КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

Центр науково-дослідницьких розробок ТОВ "Самсунг Електронікс Україна Компані"

Дипломна робота

на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки

на тему:

НОВІТНІ АЛГОРИТМИ ОПТИЧНОГО РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ

Виконав студент 2 курсу магістратури
Краснорyцький Єгор Олегович _____

Науковий керівник:
професор, доктор фіз.-мат наук
Терещенко Василь Миколайович _____

Консультант (ментор ЦНДДКР
ТОВ "Самсунг Електронікс
Україна Компані"):
доцент, канд. техн. наук
Радивоненко Ольга Сергіївна _____

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

Студент: _____

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики

«____» _____ 2021 р.,
протокол № ___
Завідувач кафедри математичної інформатики
Терещенко Василь Миколайович _____

Київ – 2021

АНОТАЦІЯ

Робота складається з 59 сторінок, містить 30 зображень, 1 таблицю та 6 додатків. Для написання роботи були використані 26 джерел, наведених у бібліографії.

Ключові слова: *ОПТИЧНЕ РОЗПІЗНАВАННЯ, РУКОПИСНИЙ ТЕКСТ, НЕЙРОННА МЕРЕЖА, MDLSTM, CTC, ПОПЕРЕДНЯ ОБРОБКА.*

Робота досліджує можливості використання передової архітектури рекурентних нейронних мереж MDLSTM для задачі оптичного розпізнавання рукописного тексту і застосування методів попередньої обробки тренувальних даних з метою покращення ефективності результуючої моделі. Запропоновані в роботі методи попередньої обробки були здатні покращити результативність моделі на тестових даних.

Зміст

Вступ

Розпізнавання рукописного тексту є однією з ключових задач в галузі машинного навчання та комп'ютерного зору. Основними класами задач цієї галузі є розпізнавання рукописного введення за допомогою певного інтерфейсу користувача або онлайн розпізнавання, та розпізнавання рукописного тексту з фотографії чи відсканованого зображення або оффлайн розпізнавання. Ці задачі працюють з різними типами вхідних даних то потребують різних підходів для їх розв'язання.

Метою оффлайн розпізнавання є перетворення рукописного тексту у вигляді зображення на текст у формі зрозумілій певному електронному обчислювальному пристрою чи комп'ютеру[1]. Отриманий таким чином текст може бути як кінцевим результатом обробки зображення, так і проміжною складовою для розв'язання інших задач, таких як класифікація документів.

Зображення 1 - Результат оптичного розпізнавання тексту

В цій роботі були розглянуті основні ідеї та принципи що застосовуються для впровадження програмних рішень націлених на розв'язання цієї задачі, описані деталі реалізації одного з передових підходів на базі рекурентної нейронної мережі MDLSTM та запропоновані ефективні способи аугментації вхідних даних спрямовані на покращення отриманих результатів.

Актуальність роботи полягає в широкій комерційній потребі розробки програмного забезпечення для розпізнавання рукописного тексту, як в контексті обробки офіційних документів, так і для загального користувацького ринку.

Об'єктом дослідження є зображення що містять типові приклади рукописного тексту у вигляді англомовних слів та речень.

Предметом дослідження є алгоритми, процеси та моделі нейронних мереж за допомогою яких виконується розпізнавання рукописного тексту.

Метою роботи є опис та тестування передової моделі оптичного розпізнавання тексту, та розробка засобів аугментації вхідних зображень з метою покращення результатів розпізнавання.

Поставлена мета потребує:

- Вивчення існуючих алгоритмів розпізнавання рукописного тексту.
- Дослідження метрик для визначення якості розпізнавання тексту
- Розгляд існуючих відкритих датасетів що містять приклади англомовного рукописного тексту.
- Аналіз та порівняння передових нейронних моделей та засобів аугментації вхідних зображень, що можуть бути використані для покращення якості розпізнавання.

Матеріалом дослідження є зображення з відкритого англомовного датасету IAM Handwriting Database[2].

Можливі сфери застосування. Модель розглянута в цій роботі в поєднанні з програмним забезпеченням для виявлення тексту може бути використана для загального розпізнавання рядків англомовного рукописного тексту.

Огляд підходів до оптичного розпізнавання тексту

1.1. Базова інформація

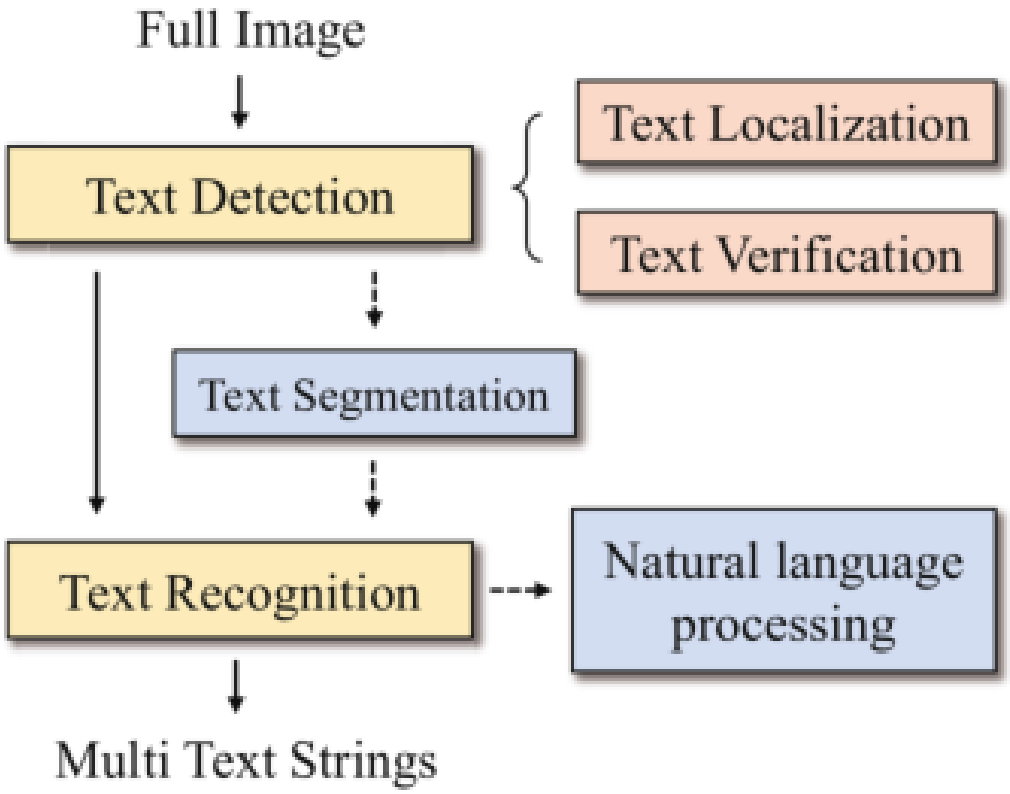
Типові системи та програмне забезпечення для розпізнавання рукописного тексту в реальних умовах окрім власне перетворення зображення рукописного тексту у його машинну інтерпретацію передбачають також знаходження і виділення цього тексту з вхідного зображення, що може містити додаткові деталі окрім самого тектсу, та розбиття багаторядкового тексту на компоненти, зрозумілі розпізнавальній системі.

Але навіть за ідеальної умови, де на вхід розпізнавальної моделі поступають виділені окремі текстові рядки, задача розпізнавання рукописного тексту не є тривіальною, та має бути розбита на підзадачі для знаходження ефективного розв'язку. Розглянемо типову процедуру розпізнавання рукописного тексту описану в джерелі[3] з метою виділення основних її основних складових частин:

1) Попередня обробка вхідних зображень.
2) Сегментація отриманого зображення.
3) Перетворення даних та виділення ознак.
4) Тренування та розпізнавання.
5) Постеріорна обробка отриманих результатів.

1.2. Виявлення та сегментація тексту

Вхідні зображення що використовуються для навчання моделі розпізнавання тексту часто потребують додаткової обробки з метою знаходження та виокремлення тексту на зображенні. Оскільки більшість загальновживаних моделей розпізнавання працює з виокремленими текстовими рядками, перед проведенням тренування необхідно отримати ці виокремлені рядки з зображення, яке може містити цілий документ чи певну його частину.



Зображення 2 - Приклад структури системи оптичного розпізнавання тексту.

Система що має на меті забезпечити повну процедуру розпізнавання тексту з зображення має включати такі компоненти[9]:

- **Локалізація тексту:** Метою локалізації тексту є знаходження на зображенні регіонів, що потенційно можуть містити текст, та максимально точне їх виділення, з найменшою можливою кількістю фону.
- **Верифікація тексту:** Метою верифікації є фільтрування хибних позитивних спрацьовувань на етапі локалізації.

- **Сегментація тексту:** Сегментація включає в себе розбиття текстових блоків на рядки, та в окремих випадках розділення частин рядка, слів, чи символів. Задача сегментації є однією з найскладніших підзадач оптичного розпізнавання, помилки на цьому етапі негативно впливають на подальшу роботу і навчання моделі.

- **Розпізнавання тексту:** Задачею розпізнавання тексту є перетворення виокремленого зображення текстового рядка у відповідний текст у машинній формі. Традиційні підходи розпізнавання базувалися на спеціально розроблених шаблонах і ознаках, але більшість сучасних рішень використовують нейронні моделі, які не потребують ручної розробки ознак.

- **Обробка отриманого тексту:** Обробка тексту або Natural Language Processing полягає у використанні мовних моделей заснованих на n-грамах слів чи символів, з метою виправлення помилок розпізнавання шляхом надання переваги більш частим мовним конструкціям в процесі розпізнавання.

Традиційні рішення для оптичного розпізнавання тексту часто вимагали додаткової сегментації з метою виявлення окремих слів та символів. Сучасні рішення використовують методики що дозволяють проводити розпізнавання без попередньої сегментації символів, такі як Connectionist Temporal Classification або CTC. Сучасні моделі без сегментації символів мають значні переваги, і показують кращі результати у переважній більшості випадків.

Зображення 3 - Приклад сегментації символів.

