допомогою нейронної мережі відповідає за обробку даних на кінцевих пристроях у мережі. Повний шлях даних від одного кінцевого пристрою до іншого можна зобразити як на рис. 2.2.

На рис. 2.2 наведено всі шість етапів методу, що розглянуто вище. З рис. 2.2 видно залежність розмірності згорткового шару штучної нейронної мережі від типу даних, що передається мережею.

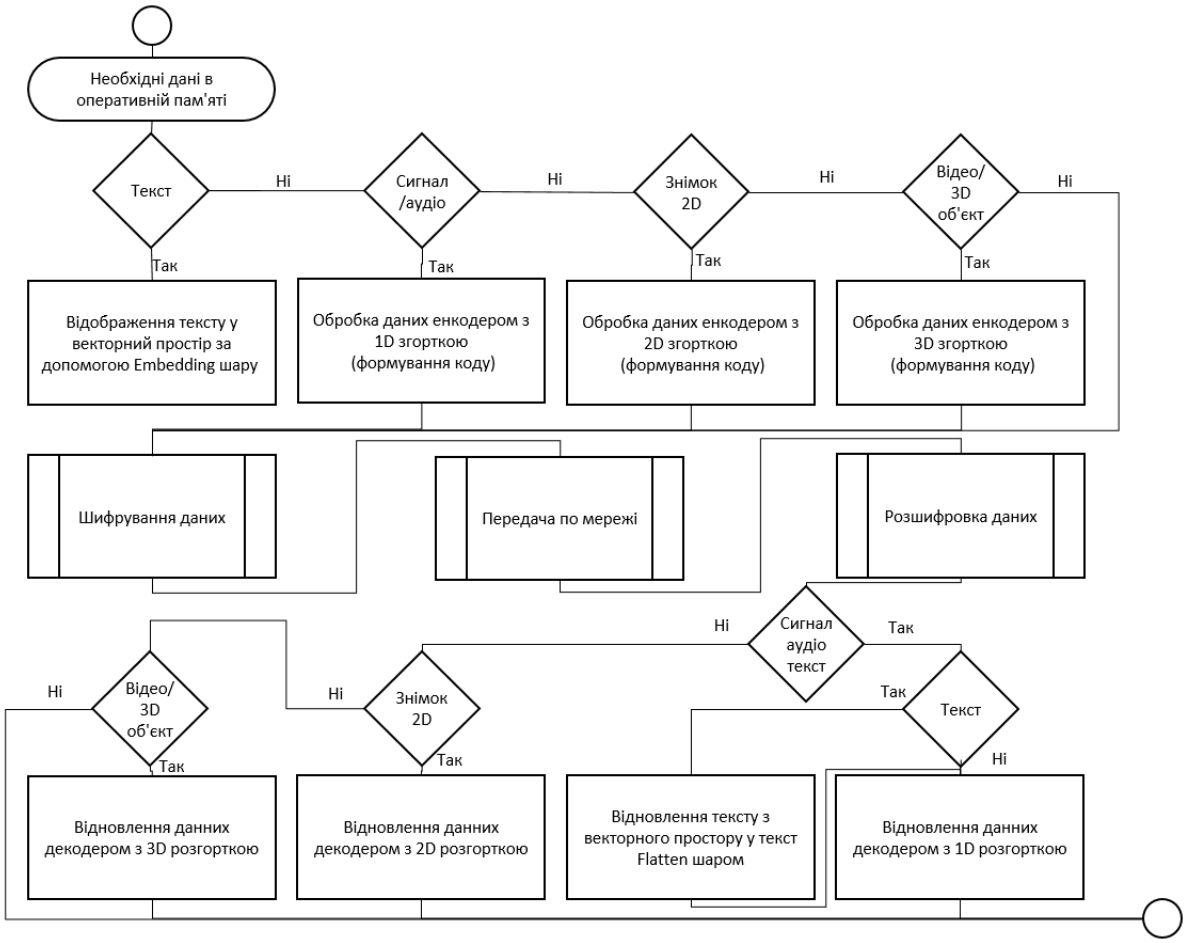


Рис. 2.2. Шлях даних від однієї кінцевої точки у мережі до іншої

Тобто, метод починає свою роботу на етапі підготовки даних до відправки по мережі. На пристрої, що здійснює відправку даних знаходиться енкодер. Він обробляє дані відповідним енкодером в залежності від їх типу.

А завершується робота відновленням даних після їх прийняття з мережі. Ця дія, як і ущільнення даних відбувається декодерами з різною архітектурою, залежно від типу даних.

2.2. Принцип роботи варіаційного автоенкодеру

Автоенкодер складається з двох частин (див. рис. 2.3): енкодеру та декодеру, що мають аналогічну структуру. Найчастіше у автоенкодерах застосовіють комбінації згорткових та повнозв’язних шарів.

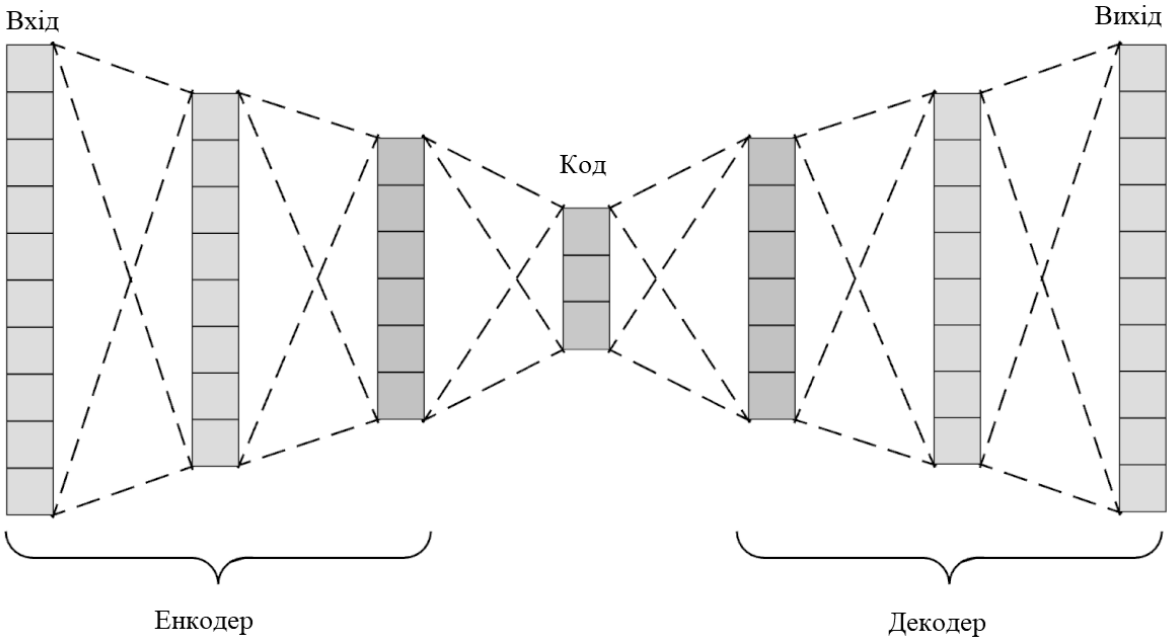


Рис. 2.3. Архітектура автоенкодеру

Дані, що подаються на вхід проходять спочатку крізь різні згорткові шари, потім розкладаються в один вектор та проходять повнозв’язні шари. В результаті початкові дані відображаються у прихований векторний простір.

На вхід декодеру подається код, утворений енкодером. Код проходить зворотний шлях: спочатку повнозв’язні шари, потім розгорткові шари. В результаті отримуємо початкові дані.

Автоенкодер навчається використовуючи у якості цільових даних ті ж зображення, що подаються на вхід. Таким чином автоенкодер й навчається відновлювати поаткові дані.

Розглянемо принцип роботи автоенкодера на прикладі. Будь-які дані – це елеманти багатомірного простору. Надалі будемо розглядатидані, що містять ЕКГ – одномірні масиви даних на 500 елементів.

Масиви сигналів ЕКГ – це елементи 500-мірного простору, як і взагалі будь-які масиви на 500 елементів. Однак, серед усіх можливих масивів, саме ЕКГ займають лише крихітну частку. Абсолютна більшість – шум. З іншого ж боку, якщо взяти довільне ЕКГ, то і масиви у деякому околі також можна вважати ЕКГ.

Якщо взяти два довільних масиви з ЕКГ, то в початковому 500-мірному просторі можна знайти неперервну криву, всі точки якої можна також вважати електрокардіограмами. Якщо ж взяти до уваги і попереднє зауваження, то сигналами ЕКГ можна вважати усі точки деякої області вздовж кривої.

Таким чином, у просторі всіх масивів є деякий підпростів меншої розмірності, в області довкола якого розташувались сигнали ЕКГ. Тобто, якщо наша генеральна сукупність – це всі сигнали ЕКГ, які тільки існують в теорії, то щільність імовірності зустріти сигнали ЕКГ у межах цієї області набагато вища, ніж поза її межами.

Автоенкодери з розмірністю k шукають k-мірний многовид у просторі об’єктів, який найбільш повно передає усі варіації вибірки. Сам код задає параметризацію даного многовиду. При цьому енкодер співвідносить об’єкту його параметр на многовиді, а декодер параметру співвідносить точку у просторі об’єктів [8].

Чим більша розмірність кодів, тим більше варіацій у даних зможе передати автоенкодер. Якщо розмірність кодів є замалою, то автоенкодер запам’ятає середнє по варіаціям, яких не вистачає. На практиці це може згладжувати деякі піки електрокардіограм у напрямку більш типового сигналу.

Для подальших пояснень для простоти візуалізації візьмемо множину точок у двомірному просторі. Нехай, область інтересу становлять лише точки розташовані у деякому околі вздовж кривої (рис. 2.4).

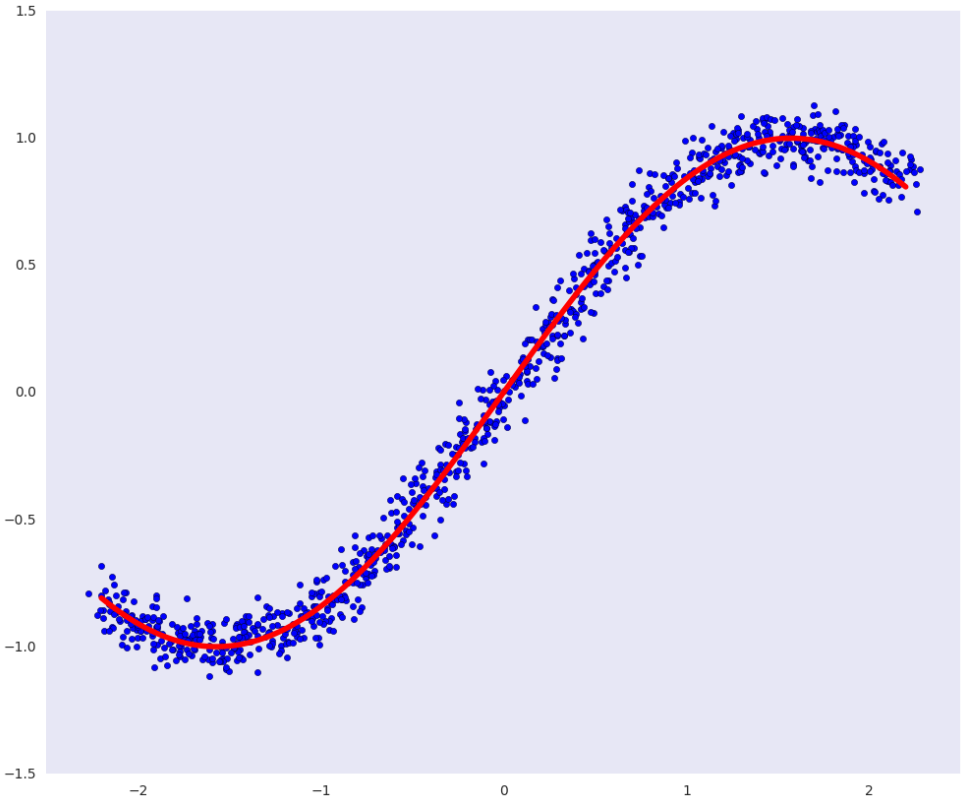


Рис. 2.4. Деяка вибірка точок у двомірному просторі

Найпростішим прикладом ущільнюючого автоенкодеру є двошаровий ущільнюючий автоенкодер з лінійними функціями активації (при лінійній функції активації більша кількість шарів не має сенсу). Такий автоенкодер шукає афінниї підпростір у просторі об’єктів, який описує найбільшу варіацію в об’єктах (рис. 2.5). Даний автоенкодер працює так само, як PCA (метод основних компонент) і знаходить той самий підпростір.

На рис. 2.5 зображено наступні компоненті:

- біла пряма лінія – многовид, в який переходять точки після автоенкодеру;
- пряма лінія – многовид, в який переходять точки після PCA;
- великі точки – деяка вибірка точок;

• зірки на лінії – образи різнокольорових точок після автоенкодеру.

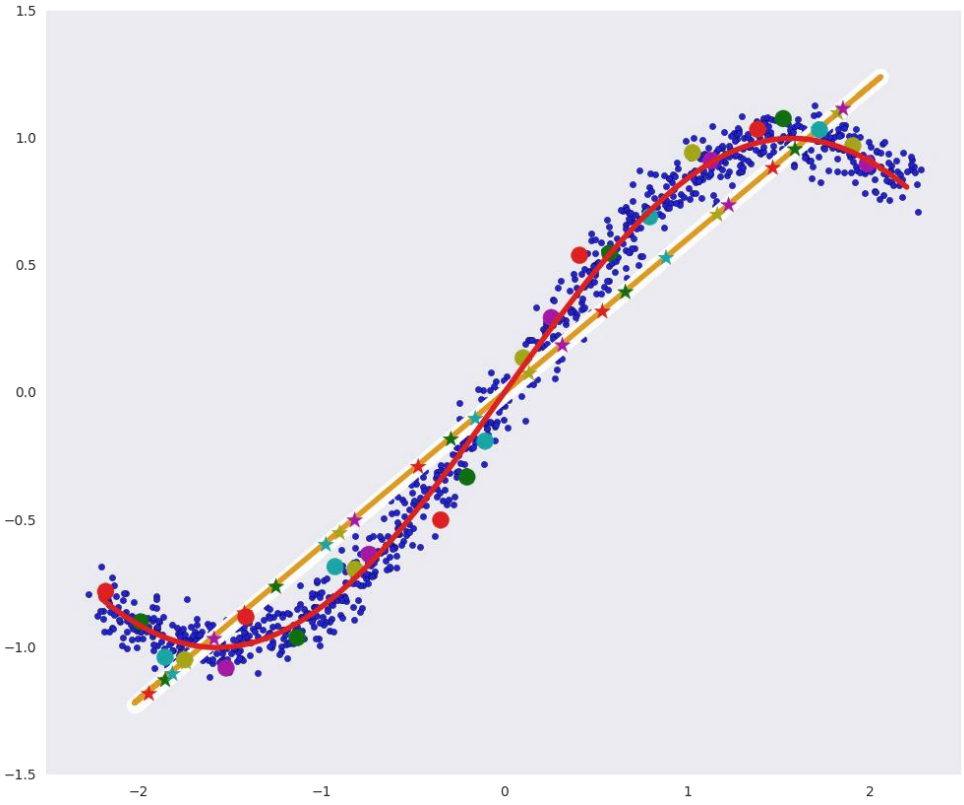


Рис. 2.5. Відображення точок двомірного простору у одномірний

Очевидно, що лінійний автоенкодер не такий корисний як той, що може знаходити довільні залежності у даних. Для цього достатньо до енкодеру та декодеру додати по обному шару та нелінійну функцію активації.

Перевіримо роботу глибокого автоенкодера. Візьмемо три шари и головне – нелінійну функцію активації між ними. Такому автоенкодеру значно краще вдається побудувати визначаючий многовид: біла крива майже співпадає з червоною. Тобто теоретично глибокий автоенкодер може знайти многовид довільної складності, наприклад, такий, біля якого розташовані всі сигнали електрокардіограм у 500-мірному просторі.

2.3. Функція втрат та метрики точності відновлення

У математичній оптимізації та теорії рішень функція втрати або функція витрат є функцією, яка відображає подію або значення однієї або

декількох змінних на реальне число, інтуїтивно представляючи деяку "вартість", пов'язану з подією. Проблема оптимізації спрямована на мінімізацію функції втрат. Цільовою функцією є або функція втрати, або її негатив (у конкретних областях, що називають функцією винагороди, функцією прибутку, функцією корисності, функцією придатності тощо), і в цьому випадку вона повинна бути максимальною.

Функція втрати є важливою частиною в штучних нейронних мережах, яка використовується для вимірювання невідповідності між прогнозованою величиною (ŷ) і фактичною (y). Це невід'ємне значення, де робастність моделі зростає разом зі зменшенням значення функції втрат. Функція втрати є жорстким ядром емпіричної функції ризику, а також важливою складовою функції структурного ризику.

Для кожного класу задач у глибинному навчанні є своя функція втрат:

- середня абсолютна помилка (Mean Absolute Error, MAE) – для задач регресії;
- середній абсолютний відсоток помилки (Mean Absolute Percentage Error, MAPE) – для задач регресії;
- L1 – для задач регресії;
- перехресна ентропія (Cross Entropy) – для класифікації;
- Пуассона (Poisson) – для перевірки відповідності розподілу.

Для навчання автоенкодера також створили функцію втрат. Вона називається функцією втрат реконструкції (reconstruction loss).

Як зазначалось вище, автоенкодер складається з двох частин: енкодера та декодера. Нехай, енкодер приймає на вхід x, віддає його код – приховане представлення – z, і воно має ваги і зміщення θ. Щоб бути конкретними, повернемось до MNIST та скажима, що x - це фотографія рукописного числа розміром 28 на 28 пікселів. Енкодер «кодує» дані, які є 784-мірними, в прихований простір представлений z, розмір якого набагато менше, ніж 784. Це, як правило, називається «пляшкове горлечко», оскільки енкодер повинен навчитися ефективному ущільненню даних у цей

зменшений простір. Позначимо енкодер $q_\theta(z \mid x)$. Зазначимо, що менший простір є стохастичним: енкодер виводить параметри до $q_\theta(z \mid x)$, який є щільністю ймовірності Гауса. Ми можемо навіть генерувати нові дані з цього розподілу, щоб отримати зашумлені значення представлення z.

Декодер – інша частина нейронної мережі. Він на вхід приймає представлення z, яке утворюється у енкодері з параметрів розподілу ймовірностей даних. Як і енкодер, має ваги та зміщення ϕ. Декодер позначається $p_\phi(x \mid z)$. Тоді, якщо говорити, наприклад, про чорно-білі зображення, нехай, значення пікселя може набувати значення 0 або 1. То розподіл ймовірності одного пікселя може бути представлено за допомогою розподіла Бернулі. Декодер отримує на вхід приховане значення z та виводить 784 параметри Бернулі, по одному для кожного з 784 пікселів на зображенні. Декодер відновлює зі значень z 784 числа від 0 до 1, але частину інформації може бути втрачено, адже дані переходять з меншої розмірності до більшої. Щоб порахувати кількість втраченої інформації використовують логарифмічну ймовірність реконструкції $\log p_\phi(x \mid z)$. Дане значення показує, наскільки ефективно декодер навчився відновлювати вхідні дані x з їх представлення z.

Функція втрат автоенкодера є негативною логарифмічною правдоподібністю з регуляризацією. Через те, що немає спільного представлення між усіма точками даних, ми можемо розкласти функцію втрат лише на частини, які залежать від однієї точки даних. Функція втрат l_i для точки даних x_i має вигляд (1):

$$l_i(\theta, \phi) = -E_{z \sim q_\theta(z \mid x_i)}[\log p_\phi(x_i \mid z)] + KL(q_\theta(z \mid x_i) \,||\, p(z)) \tag{1}$$

А загальна функція втрат (2) відповідно обчислюються як сума функцій втрат у кожній точці (1).

$$l(\theta, \phi) = \sum_{i=1}^{N} l_i(\theta, \phi) \tag{2}$$

Перша частина формули (1) – це втрата від реконструкції або очікувана негативна логарифмічна ймовірність *i*-тої точки даних. Вона відповідає за заохочення декодера до навчання відновлення даних. Якщо вихід декодера не реконструює дані добре, то можна сказати, що декодер параметризує розподіл правдоподібності, який не відображає велику частину ймовірностей у правильні дані. Тобто, якщо декодер на зображеннях ставить білі точки на місці чорних – це є найгіршим випадком реконструкції і функція втрат набуває найбільшого значення.

Друга частина формули (1) – регуляризатор. Це розбіжнасть Кулбака-Лейблера між розподілом енкодера $q_\theta(z \mid x)$ та $p(z)$. Ця дивергенція є невід'ємним функціоналом, який є несиметричною мірою віддаленості одне від одного двох ймовірносних розподілів і показує скільки інформації втрачається при використанні q для представлення p.