1.3. Попередня обробка

Вхідні дані, залежно від їх типу та умов за яких вони були зібрані, проходять ряд етапів попередньої обробки, щоб зробити їх придатними для використання на подальших етапах аналізу та навчання моделі. Попередня обробка має на меті отримання даних, які можуть бути легко опрацьовані системами оптичного розпізнавання. Основними цілями попередньої обробки є:

1) Усунення випадкових шумів та артефактів.
2) Нормалізація даних.
3) Компресія даних з метою видалення зайвої інформації.

Для досягнення цих цілей на стадіїї попередньої обробки застосовуються такі методи:

1.3.1. Усунення шуму

Шум спричинений пристроєм оптичного сканування або письмовим інструментом спричиняє розриви з’єднаних компонентів, нерівності та прогалини в лініях, заповнені петлі тощо. Спотворення можуть змінюватися відносно положе-

ння та включають в себе, заокруглення кутів, розширення контурів та ерозію. До початку оптичного розпізнавання необхідно усунути ці недоліки. Більшість доступних методів зменшення шуму можна класифікувати на три основні групи.

a) Фільтрування: Спрямоване на усунення шуму та зменшення кількості зайвих точок, які зазвичай виникають через нерівну запису та / або низьку роздільну здатність пристрою збору даних. Для цього можуть бути розроблені різні спеціалізовані фільтри. Основна ідея полягає в тому, щоб модифікувати зображення за допомогою заздалегідь визначеної маски, що являє собою функцію від значень сусідніх пікселів. Фільтри можуть виконувати роль згладжування, збільшення різкості, відкидання за пороговим значенням, видалення текстурованого або кольорового фону та корекції гами і контрасту.

b) Морфологічні операції: Основна ідея полягає у обробці зображення спеціально розробленими фільтрами, що виконують певні логічні операції. Прикладами таких операцій є з'єднання ламаних штрихів, розкладання з'єднаних штрихів, згладжування контурів, усунення точок-викидів, зменшення товщини символів та усунення зайвих пробілів. Такі операції можуть бути використаеі для усунення недоліків зображення пов'язаних з якістю паперу, чорнил та незначних помилок в процесі написання.

c) Моделювання шуму: Шум який має відомі властивості, наприклад шум викликаний особливостями зчитувального приладу можна усунути, якщо існує його статистична модель.

1.3.2. Нормалізація даних

Метою методів нормалізації даних є усунення варіацій певних властивостей, таких як розмір і кут нахилу, та отримання стандартизованих даних. Базові методи нормалізації включають в себе:

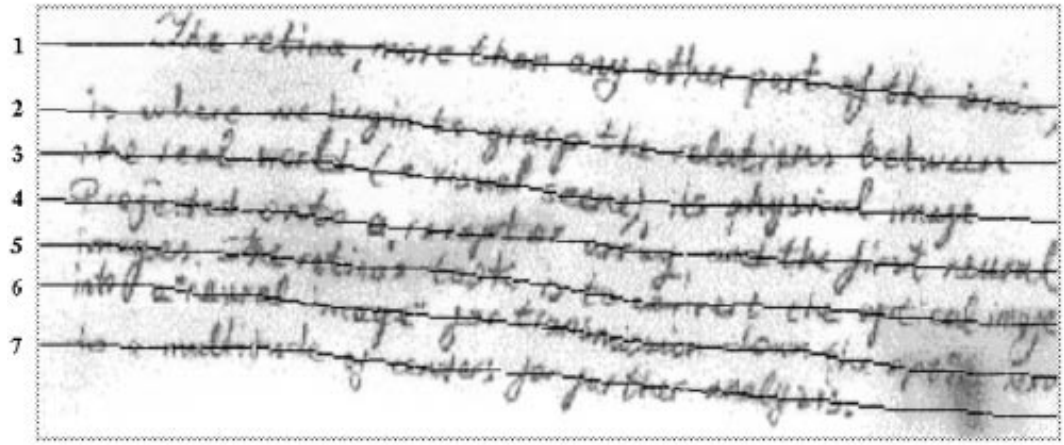
a) Виділення базової лінії та усунення вертикального зсуву: Через неточності в процесі сканування та погану якість написання, довгі рядки можуть зсунутися чи вигнутися в межах зображення(див. Зображення 4). Це може зашкодити якості роботи подальших стадій розпізнавання, і тому повинно бути виявлене та виправлене. Крім того, деякі символи розрізняються відповідно до їх положення по висоті відносно решти тексту. Методи виділення

базової лінії включають використання проекційного профілю зображення, що є типом кластеризації за найближчими сусідами, метод крос-кореляції між рядками та за допомогою перетворення Хафа[4]. Також можливе використання нейронної мережі для виділення базової лінії в особливо складних випадках. Після виявлення перекосу символ або слово, обертається або розтягується до тих пір, поки базова лінія не стане горизонтальною після чого отримане зображення зсувається так щоб розмістити її по центру.

b) Нормалізація нахилу: Однією з основних відмінностей різних почерків є кут нахилу між найдовшим штрихом у слові та вертикальною лінією. Нормалізація нахилу використовується для усунення розбіжностей між нахилами окремих символів. Найбільш поширеним методом визначення нахилу тексту є обчислення середнього кута нахилу символів, близьких до вертикальних. Вертикальні лінійні елементи отримуються з символів за допомогою пари одновимірних (1-D) фільтрів. Координати початкової та кінцевої точок кожного елемента лінії визначають кут нахилу. В іншому дослідженні використовується підхід, при якому ство для числа кутів, віддалених від вертикального напрямку. Також можливе використання перетворення Хафа шляхом сканування зліва направо на зображення і обчислення проекцій у певних заздалегідь визначених напрямках. Нахил, який зустрічається найчастіше приймається як значення кута нахилу тексту. З іншого боку, в деяких дослідженнях системи розпізнавання не використовують корекцію нахилу і компенсують її на етапі навчання.

c) Нормалізація розміру: Використовується для приведення розміру символів до одного діапазону значень. Різні методи оптичного розпізнавання тексту можуть застосовувати як горизонтальну, так і вертикальну нормалізацію за розміром. Також можливе розділення символів на зони, і масштабування цих зон окремо. З іншого боку, для розпізнавання слів, внутрішньокласові відмінності в їх довжині є важливою інформацією при розпізнаванні, тому в цьому випадку горизонтальна нормалізація не використовується, або виконується масштабування за коефіцієнтом визначеним при вертикальній нормалізації.

d) Згладжування контурів: Усуває помилки, пов’язані з нерівномірним рухом під час написання. Як правило, це зменшує кількість точок, необхідних для



Зображення 4 - Приклад виявлення базової лінії тексту

представлення отриманого тексту, що збільшує ефективність інших засобів попередньої обробки.

1.3.3. Компресія даних з метою видалення зайвої інформації

Вхідні дані містять більшу кількість інформації, ніж є необхідним для їх розпізнавання та відтворення. Ця інформація може негативно вплинути на якість розпізнавання, і призвести до утворення небажаних залежностей що можуть негативно вплинути на здатність моделі до узагальнення на невідомих даних.

a) Бінаризація: Для збільшення швидкості обробки та кращого виділення тексту порівняно з фоном, бажано представляти сірі або кольорові зображення як двійкові, обравши певне порогове значення, або застосувавши більш складний алгоритм бінаризації. Існують дві категорії порогових значень: глобальні та місцеві. Глобальне порогове значення визначає одне порогове значення для всього зображення, яке часто базується на оцінці рівня фону за допомогою гістограми інтенсивності на зображенні. Місцеве (адаптивне) порогове значення використовує різні значення для кожного пікселя відповідно до інформації про його окіл.

b) Скелетонізація: Хоча цей підхід значно зменшує кількість ненульових даних у зображенні, отриманий результат краще відображає геометричну форму символів. Найпростішими методами скелетонізаціїї є попіксельні методи які ітеративно видаляють пікселі, поки не залишиться скелет шириною

один піксель. Вони дуже чутливі до шуму і можуть спотворювати форму вхідного зображення. З іншого боку, непіксельні методи використовують деяку глобальну інформацію під час скелетонізації. Одним з прикладів є визначення медіани зображення відносно його країв.Також можлива скелетонізація за допомогою алгоритмів кластеризації: отримані центри кластерів утворюють кінцеве зображення. Варто звернути увагу, що вищезазначені методи суттєво видозмінюють дані та можуть внести несподівані спотворення у вхідні зображення. Як результат, ці методи можуть призвести до втрати важливої інформації, їх слід застосовувати обережно.

1.4. Висновки до першого розділу

Задача оптичного розпізнавання рукописного тексту полягає у перетворенні вхідного зображення тексту на відповідну послідовність символів. Цей процес складається з таких етапів: попередня обробка зображення, виділення та сегментація тексту, виділення ознак, тренування або розпізнавання, обробка отриманих результатів, наприклад за допомогою мовної моделі.

Методи попередньої обробки даних включають в себе усунення шумів та спотворень спричинених кутом нахилу тексту чи зсуву вздовж рядка, нормалізацію даних за розміром, бінаризацію та скелетонізацію. Їх метою є підготовка зображення для подальшої обробки і зменшення впливу артефактів наявних у тренувальних даних на навчання моделі.

Методи попередньої обробки необхідно обирати враховуючи особливості тренувальних даних та специфіку задачі, необережне використання засобів попередньої обробки може дати негативні результати.

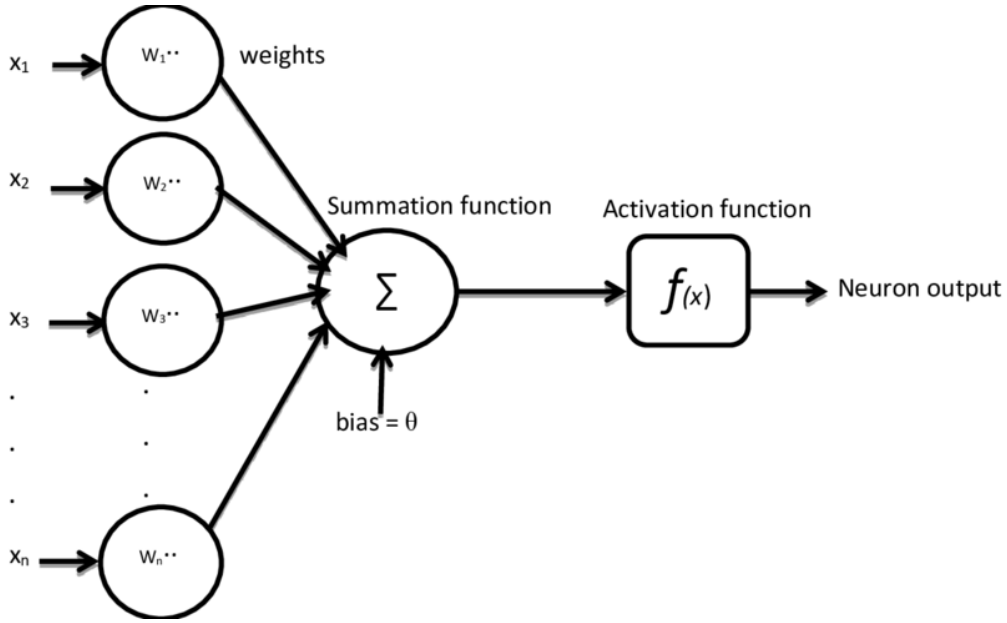
Огляд архітектур нейронних мереж для оптичного розпізнавання тексту

Нейронна мережа[5] - це паралельна розподілена структура для обробки інформації, що складається з нейронів (що здійснюють певні операції з даними отриманими на вході), взаємопов'язаних між собою завдяки односпрямованим зв'язкам. Кожен нейрон має єдиний вихід, який розгалуджується на довільну кількість зв'язків що передають один і той самий сигнал на подальшу обробку. Всі операції, що виконуються нейроном в процесі обробки є локальним: тобто залежать лише від поточних значень вхідних сигналів, що надходять до нейрону, та спеціальних вагових коефіцієнтів, що зберігаються в локальній пам'яті нейрону.

Штучний нейрон є основною функціональною одиницею нейронної мережі, він отримує певну фіксовану кількість вхідних сигналів у вигляді числових значень x_i та здійснює їх обробку шляхом множення кожного з них на відповідний коефіцієнт w_i після чого отримані вхідні дані сумуються після чого до них застосовується функція активації φ] з метою перетворення вихідного сигналу до бажаного вигляду. Отриманий на виході нейрону результат може бути визначений таким чином:

$$y = \varphi(\sum_{i=0}^{n} w_i x_i) \tag{2.1}$$

Найпростішою нейронною моделлю є перцептрон: він являє собою ряд незалежних нейронів, кожен з яких отримує однакові вхідні дані. Завдяки підбору



Зображення 5 - Структура штучного нейрону[6].

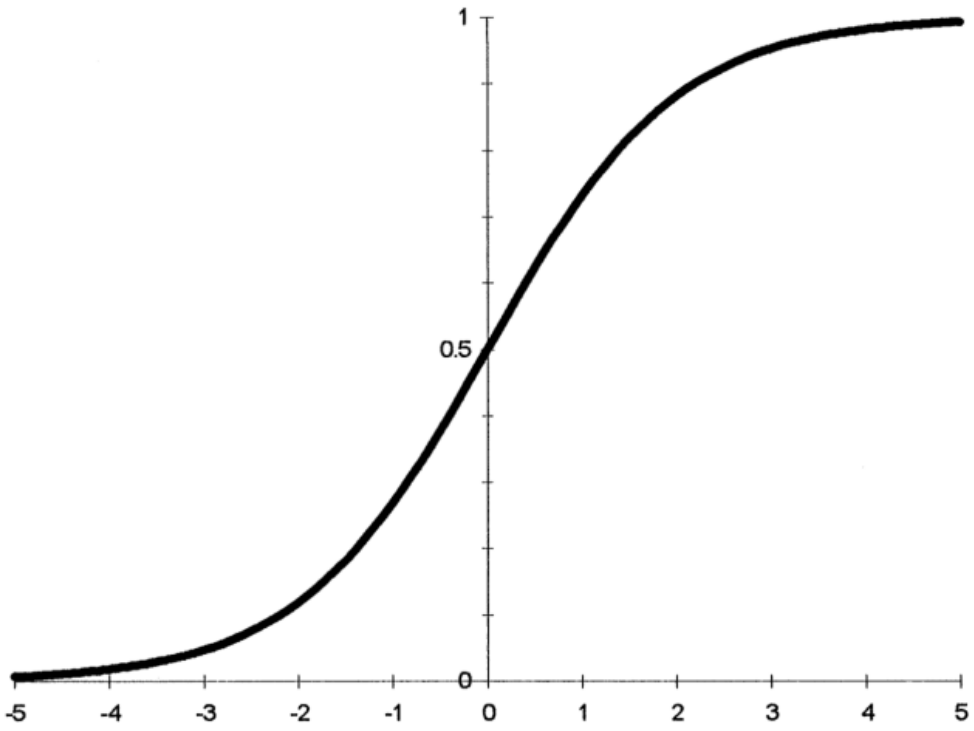
коефіцієнтів w_i для кожного з цих нейронів перцептрон може апроксимувати певну цільову функцію, в процесі підбору цих коефіцієнтів відбувається його навчання. Одним з основних недоліків перцептрона є його нездатність формулювати складні нелінійні залежності. Для вирішення цієї проблеми застосовуються декілька послідовних шарів перцептрона, що стало основою глибоких нейронних мереж.

2.1. Multi-layer perceptron

Багатошаровий перцептрон[7] або MLP аналогічно до звичайного перцептрону складається з системи нейронів, але на відміну від одношарового перцептрону, ці нейрони мають послідовні зв'язки, що дозволяє моделювати нелінійне відображення між вхідним та вихідним вектором. Нейрони з'єднані вагами і виводять сигнал який є сумою всіх входів до цього нейрона, модифікований простою нелінійною функцією активації. Така суперпозиція багатьох нелінійних функцій активації, дозволяє багатошаровому персептрону моделювати нелінійні функції. Без нелінійної функції активації, багатошаровий перцептрон не здатний утворювати нелінійні залежності і є еквівалентним до звичайного перцептрона.

Однією з найпоширеніших функцій активації є логістична функція або си-

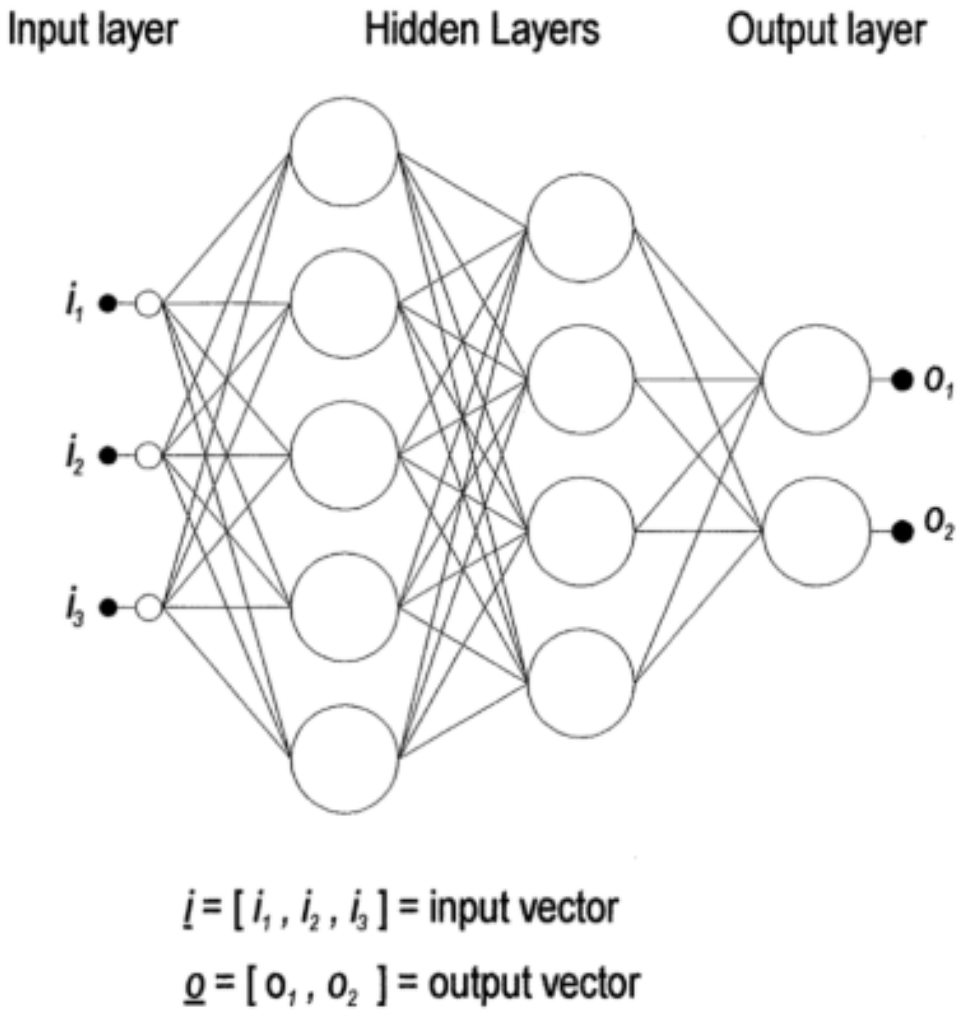
гмоїда(див. Зображення 6) завдяки її легко обчислюваній похідній та здатності масштабувати вихід отриманий з нейрону на проміжку $(0, 1)$.



Зображення 6 - Логістична функція.