У автоенкодері p визначається як нормальний розподіл з середнім значенням, рівним 0 на дисперсією 1, тобто $p(z) = Normal(0,1)$. Якщо енкодер виводить представлення z, які відрізняються від нормального розподілу, то він отримує штраф (значення функції втрат приймає більше значення). Таким чином регуляризація допомагає запобігати тому, що, наприклад, значення кожної цифри не були зовсім різними. Якщо не включати регуляризатор до функції втрат, автоенкодер міг би навчитись надавати кожній точці даних представлення у різних кінцях евклідового простору. Це було б дуже погано, адже тоді два зображення одного числа, написаного різними людьми могди б мати зовсім різні представлення. Оскільки бажаною поведінкою автоенкодера є побудова значущого простору для представлення, то така поведінка карається і значення двох цифр залишаються досить близькими. Тобто автоенкодер навчається розпізнавати загальні риси об'єктів та виділяти їх конкретні характеристики [14].

У даній роботі оптимізація функції втрат (2) автоенкодера здійснюється за допомогою градієнтного спуску.

2.4. Висновки

У першому підрозділі даного розділу описано шість основних етапів методу передачі даних за допомогою автоенкодеру. Розглянуто на яких етапах та з якою метою використовується нейронна мережа.

У другому підрозділі докладно описано принцип роботи звичайного лінійного авоенкодера. Розглянуто приклад роботи нейронної мережі з ЕКГ.

У останньому підрозділі розглянуто «область відповідальності» методу: коли він починає свою роботу, змінюючи звичайний процес передачі даних, та коли завершує. Наведено схему роботи методу в залежності від типів даних, що передаються мережею.

3. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

3.1. Обгрунтування вибору мови програмування

За статистикою останніх років для обробки даних та побудови моделей використовують наступні мови програмування:

- Python
- R
- Scala
- C#

Розглянемо переваги та недліки кожної з них для порівняння.

Python – високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує багато різноманітних підходів до програмування, зокрема: структурне, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване програмування. Основними архітектурними рисами є динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточних обчислень а також високорівневі структури даних. Підтримується розбиття програм на модулі, які, в свою чергу, можуть об'єднуватися в пакети.

CPython – інтерпретатор, що є еталонною реалізацією Python. Підтримується більшістю сучасних платформ. Поширюється під вільною ліцензією Python Software Foundation License дозволяє використовувати його без обмежень в будь-яких додатках, включаючи пропрієтарні. Також існує реалізація інтерпретатора для JVM з можливістю компіляції, CLR, LLVM та інші незалежні реалізації. Проект PyPy використовує JIT- компіляцію, яка значно збільшує швидкість виконання Python-програм.

Мова дуже активно розвивається. Нові версії з додаванням та зміною мовних властивостей виходять в середньому раз на два з половиною роки.

Мова не підлягала офіційній стандартизації, роль стандарту де-факто виконує CPython, що розробляється під контролем автора мови.

У останні роки Python займає перше місце над іншими мовами програмування, такими як C, C ++ і Java, і широко використовується програмістами. Ця мова зазнала різких змін з моменту виходу понад 25 років тому, оскільки було додано багато додаткових функцій. Python 1.0 мав модульну систему Modula-3 і взаємодіяв з операційною системою Amoeba з різноманітними інструментами функціонування. Python 2.0, представлений у 2000 році, мав особливості збирання сміття та підтримки Unicode. Python 3.0, представлений у 2008 році, мав конструктивний дизайн, що уникає дублювання модулів і конструкцій. З доданими функціями, зараз компанії використовують найстабільнішу версію – Python 3.5, хоча найновішою версією наразі є Python 3.7

Компанії, що займаються розробкою програмного забезпечення, віддають перевагу мові Python через його різноманітні можливості та меншу кількість програмних кодів. Майже 14% програмістів використовують його на операційних системах, таких як UNIX, Linux, Windows і Mac OS.

Мова Python має різноманітне застосування в обалстях розробки програмного забезпечення, таких як ігри, веб-фреймворки, розробка мов, створення прототипів, додатки графічного дизайну і т.д. Деякі з його переваг:

- широка підтримка бібліотек (Python надає великі стандартні бібліотеки, які включають такі області, як операції з рядками, Інтернет, інструменти веб-сервісів, інтерфейси та протоколи операційної системи);
- легка інтеграція (він має потужні можливості керування, наприклад, виклик функцій, безпосередньо через C, C ++ або Java через Jython. Python також обробляє XML та інші мови розмітки, оскільки може працювати на всіх сучасних операційних системах за допомогою одного і того ж байтового коду);

- ефективність роботи програміста (як згадувалось раніше, мова дає широкий вибір бібліотек та підходів високого рівня, що створює перевагу над такими мовами як Java, VB, Perl, C, C ++ і C #);
- продуктивність (завдяки потужним можливостям інтеграції процесів, система тестування модулів та покращені можливості керування сприяють збільшенню швидкості для більшості програм та їх продуктивності. Це чудовий варіант для створення масштабованих мультипротоколових мережевих додатків).

Однак, дана мова не є ідіальною та має ряд недоліків, що слід враховувати при роботі:

- ускладнює сприйняття інших мов (після Python важко звикнути до необхідності визначати типи змінних або розставляти фігурні дужки);
- не використовується для мобільної розробки;
- відносно повільний (за рахунок інтерпретації, а не компіляції мови, команди виконуються довше та без оптимізації);
- часті помилки виконання (помилка в коді, що не скомпілювалася б виявляється лише тоді, коли програма заходить у відповідну гілку виконання, що викликає необхідність у більш ретельному тестуванні).

R – мова програмування для статистичної обробки даних і роботи з графікою, а також вільне програмне середовище обчислень з відкритим вихідним кодом в рамках проекту GNU. Мова створювалась як аналог мови S, розробленої в Bell Labs, і є його альтернативною реалізацією, хоча між мовами є істотні відмінності, але в більшості своїй код на мові S працює в середовищі R. Спочатку R був розроблений співробітниками статистичного факультету Оклендського університету Россом Айхекой (англ. Ross Ihaka) і Робертом Джентлменом (англ. Robert Gentleman) (перша буква їх імен – R); мова і середовище підтримуються і розвиваються організацією R Foundation [10].

R широко використовується як статистичне програмне забезпечення для аналізу даних і фактично став стандартом для статистичних програм [11].

R доступний під ліцензією GNU GPL. Поширюється у вигляді вихідних кодів, а також відкомпільованих додатків під ряд операційних систем: FreeBSD, Solaris та інші дистрибутиви Unix і Linux, Microsoft Windows, Mac OS X.

В R використовується інтерфейс командного рядка, хоча доступні і кілька графічних інтерфейсів користувача, наприклад пакет R Commander, RKWard, RStudio, Weka, Rapid Miner, KNIME, а також засоби інтеграції в офісні пакети.

У 2010 році R увійшов до списку переможців конкурсу журналу Infoworld в номінації на краще відкрите програмне забезпечення для розробки додатків.

Наразі мова належить та підтримується компанією Microsoft. Мова має безліч інструментів для різних типів аналізу як чисто статистичного, так і для побудоми більш складних моделей. Однак, мова не має власного інструменту для розробки нейронних мереж, а використовує перекомпільовані інструменти Python – Keras.

Scala – мультипарадигмальна мова програмування, спроектована лаконічною і типобезпечною для простого і швидкого створення компонентного програмного забезпечення, що поєднує можливості функціонального і об'єктно-орієнтованого програмування.

Коли справа доходить до обробки даних, Apache Spark є широко використовуваним інструментом у галузі, який написаний на мові програмування Scala. Spark є розширенням для Hadoop, який робить пакетну обробку, а також обробку в реальному часі. У порівнянні з Hadoop, Spark є більш ефективним з багатьох причин.

Scala є розширенням Java-мови, яка взаємодіє з java, оскільки вона працює на JVM. Вона має багато різних функцій у порівнянні з java і мовою,

більш орієнтованою на функціональне програмування. Оскільки Spark написано за допомогою Scala, для користувачів Spark Scala буде більш доцільним у порівнянні з іншими мовами. Таким чином, використовуючи Scala, ми можемо отримати максимум з Spark без будь-яких обмежень.

Однак, технологія Spark вимагає довгих та важких налашьувань, а також запуску окремого серверу, що робить початок роботи на ньому нелегким завданям.

C# – об'єктно-орієнтована мова програмування, розроблена у 2001 році групою інженерів Microsoft. C# відноситься до сімейства мов з C- подібним синтаксисом, його синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML.

Перейнявши багато від своїх попередників – мов C++, Pascal, Модула, Smalltalk і, особливо, Java – C#, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, C# на відміну від C++ і деяких інших мов, не підтримує множинне успадкування класів (між тим допускається множинне спадкування інтерфейсів).

Для роботи з даними у 2019 році було випущено у вільне використання фреймворк ML.NET. ML.NET надає моделі на основі машинного навчання аналітичні та прогнозні можливості для існуючих .NET розробників. Структура побудована на .NET Core і .NET Standard, успадковуючи здатність запускати крос-платформні на Linux, Windows і macOS. Хоча фреймворк ML.NET є новим, його початок розпочався у 2002 році як проект Microsoft Research під назвою TMSN (пошук та навігація за текстовою гілкою) для внутрішнього використання в

продуктах Microsoft. Пізніше вона була перейменована в TLC близько 2011 року.

Для проведення дослідження у даній роботі було вирішено використовувати мову програмування Python. Адже вона має найбільш зручні інструменти для швидкої розробки рішень з обробкою даних.

Як середу розробки обрано Jupyter Notebook, адже він дозволяє поєднувати код, текст та діаграми на одній сторіці та легко передавати їх іншим користувачам.

3.2. Обгрунтування вибору технологій для створення та навчання нейронних мереж

Для створення та навчання нейронних мереж у Python існує кілька бібліотек і, навіть, фреймворків. Основними є:

- Tensorflow
- PyTorch
- Keras

TensorFlow – відкрита програмна бібліотека для машинного навчання, розроблена компанією Google для вирішення завдань побудови і тренування нейронних мереж з метою автоматичного знаходження та класифікації образів, досягаючи якості людського сприйняття. Застосовується як для досліджень, так і для розробки власних продуктів Google. Основний API для роботи з бібліотекою реалізований для Python, також існують реалізації для C++, Haskell, Java, Go і Swift.

TensorFlow може працювати на декількох процесорах і графічних процесорах (з додатковими розширеннями CUDA і SYCL для обчислень загального призначення на пристроях обробки графіки). TensorFlow доступний на 64-розрядних платформах Linux, MacOS, Windows і мобільних комп'ютерах, включаючи Android і iOS.

Її гнучка архітектура дозволяє легко розгортати обчислення на різних платформах (процесори, графічні процесори, ТПУ), а також від настільних комп'ютерів до кластерів серверів до мобільних і крайових пристроїв.

Обчислення тензорного потоку виражаються у вигляді графіків потоків даних, що є станами. Назва TensorFlow випливає з операцій, які такі нейронні мережі виконують на багатовимірних масивах даних, які називаються тензорами. Під час конференції Google I/O у червні 2016 року Jeff Dean заявив, що 1500 сховищ на GitHub згадували TensorFlow, з яких тільки 5 належали Google.