Вихід вузла модифікується за допомогою ваги для кожного із з'єднань і подається вперед, на вхід до нейронів наступного шару мережі. Такий процес передбачає послідовну обробку інформації, тому багатошаровий перцептрон також відомий як нейронна мережа прямого поширення або Feedforward Neural Network. Архітектура багатошарового перцептрону має різні варіації, але загалом складається з декількох шарів нейронів. Вхідний шар не виконує ніяких обчислень, і потрібен лише для передачі вектора вхідних даних в мережу. Багатошаровий персептрон може мати один або кілька прихованих шарів і, вихідний шар що повертає результат обчислення. Багатошарові персептрони є повнозв'зними, тобто кожен нейрон є з'єднаним з кожним нейроном на наступному та попередньому рівні.

В багатьох дослідженнях[8] було продемонстровано, що підібравши певний набір ваг та функцій активації, багатошаровий перцептрон може наблизити будь-яку неперервну функцію відображення між вхідним та вихідним векторами. Багатошарові перцептрони мають можливість навчатися за допомогою спеціальних наборів навчальних даних, що містять приклади вхідних даних та очікувані



Зображення 7 - Багатошаровий перцептрон.

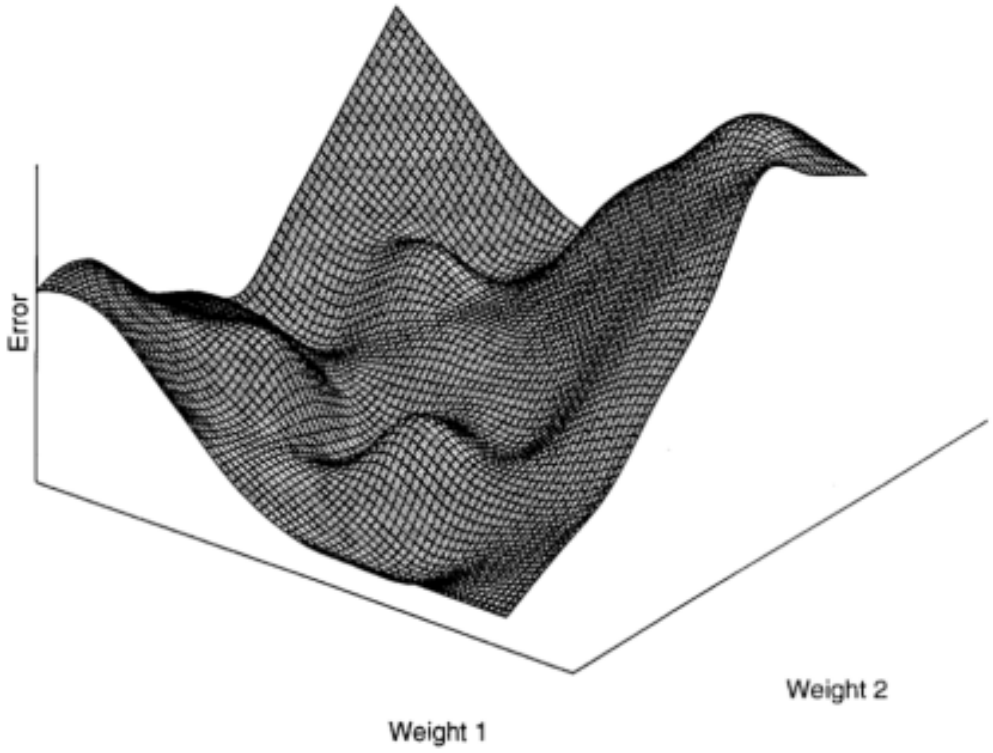
результати. Під час навчання багатошаровий перцептрон багаторазово опрацьовує тренувальні дані, в процесі чого ваги в мережі модифікуються, доки не буде отримано бажаного результату.

Під час навчання, вихід отриманий з перцептрона для певного вхідного вектора може відрізнятися від бажаного результату. Помилка визначається як різниця між бажаним та фактичним результатом, величина цієї помилки визначає як регулюються ваги в мережі, з метою зменшення загальної помилики моделі. Існує багато алгоритмів, за допомогою яких можна тренувати багатошаровий перцептрон. За умови що тренувальні дані є репрезентативними щодо загального випадку, багатошаровий перцептрон здатний давати коректні результати на даних, які він не зустрічав в процесі навчання.

2.2. Метод зворотного поширення помилки

Навчання багатошарового персептрону чи більш складної нейронної моделі - це процедура, за допомогою якої значення окремих ваг визначаються таким чином, щоб цільова функція яку моделює мережа відображалася з максимальною точністю.

При тренуванні на певному зразку даних для будь-якої комбінації ваг може бути визначена помилка мережі для даного випадку. Підбираючи вагові коефіцієнти серед усіх можливих значеннь, можна утворити графік помилок у багатовимірному просторі. Задача навчання нейронної мережі полягає в знаходженні комбінації ваг, яка призводить до найменшої помилки. На практиці перебрати всі комбінації ваг не є можливим, тому потрібен метод, для знаходження мінімальної точки поверхні помилки.



Зображення 8 - Приклад графіку помилки нейронної мережі залежно від ваг в багатовимірному просторі.

Однією з можливих методик є використання алгоритму градієнтного спуску. Навчання за допомогою зворотного поширення використовує цей метод з метою знаходження глобального мінімуму помилки. Алгоритм зворотного поширення є легким в обчисленні, і широко використовується для навчання переважної більшості нейронних моделей.

В процесі навчання в мережі ініціалізуються як малі випадкові значення. після чого алгоритм зворотного поширення обчислює поточний градієнт поверхні помилки і змінює ваги в напрямку найбільшого градієнта. За умови що поверхня помилки є порівняно гладкою, можна сподіватися, що ваги зійдуться до глобального мінімуму, але цей результат не є гарантованим, і навчання може зупинитися досягши одного з локальних мінімумів.

Алгоритм: Backpropagation
1) Ініціалізувати ваги w_{ij}
2) Подати дані x_i *на вхід та отримати вихід* o_i
3) Порахуати значення помилки
$\delta_k = o_k(1 - o_k)(t_k - o_k)$
4)Визначити загальну помилку
$\delta_j = o_j(1 - o_j) \sum \delta_k w_{j,k}$
5) Оновити значення вагів
$\Delta w_{i,j} = \alpha \Delta w_{i,j} + (1 - \alpha)\eta \delta_j o_i$
$w_{i,j} = w_{i,j} + \Delta w_{i,j}$
6) Повторити кроки 2-5 до отримання бажаного результату

2.3. Recurrent Neural Networks

Рекурентні нейронні мережі (RNN)[10] є типом архітектури нейронної мережі, яка використовується для роботи з послідовностями даних. Прикладами таких даних є рукописний текст, текстові або числові ряди. Рекурентні нейронні мережі широко використовуються для задач моделювання натуральних мов, обробки, аналізу та генерацію тексту, розпізнавання мовлення, та рукописного тексту. Основна відмінність рекурентних нейронних мереж від багатошарового перцептрону, в тому як інформація проходить через мережу. На відміну від перцептронів, RNN має цикли, і передає інформацію що проходить через мережу назад на вхід. Це дозволяє їм розширити функціональність перцептрона, враховуючи попередні вхідні дані $X_{0:t-1}$, а не тільки поточні X_t. Рекурентна нейронна мережа може мати декілька таких послідовних блоків.

Процес передачі інформації з попередньої ітерації до прихованого шару може бути описаний таким чином:
Позначимо прихований стан та вхідні у вигляді $H_t \in R^{n \times h}$ і $X_t \in R^{n \times d}$ відповідно, де n позначає кількість семплів, d розмірність кожного семплу, і h розсірність прихованого шару. Також в процесі роботи рекурентної мережі використовується матриця вагів $W_{xh} \in R^{d \times h}$, матриця переходу прихованого стану $W_{hh} \in R^{h \times h}$ та параметр $b_h \in R^{1 \times h}$hidden-state-to-hidden-state matrixwhh□RH × hand a bias parameterbh□R1 × h.lastly, всі ці інформації передаються до функції активації, що зазвичай є логістичною сигмоідором функція TANH для підготовки градієнтів для

використання в спину. Встановлення всіх цих нотаційного ступеня врожає рівняння 1 як приховану змінну та рівняння 2 як вихідну змінну.

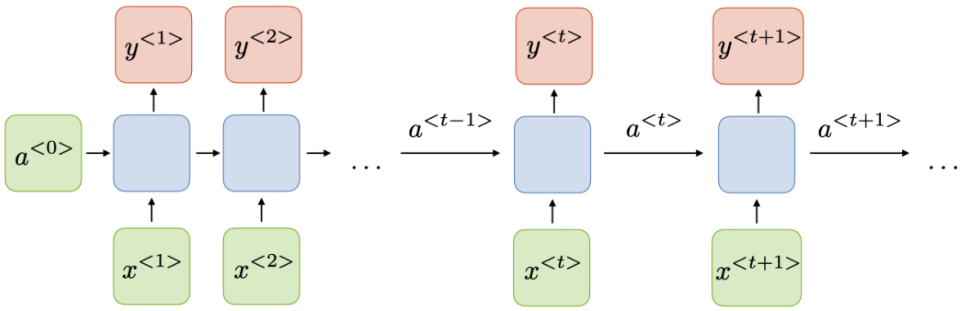
Для отриманняя кінцевого результату ці значення передаються до функції активації, як правило сігмоїди чи $tanh$, що забезпечують легкий підрахунок градієнтів що використовуються для навчання. Значення прихованого стану такої мережі можна підрахувати за формулою (2.2), а вихідне значення за формулою (2.3).

$$H_t = \varphi_h(X_t W_{xh} + H_{t-1} W_{hh} + b_h) \tag{2.2}$$

$$O_t = \varphi_o(H_t W_{xo} + b_o) \tag{2.3}$$

Оскільки поточний прихований стан H_t рекурентно включає H_{t-1}, окрім власне попереднього стану H_{t-1}, він також містить всі попередні приховані стани $H_{0:t-1}$. Така взаємопов'язаність прихованих станів робить використання традиційного алгоритму зворотнього поширення неможливим, замість нього для навчання використовується його модифікація під назвою Backpropagation Through Time.

Backpropagation Through Time - це адаптація алгоритму зворотнього поширення для рекурентних нейронних мереж. Цей алгоритм розгортає рекурентну мережу за часом, щоб побудувати традиційну нейронну мережу до якої можна застосовувати алгоритм зворотнього поширення.



Зображення 9 - Розгорнута RNN[11].

Як і в більшості нейронних мереж, надмірне зростання або затухання градієнта є ключовою проблемою рекурентних мереж. Градієнти мережі в наслідок послідовного множення на значення менші за 1 затухають з кожним шаром (або

кроком часу) і згодом повністю зникають. Це значно зменшує вплив станів віддалених від поточного. Аналогічним чином, якщо значення градієнту великі, такий процес може призвести до протилежного результату, що називається проблемою вибухаючого градієнту. Вибухаючий градієнт призводить до нестабільності вагів мережі. Проблема зникаючого та вибухаючого градієнтів є одним з основних факторів обмежуючих навчання нейронної мережі, для їх розв'язання було розроблено ряд рішень, які здатні значно покращити результативність навчання.

2.4. Long Short-Term Memory

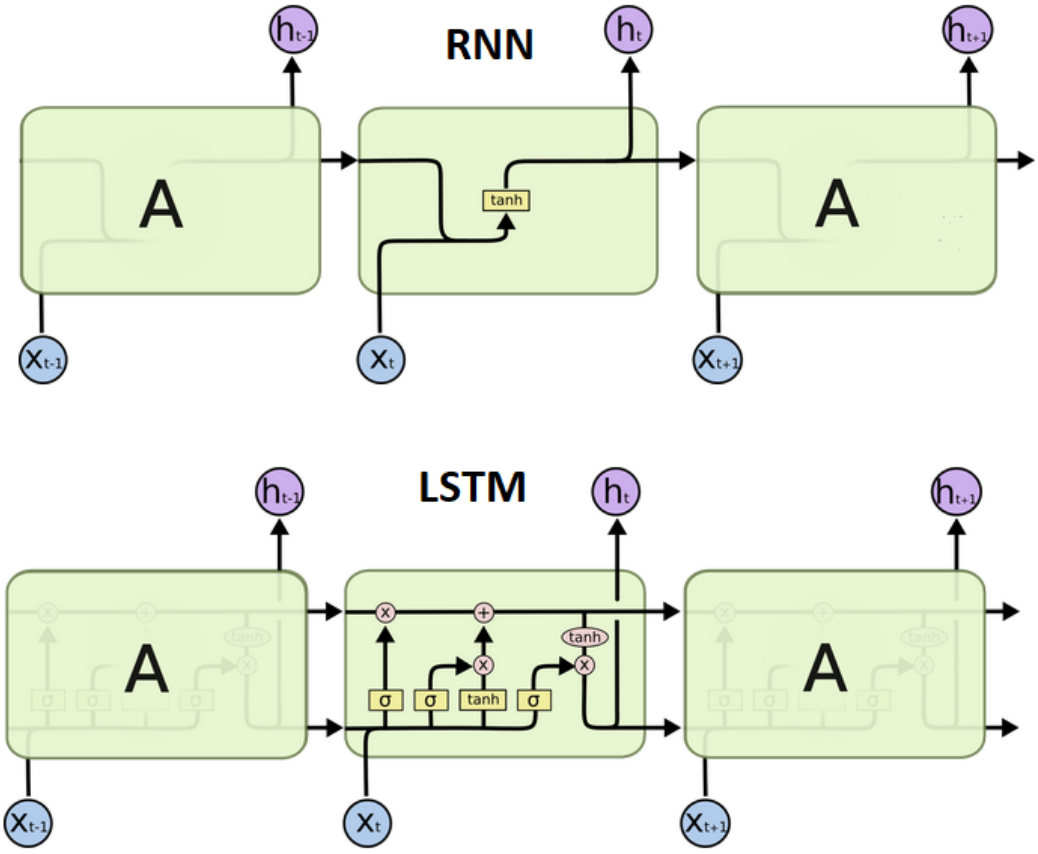
Рекурентні нейронні мережі є дуже потужним класом обчислювальних моделей, здатних моделювати вкрай складні залежності. Але їх значний потенціал на практиці обмежений ефективністю процесу тренування. В процесі навчання утворення залежностей ускладнюється з віддаленням від поточного стану через проблему вибуху або затухання градієнту[12][13]. Через це типові імплементації RNN не здатні створювати залежності з часовою затримкою більшою ніж 5-10 кроків, що значно обмежує ефективеість таких моделей для будь-яких задач що потребують довгострокового запам'ятовування.

Для вирішення цієї проблеми була розроблена модифікація рекурентної нейронної мережі під назвою Long Short-Term Memory. LSTM може зберігати дані протягом до 1000 дискретних кроків часу, завдяки спеціальній архітектурі що забезпечує стабільність значень збережених у клітинах завдяки використанню вхідних та вихідних мультиплікативних вентилів(ваг) що навчаються відкриватися і закривати доступ до клітин пам'яті. Алгоритм навчання LSTM виконується за сталий час, Його обчислювальна складність дорівнює $O(1)$ за кількістю ваг та часових кроків. Здатність до утворення довготривалих залежностей надає LSTM значну перевагу над традиційними рекурентними мережами для задач обробки природнього тексту та його оптичного розпізнавання.

Основною структурною одиницею прихованого шару мережі LSTM є блок пам'яті, який містить одну або декілька запам'ятовуючих клітин та пару адаптивних, мультиплікативних вентилів, що обмежують вхідний та вихідний потік даних для всіх клітин у блоку пам'яті. Кожна клітина пам'яті має власний стан, який залишається сталим за відсутності нових сигналів, що вирішує проблему зникаючого градієнту. Стабільність внутрішнього стану забезпечується вхідним та ви-

хідним вентилями. Коли вони закриті, (значення близьке до нуля), нерелевантні вхідні дані не впливають на внутрішній стан клітини, а стан клітини не впливає на решту мережі. Стан клітини пам'яті оновлюється на основі її поточного стану та входів самої клітини, та вхідних і вихідних вентилів.

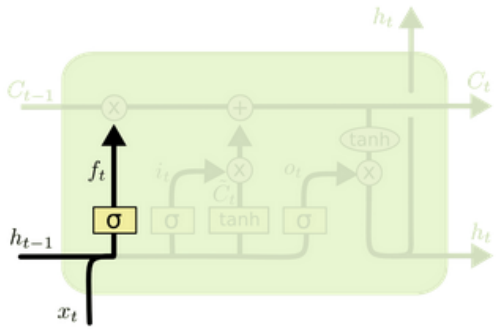
Щоб запобігти необмеженому зростанню значень збережених в пам'яті, що може призвести до повної втрати дискримінативної здатності пам'яті, застосовується спеціальний адаптивний вентиль забування, який поступово зменшує значення збережені в пам'яті з метою забування нерелевантної інформації.



Зображення 10 - Порівняння архітектур RNN і LSTM[14].

2.4.1. Покроковий огляд LSTM

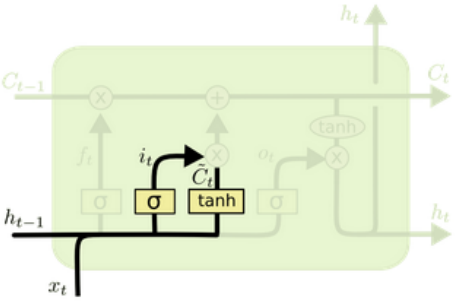
Перший крок у роботі LSTM є вирішення того, яка інформація зі стану клітини викидається. Це рішення здійснюється сигмовидним шаром, що називається вентилем забування. Він розглядає попередній прихований стан h_{t-1} та отриманий вхід x_t, і виводить число від 0 до 1 для кожного значення в стані клітини C_{t-1}. Число 1 означає повністю зберегти поточний стан, а 0 - повністю його видалити.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Зображення 11 - Вентиль забування.

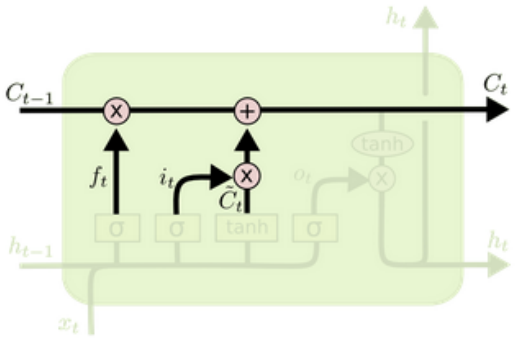
Наступним кроком є вирішення того, яку нову інформація ми збираємося зберігати у клітині. Цей процес складається з двох частин: по-перше, сигмоїдний шар вхідного вентиля вирішує, які значення ми будемо оновлювати. Далі, шар $tanh$ утворює вектор нових кандидатів з вхідних даних, $\tilde{C}_t$, який може бути додаеим до стану. На наступному кроці ці значення поєднуються щоб отримати оновлення для стану пам'яті.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Зображення 12 - Вхідний вентиль.

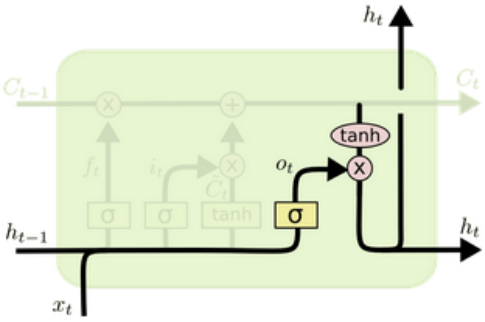
Тепер оновимо внутрішній стан клітини пам'яті C_{t-1} та отримаємо новий стан C_t. Для цього помножимо попередній стан на підраховане раніше значення вентиля забування f_t та додаємо нові значення з вхідного вентилю, що дорівнюють $i_t * \tilde{C}_t$.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Зображення 13 - Оновлення стану пам'яті.