У січні 2018 року Google випустив TensorFlow 2.0. У березні 2018 року компанія Google випустила TensorFlow.js версію 1.0 для машинного навчання в JavaScript і TensorFlow Graphics для глибокого навчання в комп'ютерній графіці.

Дана бібліотека є дуже потужним інструментом для створення та навчання нейронніих мереж. Однак, має складне API, що значно ускладнює розробку.

PyTorch – сучасна бібліотека глибокого навчання, розвивається під крилом Facebook. Вона не схожа на інші популярні бібліотеки, такі як Caffe, Theano і TensorFlow. Вона дозволяє дослідникам втілювати в життя свої найсміливіші фантазії, а інженерам з легкістю ці фантазії імплементувати.

PyTorch є аналогом фреймворка Torch7 для мови Python. Розробка його почалася в надрах Facebook ще в 2012 році, всього на рік пізніше появи самого Torch7, але відкритим і доступним широкому загалу PyTorch став лише в 2017 році. З цього моменту фреймворк дуже швидко набирає популярність і привертає увагу все більшого числа дослідників.

На відміну від TensorFlow, PyTorch динамічно будує граф обчислень, що позбавляє від необхідності «компілювати» модель, хоча і сповільнює трохи роботу. Як пишеться на самій сторінці Facebook: «Незважаючи на те, що статичні графіки чудово підходять для розгортання продукції, процес дослідження, який бере участь у розробці наступного великого алгоритму, є дійсно динамічним. PyTorch використовує техніку, що називається авто - диференціацією зворотного режиму, яка дозволяє розробникам

змінювати поведінку мережі довільно з нульовим відставанням або накладними витратами, прискорюючи ітерації досліджень» [12].

Keras – відкрита нейромережева бібліотека, написана на мові Python. Вона являє собою надбудову над фреймворками Deeplearning4j, TensorFlow і Theano. Націлена на оперативну роботу з мережами глибинного навчання, при цьому спроектована так, щоб бути компактною, модульної та розширюється. Вона була створена як частина дослідницьких зусиль проекту ONEIROS (англ. Open-ended Neuro-Electronic Intelligent Robot Operating System), а її основним автором і підтримуючим є Франсуа Шолль (фр. François Chollet), інженер Google.

Планувалося що Google буде підтримувати Keras в основний бібліотеці TensorFlow, однак Шолль виділив Keras в окрему надбудову, так як згідно з концепцією Keras є скоріше інтерфейсом, ніж наскрізний системою машинного навчання. Keras надає високорівнева, більш інтуїтивний набір абстракцій, який робить простим формування нейронних мереж, незалежно від використовуваної в якості обчислювального бекенд бібліотеки наукових обчислень. Microsoft працює над додаванням до Keras і низькорівневих бібліотек CNTK.

В результаті вивчення особливостей кожної з бібліотек було вирішено використовувати Keras, як зручну надбудову над TensorFlow, адже така комбінація є найоптимальнішою з точки зору часу розробки та можливостей самої бібліотеки.

3.3. Висновки

У даному розділі розглянуто переваги та недоліки чотирьох мов програмування, що за статистикою на GitHub частіше за інші використовуються розробниками програмного забезпечення та аналітиками для обробки даних. Внаслідок аналізу обрано інтерпретовну мову програмування Python, як мову, що має найбільше зручних інструментів для роботи з даними, що підтримуються. Ця мова також є найпопулярнішою мовою для побудови нейронних мереж та може використовуватись для

створення клієнт-серверних програм, що є дуже корисним на етапі тестування методу.

Також наведено особливості роботи з трьома відомими інструментами для побудови та навчання нейронних мереж. В результаті проведеного аналізу було обрано бібліотеку Keras, що працює на основі TensorFlow. Таке поєднання є оптимальним з точки зору швидкості розроблення та можливостей бібліотеки.

4. РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДОСЛІДЖЕННЯ МЕТОДІВ ПЕРЕДАЧІ ДАНИХ ПО МЕРЕЖІ

4.1. Генератори даних для тестування

Для перевірки швидкості роботи методу передачі даних за допомогою автоенкодера та навантаження на мережу, що ним створюється, було обрано наступні набори даних:

- набір сигналів ЕКГ для перевірки роботи методу з сигналами;
- набір зображень MNIST для перевірки роботи методу з зображеннями;
- згенеровене відео із зображень MNIST для перевірки роботи методу з відео;
- тексти статей з Википедії для перевірки роботи методу з текстом.

Для винесення логіки завантаження різних типів даних до пам’яті програми написано окремий модуль, що генерує дані заданого розміру та віддає їх порціями. Модуль складається з чотирьох класів: по одному на кожний з типів даних. Всі вони наслідують один клас Sequence з модулю keras.utils. Створені генератори були використані також при навчанні автоенкодерів.

Завданям генератора є:

- визначення файлів, що будуть завантажені за весь час;
- визначення кількості порцій, за яку буде завантажено увесь обсяг даних;
- визначення файлів, що мають бути завантажені у поточній порції даних;
- завантаження необідних файлів;
- нормалізація даних;
- приведення даних до заданого розміру;

Наведемо приклад коду основних методів генератора зображень у лістингу 1.

Лістинг 1. Клас генератора зображень

```
class ImgGenerator(Sequence):
    def __init__(self, list_IDs, batch_size=2, dim=image_size):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.list_IDs = list_IDs
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        indexes =
         self.indexes[index*self.batch_size:(index+1)*self.batch_si
         ze]
        list_IDs_temp = [self.list_IDs[k] for k in indexes]
        X = self.__data_generation(list_IDs_temp)
        return X, None

    def __data_generation(self, list_IDs_temp):
        'Generates data containing batch_size samples'

        x_test = []
        x_test_ = []
        for i in list_IDs_temp:
            x_test.append(cv2.cvtColor(cv2.imread("img/data{}.jpg".
         format(i)), cv2.COLOR_BGR2RGB).astype('float32') / 255)
        x_test_ = np.zeros([self.batch_size, image_size[0],
         image_size[1], 3])
        for i in range(self.batch_size):
            x_test_[i] = cv2.resize(x_test[i], (self.dim[1],
         self.dim[0]))
        x_test = x_test_.astype(float)
        return x_test
```

4.2. Створення клієнт-серверної програми для тестування методу

Для перевірки швидкості роботи методу передачі даних по мережі за допомогою автоенкодера та без нього було створено два скрипти: клиєнтський та серверний.

Клієнт відповідає за завантаження та відправку даних. Сервер – за прийняття та віддтворення даних. Обидві частини були створені за допомогою бібліотеки zmq та працюють за мережевим протоколом TCP. У

лістингу 2 та 3 наведено код створення, налаштування та з’іднання клієнту та серверу відповідно.

Лістинг 2. Створення клієнта

```
import zmq

context = zmq.Context()
footage_socket = context.socket(zmq.PUB)
footage_socket.connect('tcp://192.168.1.9:5555')
```

Лістинг 3. Створення сервера

```
import zmq

context = zmq.Context()
footage_socket = context.socket(zmq.SUB)
footage_socket.bind('tcp://192.168.1.9:5555')
footage_socket.setsockopt_string(zmq.SUBSCRIBE, np.unicode(''))
```

Як видно з лістингів 2 та 3, клієнт під’єднувався до сервера по локальній мережі. Локальна мережа, звісно, не є настільки завантаженою, як глобальна мережа, тож великого приросту у кількості переданих файлів помічено не було. Однак, різниця навантаженості мережі була помітна на ресурсних моніторах.

Обидві частини мають функцію завантаження моделі енкодера або декодера. Дана функція відповідає не тільки зававнтаження ваги кожного з нейронів, вона також завантажує саму архітектуру штучної нейронної мережі. Така можливість позбавляє необхідності написання великого коду створення нейронної мережі у коді та, навіть, імпортувати відповідні модулі у програму. Код даної функції наведено у лістингу 4. Функція приймає всього два параметри: файл з вагами та файл зі структурою самої нейронної мережі. Файл з вагами має стандартне для keras розширення «.h5». А архітектура нейронної мережі за допомогою серіалізації зберігається у форматі JSON та десеріалізується з використанням фукції model_from_json з бібліотеки keras.

Лістинг 4. Код завантаження моделі

```
from keras.models import model_from_json

def get_model(file_model, file_weights):
    json_file = open(file_model, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    loaded_model.load_weights(file_weights)

    return loaded_model
```

Після того, як з'єднання встановлено та підготовано код для завантаження моделі, відбувається налаштування параметрів. По-перше, обирається тип та розмір даних для тестування. По-друге, завантажується сама модель (енкодер на стороні відправники-клієнта та декодер на стороні приймача-сервера). По-третє, на сервері створюється лічильник файлів, що були прийняті, на ньому встановлюється значення 0. Останнім кроком є збереження поточного часу для того, щоб зупинити тест рівно за хвилину після початку та перевірити результат.

Далі запускається цикл, в якому клієнт постійно надсилає дані, а сервер постійно їх приймає. У першому випадку (див. Лістинг 5, 6), який перевіряється, процес надсилання даних відбувається з використання автоенкодеру.

Лістинг 5. Клієнт ущільнює та надсилає дані

```
data_type = "img"
w, h = 30,40
model = get_model(f"model_enc_{data_type}_{w}_{h}.json",
                  f"vae_enc_{data_type}_{w}_{h}.h5")
generator = VideoGenerator(list(range(0,100000)))
i = 0
while True:
    try:
        frame = generator[i][0]
        code = model.predict(frame)
        footage_socket.send(code)
        i += 1

    except KeyboardInterrupt:
        break
```

Лістинг 6. Сервер приймає та відновлює дані

```
import cv2
import numpy as np

counter = 0
data_type = "video"
t, w, h = 1, 30, 40
model = get_model(f"model_dec_{data_type}_{w}_{h}.json",
                  f"vae_dec_{data_type}_{w}_{h}.h5")
dim = int(t*60*24*w*h/20)
start = dt.now()

while (dt.now() - start)>60:
    frame = footage_socket.recv()
    npimg = np.frombuffer(frame, dtype=np.float32)
    source = model.predict(npimg.reshape(-1, dim))
    cv2.imshow("Stream", source[0])
    cv2.waitKey(1)
    counter += 1
cv2.destroyAllWindows()
```

У наведених лістингах 5 та 6 виконано налаштування на передачу відео довжиною в одну хвилину та розміром кадру 30x40. Для відтворення обрано перший кадр.

Другий випадок, що перевіряється, (див. Лістинг 7, 8) – передача даних звичайним способом: без ущільнення. Тут також застосовується генератор даних на стороні клієнта, а сервер засікає хвилину на прийняття та відтворення даних.