Щоб отримати кінцевий результат нам потрібно вирішити, що ми збираємося вивести. Значення для виведення буде отримане зі стану клітини за допомогою вентилю виведення, аналогічного до вентилів забування та введення. Стан клітини стискається до проміжку $(-1, 1)$ за допомогою функції $tanh$ та множиться на значення отримане з шару вентиля виведення, що забезпечує виведення лише бажаних даних.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Зображення 14 - Вихідний вентиль.

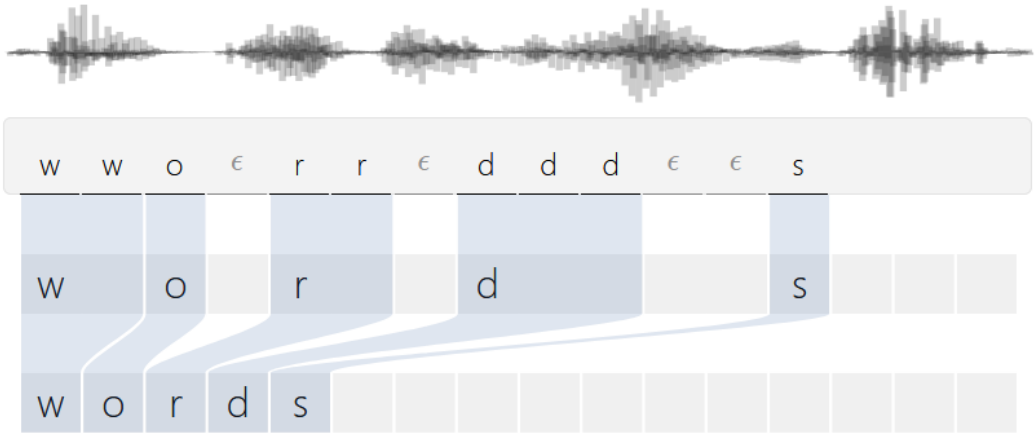
2.5. Connectionist Temporal Classification

Розмічення несегментованих послідовностей даних є однією з ключових проблем реальних задач машинного навчання[15]. Прикладами таких задач є розпізнавання жестів, мовлення, та рукописного тексту, ці задачі характеризуються довгими мінливими та зашумленими потоками вхідних даних, і вимагають встановлення відповідності між частинами вхідного потоку та певними дискретними значеннями.

Для розв’язання задач розмічення послідовностей широко використовувалися складні підходи що вимагали значних зусиль та знання специфіки задачі для їх розробки, такі як приховані марківські моделі та умовні випадкові поля. Ці підходи вимагали окремої розробки ознак та станів, і у випадку прихованих марківських моделей робили припущення про незалежність випадкових подій, що в реальних задачах, таких як розпізнавання мовлення чи тексту не виконується.

З іншого боку, рекурентні нейронні мережі не вимагають ніякої попередньої інформації, окрім подання вхідних даних на обробку моделі. Вони навчаються дискримінативно, а не генеративно, і використання прихованого стану забезпечує потужний загальний механізм моделювання часових рядів. Крім того вони, як правило, є стійкими до шумів та викидів у вхідних даних.

Але при цьому застосувати традиційну RNN безпосередньо до розмічення послідовностей не є можливим. Проблема полягає в тому, що стандартні цільові функції нейронної мережі визначаються окремо для кожної точки тренувальної послідовності, тобто RNN можна навчити тільки робити незалежні класифікації для кожного окремого елемента послідовності, але оскільки для багатьох реальних послідовностей існує безліч різних способів коректного розмічення необроблених даних, це означає, що тренувальні дані повинні бути попередньо сегментовані, а результат роботи мережі має бути додатково оброблений щоб отримати остаточну вихідну послідовність.



Зображення 15 - Невідповідність між довжинами вхідної та вихідної послідовностей на прикладі розпізнавання мовлення[16].

Connectionist Temporal Classification або CTC є підходом що має на меті

розв'язання цієї проблеми. Основна ідея полягає в інтерпретації виходів мережі як розподілу ймовірностей всіх можливих послідовностей значень, за умови певної вхідної послідовності. Маючи такий розподіл, можна отримати цільову функцію, яка безпосередньо максимізує ймовірність правильного розмічення. Оскільки ця функція є диференційованою, то мережа може навчатися за допомогою стандартного алгоритму зворотнього поширення в часі.

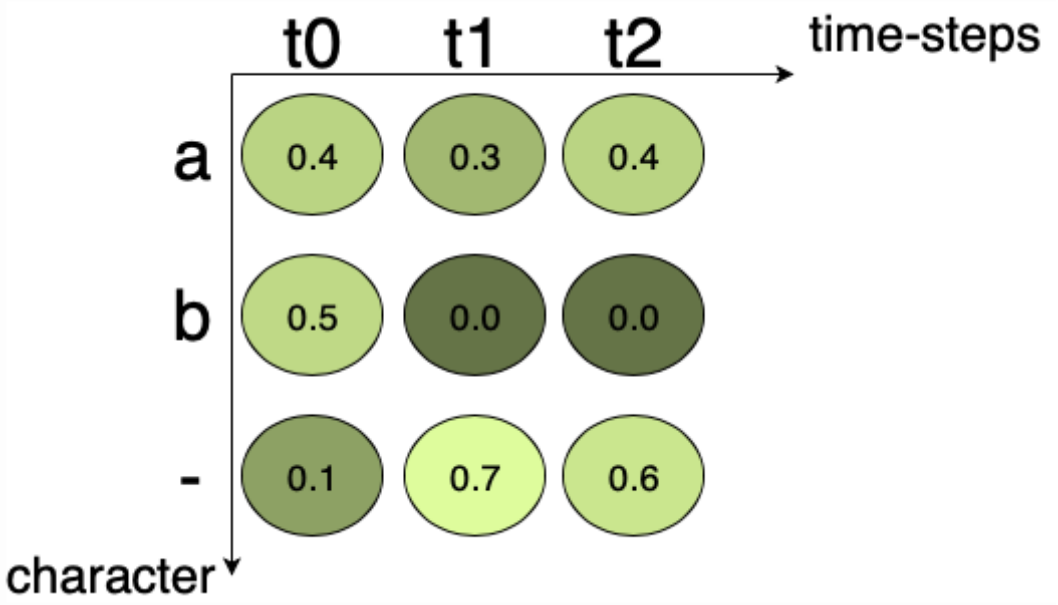
Нейронна мережа з CTC має вихідний шар softmax, розмір якого на 1 більше за розмір алфавіту L. Активації перших $|L|$ нейронів інтерпретуються як ймовірність спостереження відповідних значень у певний час. Останній нейрон позначає ймовірність спеціального символа ϵ, що позначає порожній символ. Разом ці виходи визначають ймовірності всіх можливих способів утворення відповідності між можливими значеннями та вхідною послідовністю. Тоді загальну ймовірність будь-якої послідовності символів можна знайти шляхом підсумовування ймовірностей різних її вирівнювань.

Отримана на виході мережі послідовність матиме таку саму довжину, як і вхідна послідовність, для того щоб перетворити її на коректний текст, спочатку приберемо всі послідовні дублікати символів, після чого видалимо всі порожні токени ϵ, щоб отримати бажану вихідну послідовність. Використання порожнього токена ϵ є необхідним для позначення кількох повторів одного символа підряд.

Зображення 16 - Перетворення вихідної послідовності у текст.

2.5.1. Функція втрат та тренування моделі

Щоб навчити RNN виконувати задачу розпізнавання послідовності, нам потрібна відповідна функція втрат що дозволяє оцінити якість роботи мережі за вхідним потоком(зображенням) та цільовим виходом(текстом). Результатом роботи мережі є матриця ймовірностей для кожного символу на кожному кроці, при цьому на кожному кроці часу сума ймовірностей всіх символів має дорівнювати 1.



Зображення 17 - Приклад виходу нейронної мережі[17].

Для того щоб визначити ймовірність певної послідовності за результатом роботи мережі, необхідно перемножити значення ймовірностей на кожному кроці. При цьому цільова текстова послідовність спочатку має бути перетворена до вигляду послідовності що використовується у мережі. Наприклад за умови що довжина вхідної послідовності дорівнює 3, та цільова послідовність дорівнює a, така послідовність може бути утворена будь-якою з наступних: aaa, $aa\epsilon$, $a\epsilon\epsilon$, ϵaa, $\epsilon\epsilon a$ $\epsilon a\epsilon$. Сума ймовірностей всіх послідовностей що утворюють цільову послідовність є ймовірністю цієї послідовності.

Таким чином можемо визначити функцію втрати так щоб максимізувати ймовірність для правильної результуючої послідовності.

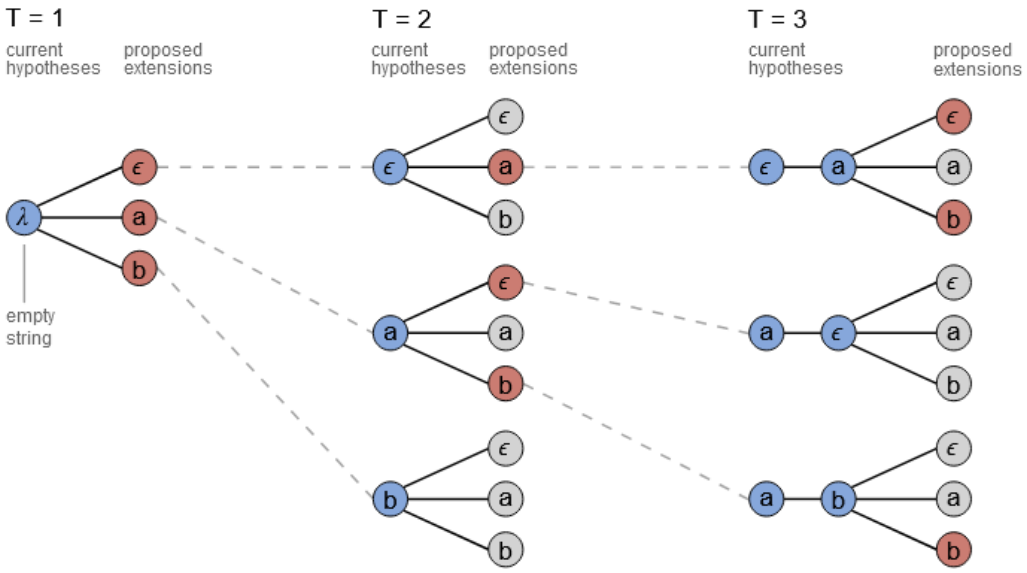
$$y = \sum_{X,Y} -log_p(Y|X) \tag{2.4}$$

2.5.2. Декодування результатів

Після того як нейронна модель була натренована, щоб використовувати її для розпізнавання нових вхідних даних необхідно вміти декодувати отриманий результат. Тобто задача полягає у визначенні найбільш ймовірної послідовності за матрицею ймовірностей символів для кожного кроку.

У реальних задачах виконати перебір всіх варіантів не є можливим через значну кількість необхідних обчислень, тому використовуються алгоритми що дозволяють отримати послідовність, близьку до найкращої. Найпростіший спосіб це зробити - використовуючи жадібний алгоритм що обирає символ з максимальною ймовірністю на кожному кроці.

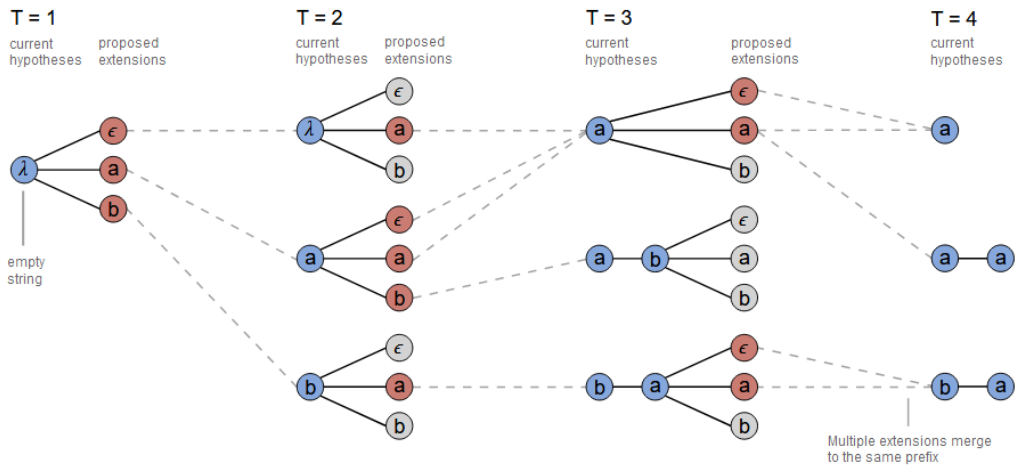
Більш ефективним але більш вибагливим до кількості обчислень є алгоритм декодування Beam Search, який ітеративно створює кандидати послідовності (промені) і оцінює їх. Алгоритм починаючи з порожньої послідовності, повторюється протягом усіх часових кроків. На кожному кроці послідовності-кандидати розширюються всіма символами алфавіту, після чого проводится їх оцінка, і всі кандидати окрім заданої кількості найкращих відкидаються.



Зображення 18 - Приклад роботи алгоритму Beam Search.

Недоліком такого алгоритму є те, що ймовірності різних послідовностей які утворюють один і той самий результат вважаються різними. Для вирішення цієї проблеми для кожного кандидату можна зберігати префікс утвореного тексту, і за-

мінювати кандидатів що утворюють однакову результуючу послідовеість одним, ймовірність якого дорівнює сумі ймовірностей всіх таких кандидатів.



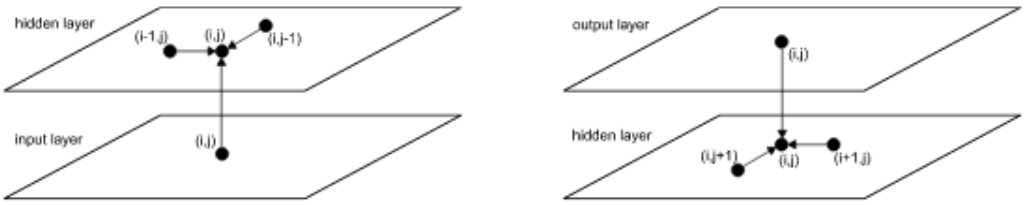
Зображення 19 - Модифікований алгоритм Beam Search з урахуванням префіксів.

2.6. Multi Dimensional Reccurent Neural Networks

Рекурентні нейронні мережі були розроблені як спосіб використання нейронних мереж для обробки послідовностей даних[18]. Наявніть рекурентних зв'язків надають таким мережам можливість використовувати попередній контекст. Окрім того, RNN здатні варіювати швидкість зміни їх внутрішнього стану, що робить їх більш стійкими до різноманітних спотворень вхідних даних.

Однак традиційні архітектури рекурентних нейронних мереж здатні працювати лише з одновимірними даними, що для їх застосування у багатовимірних завдачах, вимагає попередньої обробки даних з метою перетворення до одновимірної форми, наприклад послідовно подаючи на вхід мережі по одній вертикальній лінії зображення.

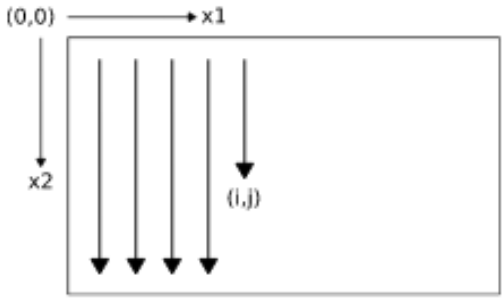
Основна ідея багатовимірної рекурентної нейронної мережі (MDRNN) полягає в тому, щоб замінити єдиний рекурентний зв'язок, як в стандартних RNN, на стільки рекурентних зв'язків, скільки є вимірів у вхідних даних. Під час прямого проходу в кожній точці послідовності даних, прихований шар отримує як зовнішній вхід, так і власні активації за попередній крок по всім вимірам.



2D RNN Forward pass. 2D RNN Backward pass.

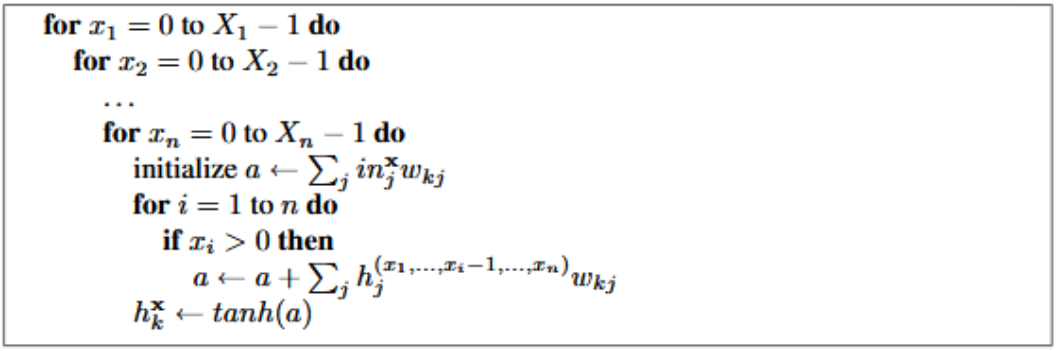
Зображення 20 - Рекурентні зв'язки MDRNN для двовимірного випадку.

Очевидно, що дані повинні оброблятися таким чином, що коли мережа досягає певної точки в n-вимірній послідовності, всі точки з яких вона має отримати попередні активації вже були пройдені. Це можна забезпечити, проходячи вхідні дані в певному порядку. Приклад одного з можливих порядків проходження наведено в зображенні 21.



Зображення 21 - Порядок проходження який забезпечує що для кожної точки (i, j) попередні точки $(i-1, j)$ та $(i, j-1)$ є відомими.

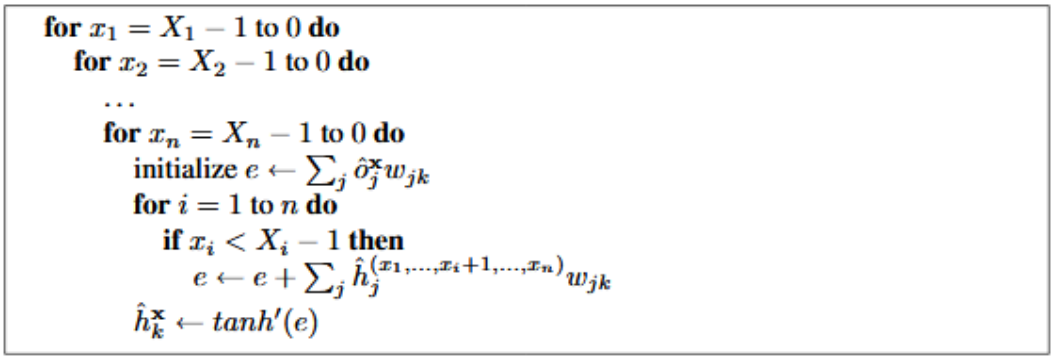
Визначимо i_j^x та h_k^x як відповідні активації вхідних та прихованих нейронів мережі, а також визначимо ваги між зв'язками як w_{kj}, та візьмемо функцію активації $tanh$. Тоді алгоритм прямого проходу мережі може бути визначений таким чином:



for $x_1 = 0$ to $X_1 - 1$ **do**
  for $x_2 = 0$ to $X_2 - 1$ **do**
    ...
    for $x_n = 0$ to $X_n - 1$ **do**
      initialize $a \leftarrow \sum_j in_j^{\mathbf{x}} w_{kj}$
      for $i = 1$ to n **do**
        if $x_i > 0$ **then**
          $a \leftarrow a + \sum_j h_j^{(x_1,\ldots,x_i-1,\ldots,x_n)} w_{kj}$
      $h_k^{\mathbf{x}} \leftarrow tanh(a)$

Зображення 22 - MDRNN Forward Pass.