Лістинг 7. Клієнт надсилає дані

```
i = 0
generator = TextGenerator(list(range(0,100000)))
while True:
    try:
        frame = generator[i][0]
        footage_socket.send(frame)
        i += 1
    except KeyboardInterrupt:
          break
```

Лістинг 8. Сервер приймає дані

```
import cv2
import numpy as np

counter = 0
```

```
    start = dt.now()

    while (dt.now() - start)>60:
        frame = footage_socket.recv()
        npimg = np.frombuffer(frame, dtype=np.uint8).reshape((h,
w, -1))
        cv2.imshow("Stream", npimg)
        cv2.waitKey(1)
        counter += 1
    cv2.destroyAllWindows()
```

Після завершення роботи програми пеервіряється значення змінної counter та зберігається для подальшого аналізу результатів.

4.3. Порівняння розробленого методу з існуючим

Для перевірки роботи методу з даними різних розмірів замірялись наступні показники:

- кількість переданих файлів за хвилину;
- навантаження на мережу.

При перевірці було протестовано роботу методу на різних розмірах даних. Розмір даних визначався як кількість елементів, що формують тензор представлення даних. Наприклад:

- ЕКГ – одномірний тензор на 500 елементів;
- Зображення MNIST – двовимірний тензор 28*28 = 784 елементи;
- Зображення MNIST у кольоровому форматі – тривимірний тензор 28*28*3 = 2352 елементи;
- Відео, згенеровані з зображень MNIST – тривимірний тензор 28*28*24*10 = 188160;
- Текст – представлення одного слова – одновимірний тензор на 5000 елементів.

Розміри даних додатково змінювались шляхом збільшення та зменшення масштабу у 2 та 4 рази.

Швидкість роботи розробленого методу з текстом у даній роботі не перевірялась через виявлення недоцільності використання даного методу для текстових даних. Це пояснюється тим, що одна літера при передачі по

мережі без ущільнення займає від 1 до 4 байтів. Тобто одне слово може займати до 60 байтів. А представлення одного слова для нейронної мережі – це вектор на 5000 елементів, що займає приблизно 20000 байтів на сучасних процесорах. Очевидно, що при ущільненні даних навіть у 100 разів представлення одного слова при передачі по мережі не стане меншим, ніж 60.

Перевірка методу відбувалась у локальній мережі, що не була навантажена передачею інших даних. Заміри обох методів проводились в однакових умавах та тих самих пристроях. Локальна мережа була створена роутером.

Перевірка передачі інших даних показала результати, що наведені на рис. 4.1.

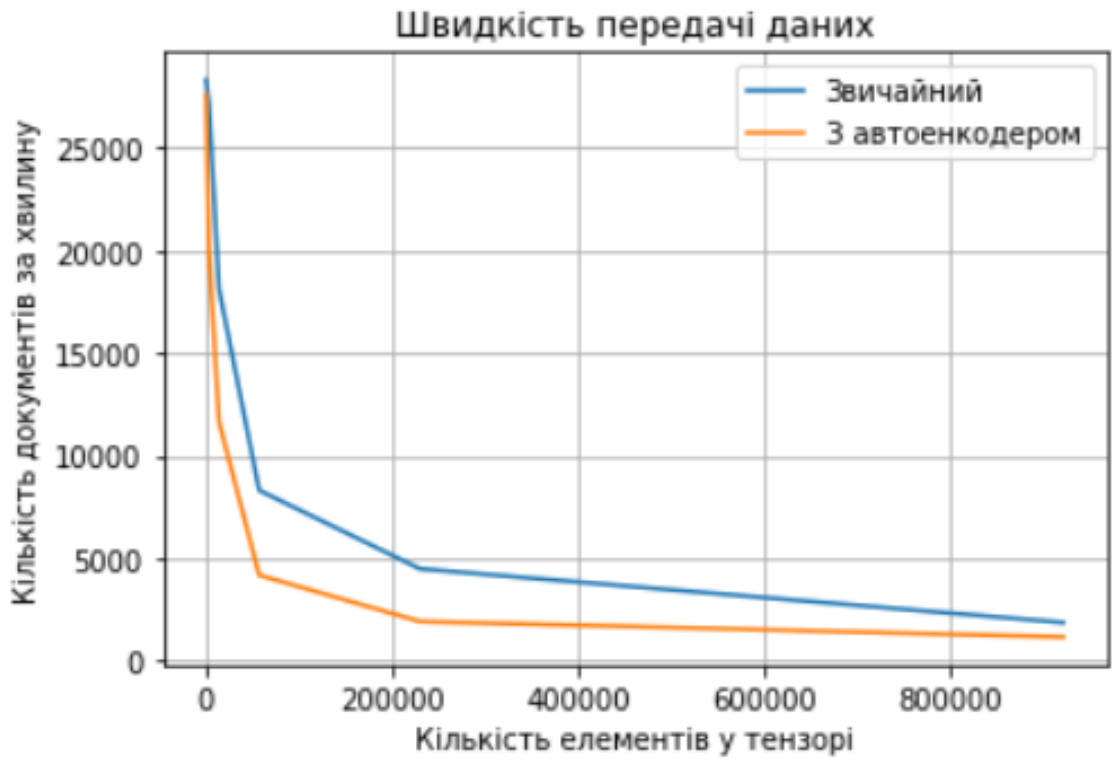


Рис. 4.1. Графік залежності кількості переданих файлів по мережі від їх розміру

Як видно на графіку, що наведено на рис. 4.1, передача даних з автоенкодером показує гіршу швидкість, ніж звичайна передача даних. Це пояснюється витратами часу на ущільнення даних. Однак, навіть при передачі файлів з роздільною здатністю 480*640, показує швидкість, достатню для транслювання даних у режимі реального часу.

Навантаження мережі замірювалось на пристрої, що приймав дані. Одиницею вимірювання є байт на секунду (В/сек). Заміри проводились за допомогою вбудованого програмного забезпечення «Монітор ресурсів» в операційній системі Windows 10. До уваги бралось тільки навантаження на прийняття даних, створене на обраному мережевому порту (на лістингах 2 та 3 видно, що це порт 5555). Результати перевірки навантаженості мережі наведено на рис. 4.2.

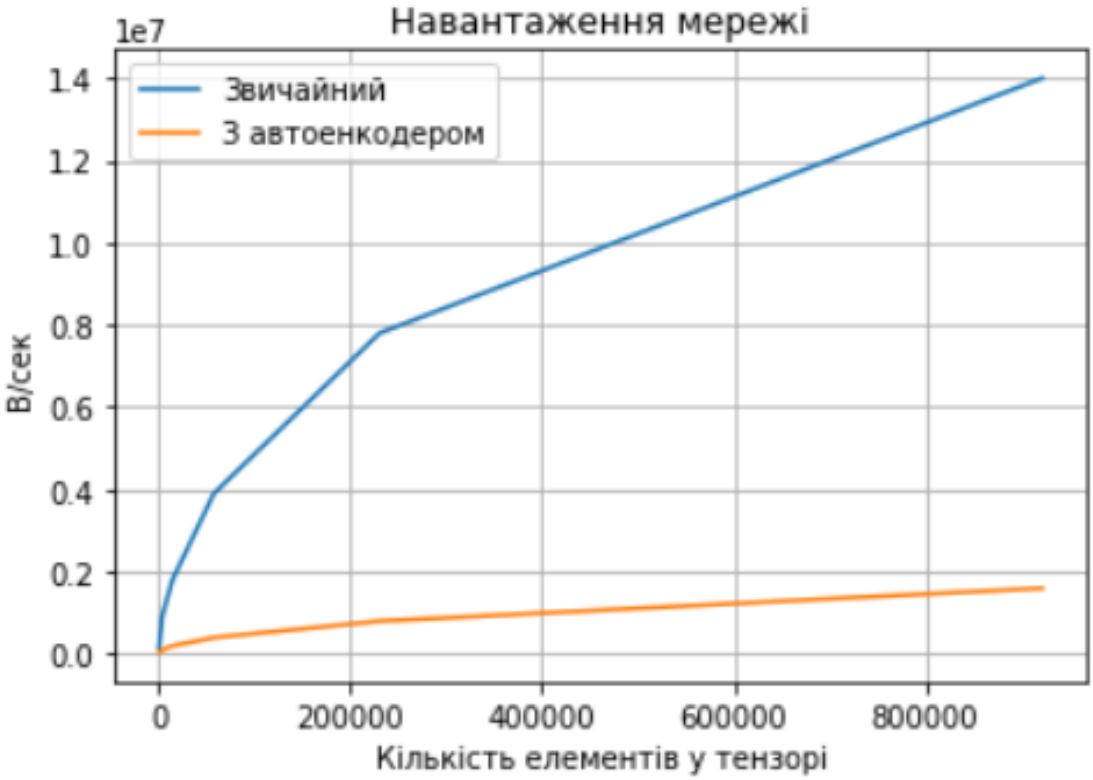


Рис. 4.2. Графік залежності навантаженості мережі на прийняття від розміру файлів

Як видно з графіку, що наведено на рис. 4.2, навантаження мережі при використанні автоенкодеру зменшується у 8.75 разів, що у високонавантажених системах може зіграти вирішальну роль.

4.4. Висновки

У даному розділі розроблено програмний код для тестування розробленого методу та наведено результати тестування.

Перевірка розробленого методу здійснювалась шляхом передачі даних між двома кіцевими пристроями по протоколу TCP. При передачі файлів з різним розміром проводилось замірювання швидкості передачі даних та навантаженості мережі.

У результаті проведеного дослідження виявлено, що метод передачі даних за допомогою автоенкодера пряцює в 1.8 рази повільніше та зменшує навантаження мережі бульше ніж у 8 разів. Сповільнення швидкості передачі даних буде частково скомпенсовано великою пропускною здатністю мережі, хоча навіть така швидкість є допустимою для транслювання даних у режимі реального часу.

ВИСНОВКИ

У даній роботі розроблено метод передачі даних за допомогою автоенкодера, що зменшує навантаження мережі. Використання даного методу дозволяє перенести навантаження з мережі на кінцеві пристрої.

Під час виконання даної роботи проведено аналіз існуючих мережевих протоколів; розглянуто особливості побудови нейронних мереж для різних типів даних. Розроблено метод ущільнення даних на кінцевих пристроях мережі за допомогою автоенкодера. Проведено аналіз інструментів та технологій для проведення тестування розробленого методу.

Результати перевірки розробленого методу показали, що він працює повільніше, ніж звичайна передача даних приблизно у 1.8 рази але зменшує навантаженість мережі у 8.75 разів.

Недоліком роботи є те, що не вдалось досягти повного відновлення даних після ущільнення. Можливим варіантом вирішення даної проблеми може бути використання іншого типу нейронних мереж для відновлення даних.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Kurose James F. Computer networking: a top-down approach / James F. Kurose, Keith W. Ross., 2012. – 889 с.
2. Miller M. A. Voice over IP technologies: Building the converged network [Text] / Miller M. A., John Wiley & Sons, Inc., 2002.
3. Using the RTSP protocol [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc770781(v=ws.10).
4. Liu T. et al. Implementation of training convolutional neural networks [Text] //arXiv preprint arXiv:1506.01195. – 2015.
5. Meier B. et al. Fully convolutional neural networks for newspaper article segmentation [Text] //2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). – IEEE, 2017. – Т. 1. – С. 414-419.
6. Zabihi M. et al. 1D Convolutional Neural Network Models for Sleep Arousal Detection [Text] //arXiv preprint arXiv:1903.01552. – 2019.
7. Hou R., Chen C., Shah M. An end-to-end 3D convolutional neural network for action detection and segmentation in videos [Text] //arXiv preprint arXiv:1712.01111. – 2017.
8. Doersch C. Tutorial on variational autoencoders [Text] //arXiv preprint arXiv:1606.05908. – 2016.
9. Марк Саммерфилд. Python на практике. — Переклад з англійської. — М.: ДМК Пресс, 2014. – 338 с. – ISBN 978-5-97060-095-5.
10. Paradis E., Claude J., Strimmer K. APE: analyses of phylogenetics and evolution in R language [Text] //Bioinformatics. – 2004. – Т. 20. – №. 2. – С. 289-290.
11. Ihaka R., Gentleman R. R: a language for data analysis and graphics [Text] //Journal of computational and graphical statistics. – 1996. – Т. 5. – №. 3. – С. 299-314.

12. PyTorch [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://ai.facebook.com/tools/pytorch/
13. Chollet F. Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek. – MITP-Verlags GmbH & Co. KG, 2018.
14. Jin W., Barzilay R., Jaakkola T. Junction tree variational autoencoder for molecular graph generation [Text] //arXiv preprint arXiv:1802.04364. – 2018.

Метод передачі даних за допомогою нейронної мережі

Студентка групи КП-51
Граділь Анастасія Валеріївна
Керівник: к.т.н., доцент Онай Микола Володимирович

Актуальність теми

Великий об’єм даних, що передаються через мережу є причиною щорічного збільшення навантаження.

Майже усі сервіси ставлять обмеження на розмір файлів для передачі та зберігання.

Тому ущільнення данних, з швидким відновленням вирішить проблему багатьох сервісів.

Завдання

- Розробити метод передачі даних по мережі, що забезпечить зменшення навантаження на мережу при передачі даних, порівняно з існуючими методами.
- Розробити модель для тестування запропонованого методу.

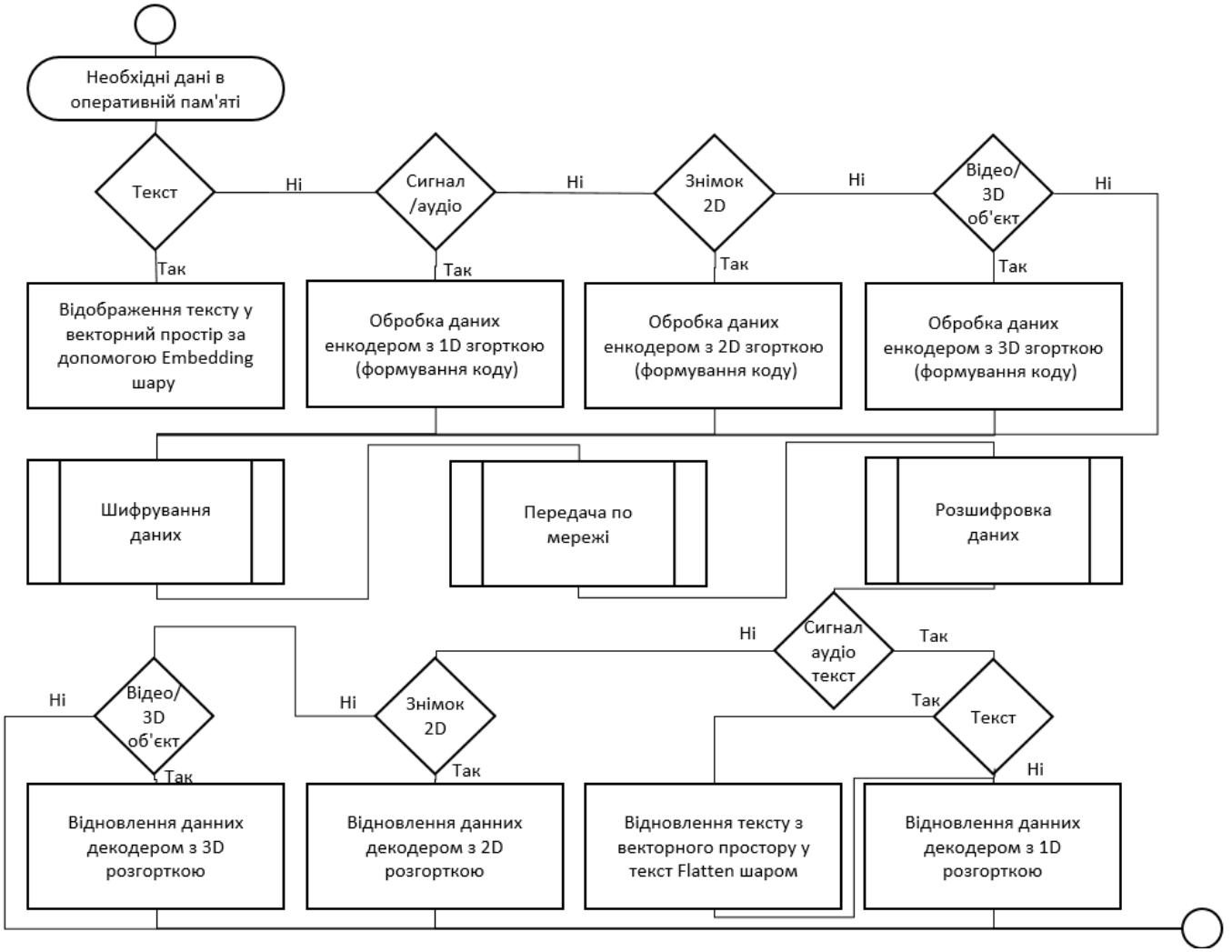
Предмет дослідження

Методи передачі даних з попереднім ущільненням за допомогою автоенкодерів та відновлення даних на основі мережевого протоколу TCP.

Об’єкт дослідження

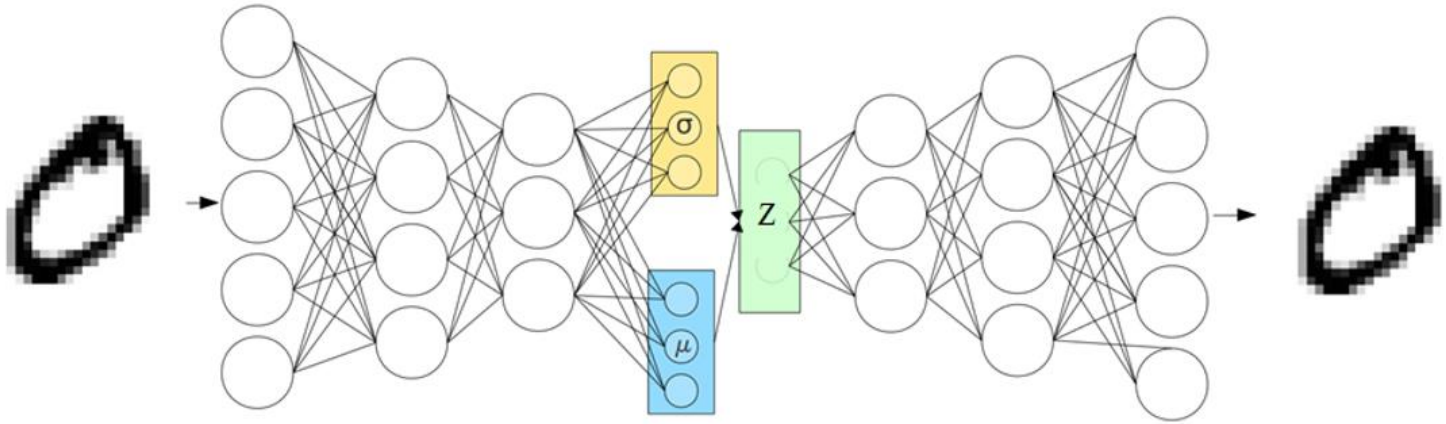
Процес ущільнення та відновлення текстових і мультимедійних даних за допомогою відповідних автоенкодерів в залежності від їх розміру.

Метод передачі даних за допомогою нейронної мережі



Автоенкодери

Автоенкодери складаються з двох частин: енкодера *g* і декодера *f*.
Енкодер переводить вхідний сигнал в його образ (код): *h = g (x),*
а декодер відновлює сигнал по його коду: *x = f (h).*



$$Z = \mu + \exp(\sigma) * \varepsilon$$

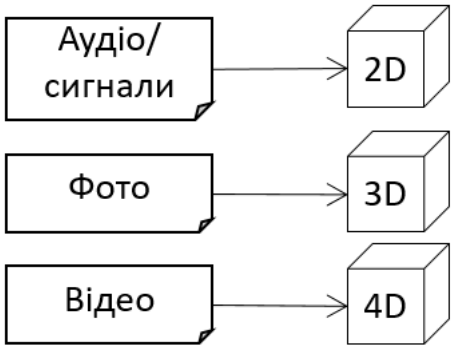
Особливості обробки тексту при використанні запропонованого методу

- Формування коду для кожного слова (вектор на 5000 елементів);
- Відображення кожного слова у точку векторного простору (вектор на 200 елементів);

- Обробка згортковим шаром;
- Формування коду (вектор на 10 елементів).

Обробка мультимедійних даних при використанні запропонованого методу

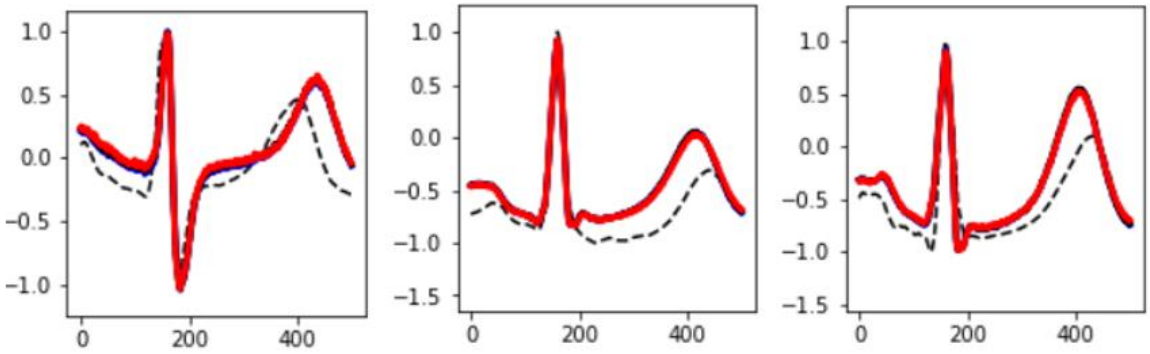
- Представлення даних у форматі тензору;



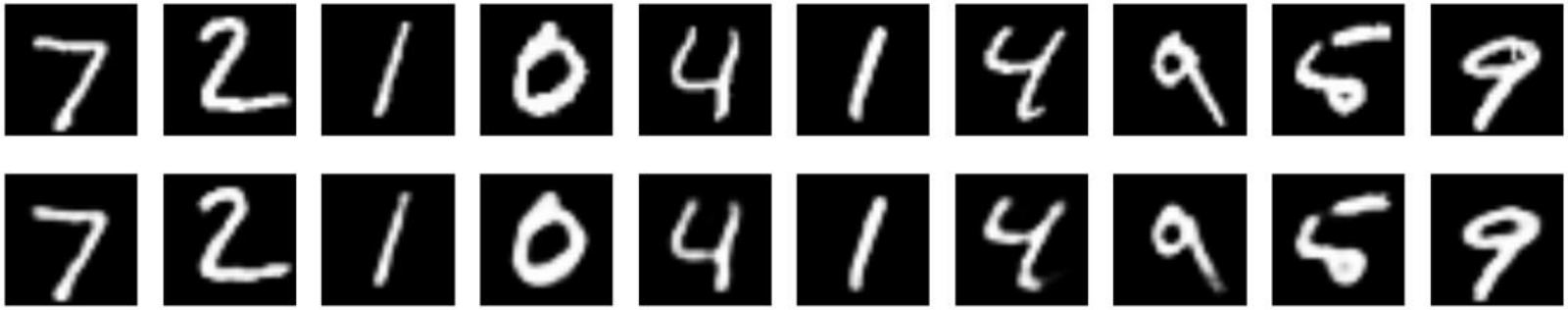
- Обробка згортковим шаром;
- Формування коду.

Відновлення даних

ЕКГ



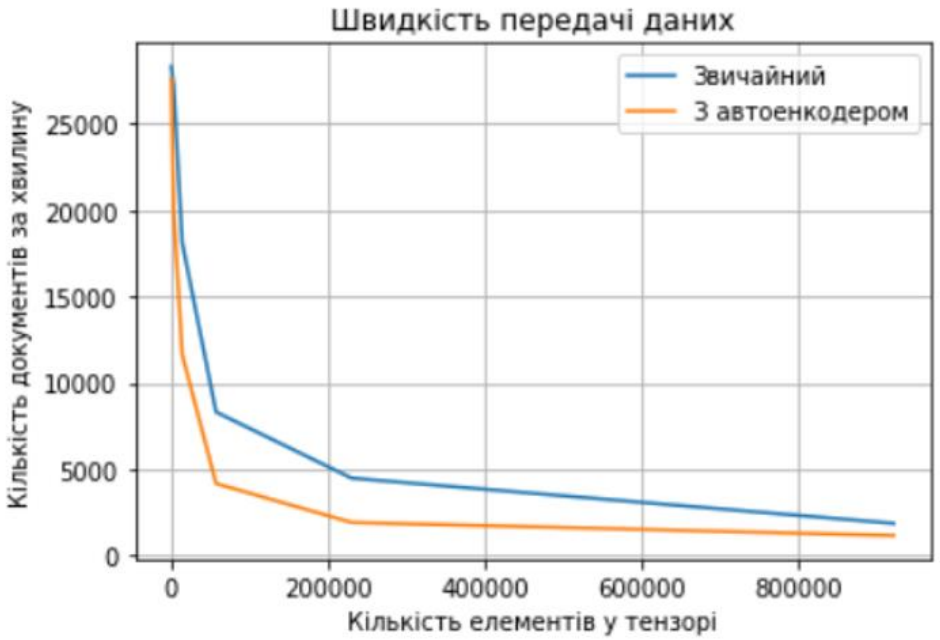
MNIST

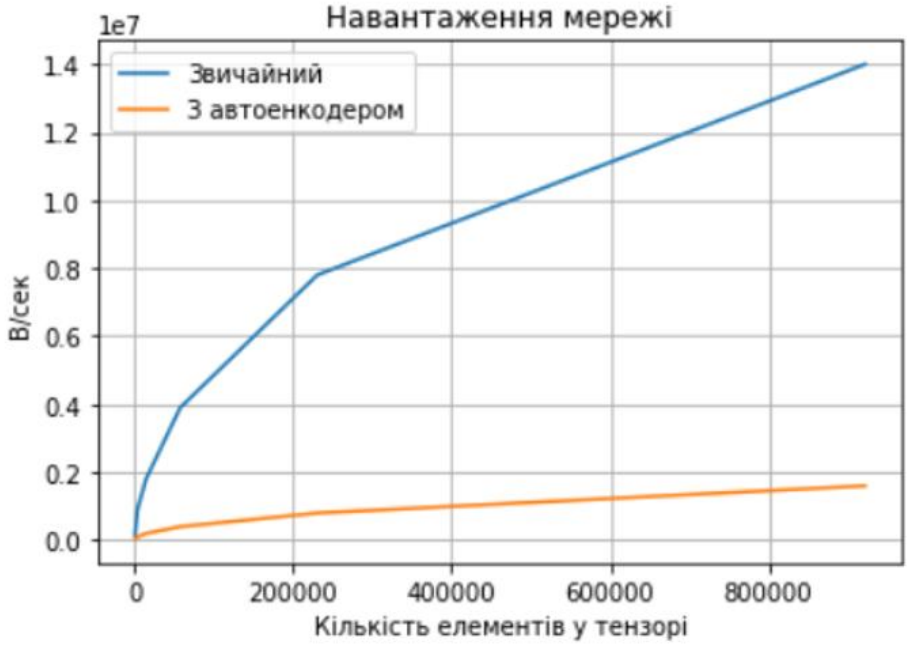


Тексти з Вікіпедії

«Сколько еще мы будем терпеть эту несправедливость?»
Сколько еще мы будем переносить эту несправедливость

Результати досліджень





Наукова новизна

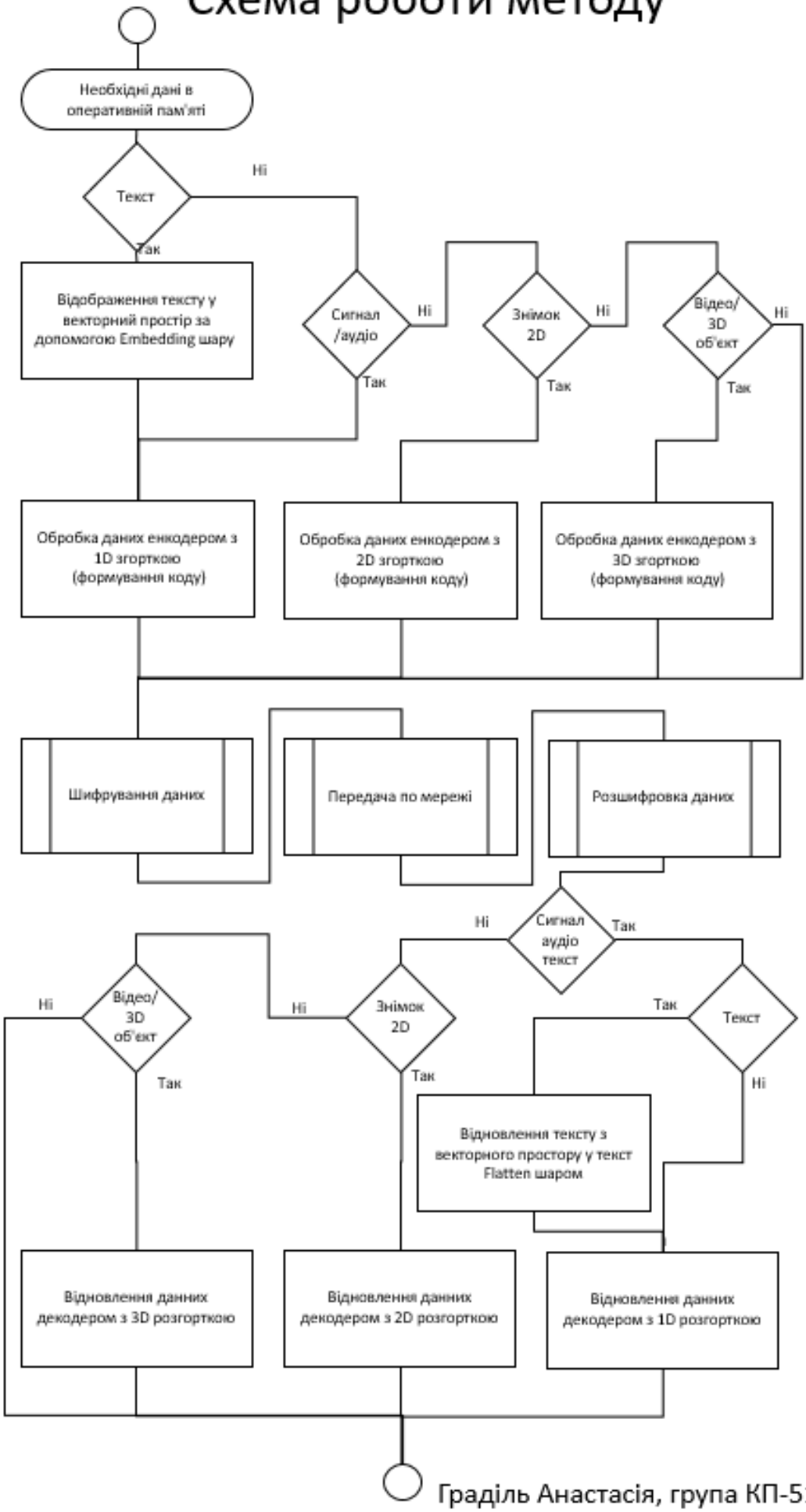
Запропоновано метод передачі даних з використанням нейронної мережі, який використовує автоенкодери для ущільнення даних при передачі даних мережею та за рахунок цього зменшує навантаження мережі у 8.75 разів порівняно з існуючим методом.

Висновки

- Швидкість методу передачі даних з використанням автоенкодерів було перевірено на локальній мережі. Даний метод забезпечив зменшення навантеженості мережі у 8.75 разів та зменшив швидкість передачі даних майже у 1.8 рази, порівняно з існуючим.
- Не вдалось досягти повного відновлення даних. Цю проблему можна вирішити збільшивши розмір вектору кодування або використати інший тип нейронної мережі (GAN) для відновлення даних.
- Метод не доцільно використовувати для передачі текстових даних.

Дякую за увагу!

Схема роботи методу

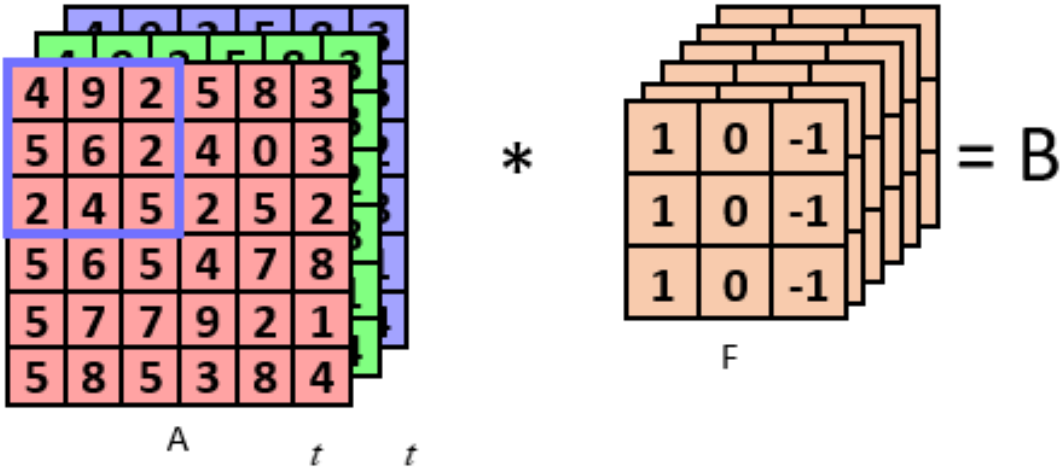


Граділь Анастасія, група КП-51

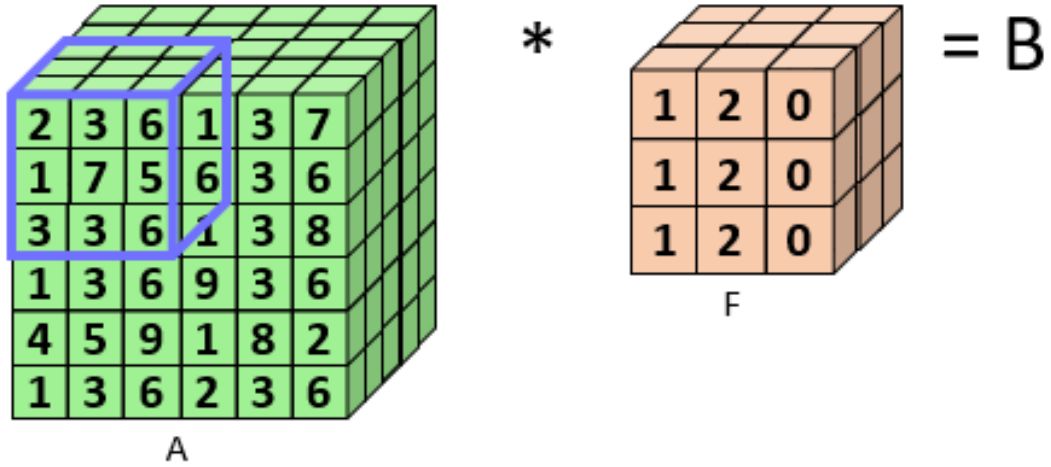
Принцип роботи згорткового шару

$$t = \frac{s_F - 1}{2}$$

$$B_i = \sum_{a=-t}^{t} A_{i+a} * F_{a+t+1}$$



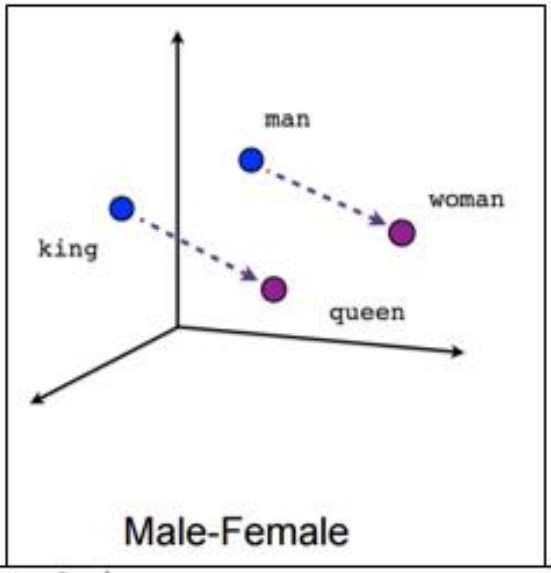
$$B_{ij} = \sum_{a=-t}^{t} \sum_{b=-t}^{t} A_{i+a,j+b} * F_{a+t+1,b+t+1}$$



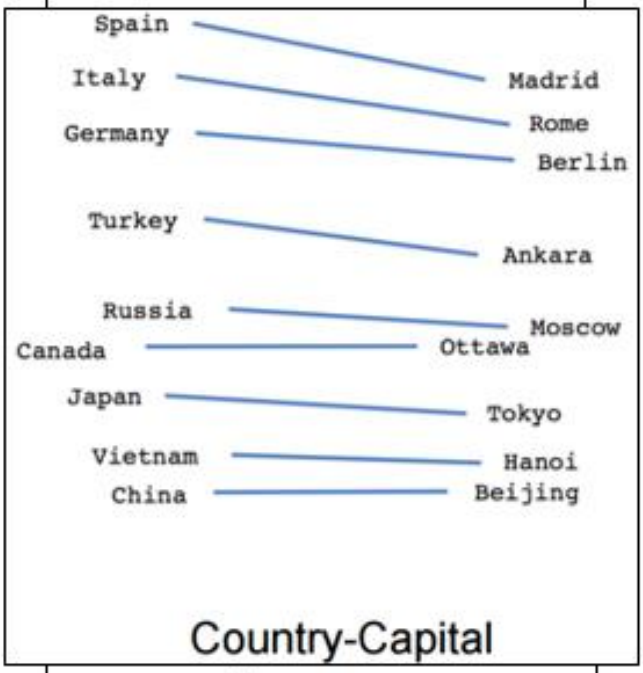
$$B_{ijk} = \sum_{a=-t}^{t} \sum_{b=-t}^{t} \sum_{c=-t}^{t} A_{i+a,j+b,k+c} * F_{a+t+1,b+t+1,c+t+1}$$

Граділь Анастасія, група КП-51

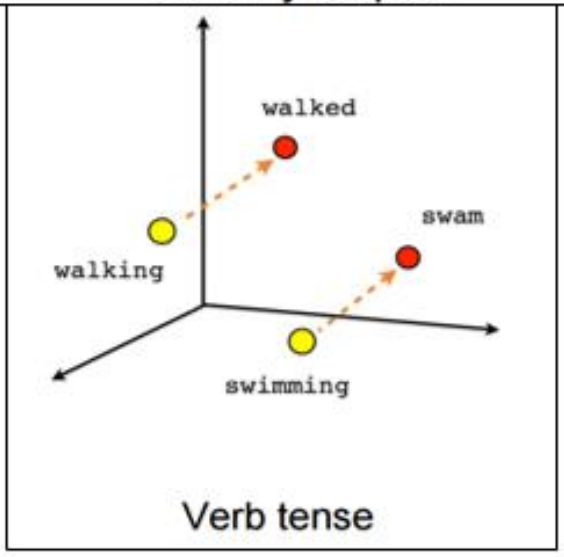
Приклади векторного простору слів



Male-Female



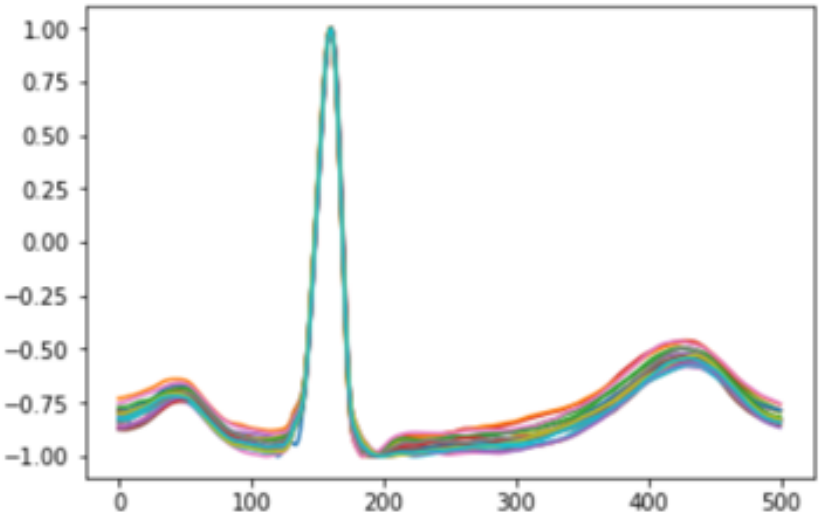
Country-Capital

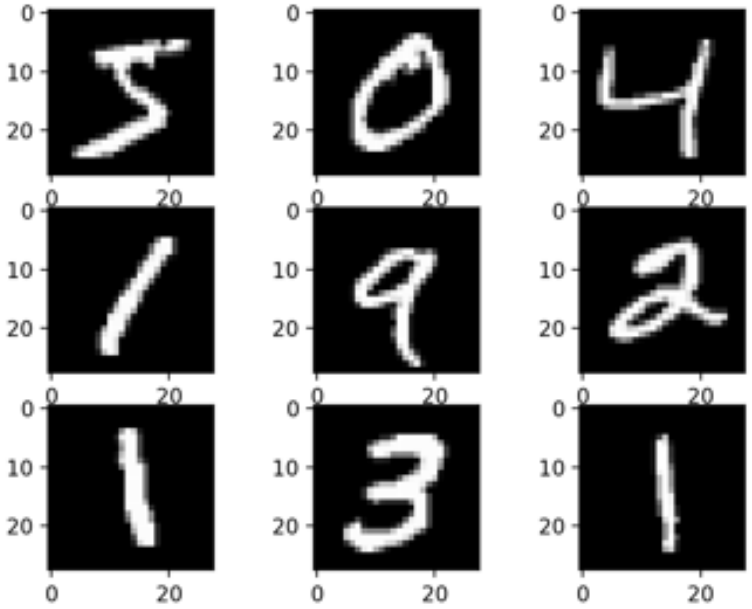


Verb tense

Граділь Анастасія, група КП-51

Приклади даних





Граділь Анастасія, група КП-51