Визначимо $\hat{o}_j^x$ та $\hat{h}_k^x$ як похідні цільової функції відносно відповідних активацій вихідних та прихованих нейронів мережі. Тоді алгоритм зворотнього проходу мережі може бути визначений таким чином:

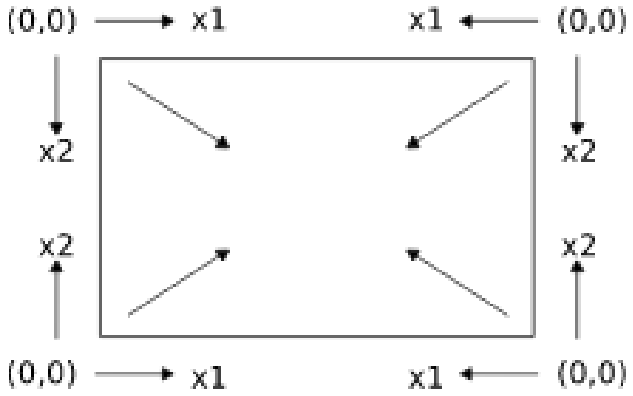


for $x_1 = X_1 - 1$ to 0 **do**
  for $x_2 = X_2 - 1$ to 0 **do**
    ...
    for $x_n = X_n - 1$ to 0 **do**
      initialize $e \leftarrow \sum_j \hat{o}_j^{\mathbf{x}} w_{jk}$
      for $i = 1$ to n **do**
        if $x_i < X_i - 1$ **then**
          $e \leftarrow e + \sum_j \hat{h}_j^{(x_1,\ldots,x_i+1,\ldots,x_n)} w_{jk}$
      $\hat{h}_k^{\mathbf{x}} \leftarrow tanh'(e)$

Зображення 23 - MDRNN Backward Pass.

В певній точці вхідної послідовності мережа може мати контекст лише з попередніх точок. Для задач оптичного розпізнавання тексту це обмеження не є критичним, але існує можливість його усунення шляхом одночасної обробки кількома шарами в різних напрямках.

Для одновимірних RNN, ця проблема була вирішена шляхом введення двонаправлених рекурентних нейронних мереж (BRNN) [19]. BRNN містить два окремих прихованих шара, які обробляють вхідну послідовність у прямому та зворотньому напрямках. Два прихованих шара мають спільний вихід, тим самим забезпечуючи доступом як до минулого, так і майбутнього контексту. Ідея BRNN може бути розширена до N-вимірних даних, використовуючи 2^n окремих прихованих шарів, кожен з яких обробляє послідовність у вказаному вище порядку, в різних напрямках.



Зображення 24 - Multi-directional MDRNN.

2.6.1. MDLSTM

В цьому розділі ми неявно припускали, що мережа може використовувати весь контекст, до якого вона має доступ. Однак, для стандартних архітектур RNN, обсяг контексту, який може ефективно використовувався є обмеженим. Проблема полягає в тому, що вплив певного вхідного сигналу на стан прихованих шарів мережі зменшується з віддаленням від поточного елемента послідовності в наслідок зникаючого градієнта.

LSTM - це архітектура RNN, спеціально розроблена для вирішення проблеми зникнення градієнту. Прихований шар LSTM складається з декількох блоків пам'яті. Кожен блок має задану кількість запам'ятовуючих одиниць, активація яких контролюється трьома мультиплікативними вентилями для фільтрації входу, виходу та забування нерелевантних даних. Ці вентилі дозволяють клітинам зберігати інформацію протягом тривалого часу, тим самим уникаючи проблеми зникнення градієнту.

Стандартна архітектура LSTM є одновимірною, оскільки клітинка містить єдиний зв'язок. Цю архітектуру можна легко розширити до n вимірів, використовуючи по одному з'єднанню з вентилем забування для кожного виміру вхідної послідовності.

2.7. Висновки до другого розділу

Для задачі оптичного розпізнавання рукописного тексту, глибокі нейронні мережі є домінуючим підходом що має значні переваги над різноманітними альтернативами. Але не всі з розглянутих архітектур нейронних мереж є застосовними для задачі оптичного розпізнавання тексту.

Багатошаровий перцептрон є базовою архітектурою нейронної мережі що може бути використана для задач класифікації, але вона не підходить для роботи з послідовностями вхідних даних і не є застосовною для задачі оптичного розпізнавання тексту

Рекурентні нейронні мережі здатні працювати з послідовностями вхідних даних і завдяки наявності циклічних зв'язків враховують минулий контекст, але через зменшення градієнтів з віддаленням від поточного кроку послідовності вхідних даних, цей контекст є сильно обмеженим. Для вирішення цієї проблеми була розроблена архітектура LSTM що здатна зберігати значення у своєму внутрішньому стані протягом тривалого часу.

Хоча рекурентні моделі здатні працювати з послідовностями, вони виконують розпізнавання для кожного окремого вхідного вектора, тобто у випадку оптичного розпізнавання тексту, однієї вертикальної лінії зображення шириною в один піксель. Така послідовність вхідних даних не утворює однозначну відповідність бажаному результату, тому для навчання рекурентних моделей утворювати коректну текстову послідовність з вхідних даних використовується Connectionist Temporal Classification що дозволяє навчати модель утворювати відповідну текстову послідовність.

Хоча модель на базі архітектури LSTM з використанням CTC є ефективним рішенням для задачі оптичного розпізнавання тексту і широко застосовується на практиці, для багатовимірних задач, таких як розпізнавання тексту на зображенні, існує модифікована версія архітектури під назвою MDLSTM що здатна утворювати контекстні залежності в кількох напрямках, і показує кращі результати на практиці.

Тренувальні дані та їх обробка

3.1. IAM Database

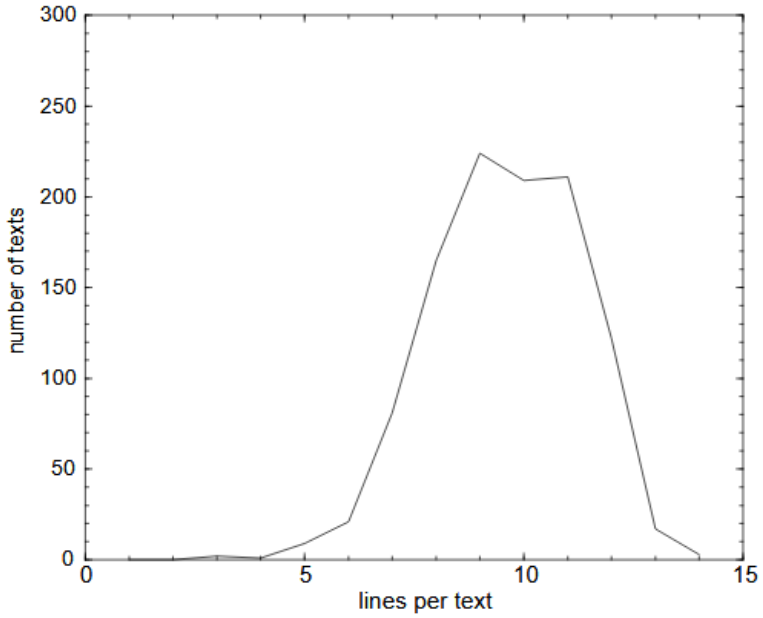
Наявність великих обсягів даних для навчання та тестування є необхідною передумовою для побудови систем розпізнавання рукописного тексту з високою точністю.

Для виконання цієї роботи був використаний датасет IAM[2] що містить зображення слів та речень рукописного англомовного тексту. Датасет складається з 82,227 екземплярів рукописних слів, розподілених серед 9,285 текстових рядків, написаних приблизно 400 особами. Лексикон датасету включає 10841 слів. Датасет IAM широко використовується для тренування моделей оптичного розпізнавання тексту і сегментації та аналізу документів.

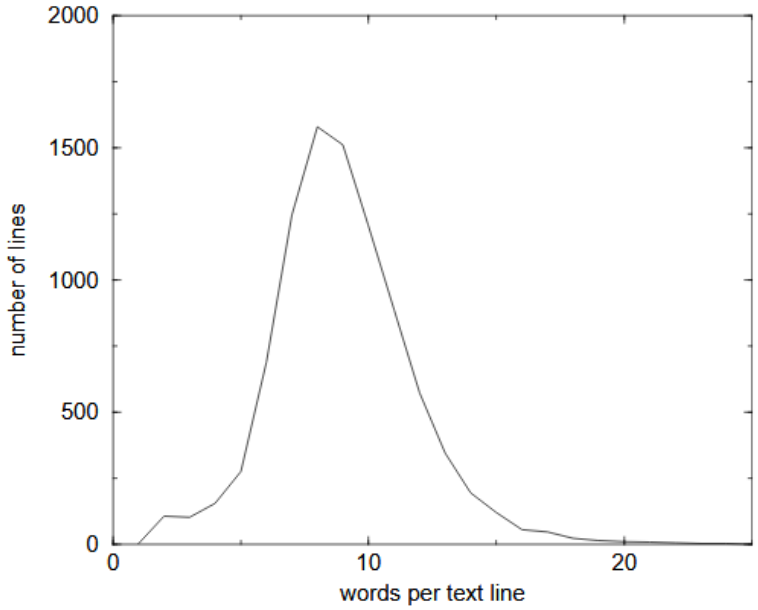
A	Press: reportage	44
B	Press: editorial	27
C	Press: reviews	17
D	Religion	17
E	Skills, trades, and hobbies	38
F	Popular lore	44
G	Belles letters, biography, essays	77
H	Miscellaneous	30
J	Learned and scientific writings	80
K	General fiction	29
L	Mystery and detective fiction	24
M	Science fiction	6
N	Adventure and western fiction	29
P	Romance and love story	29
R	Humour	9
	Total	500

Зображення 25 - Тексти датасету по категоріям.

Кількість текстів в датасеті за їх типом проілюстрована в зображенні. Розподіл текстів за кількістю рядків проілюстрований в зображенні. Розподіл рядків за кількістю слів проілюстрований в зображенні.



Зображення 26 - Розподіл кількості рядків у тексті.



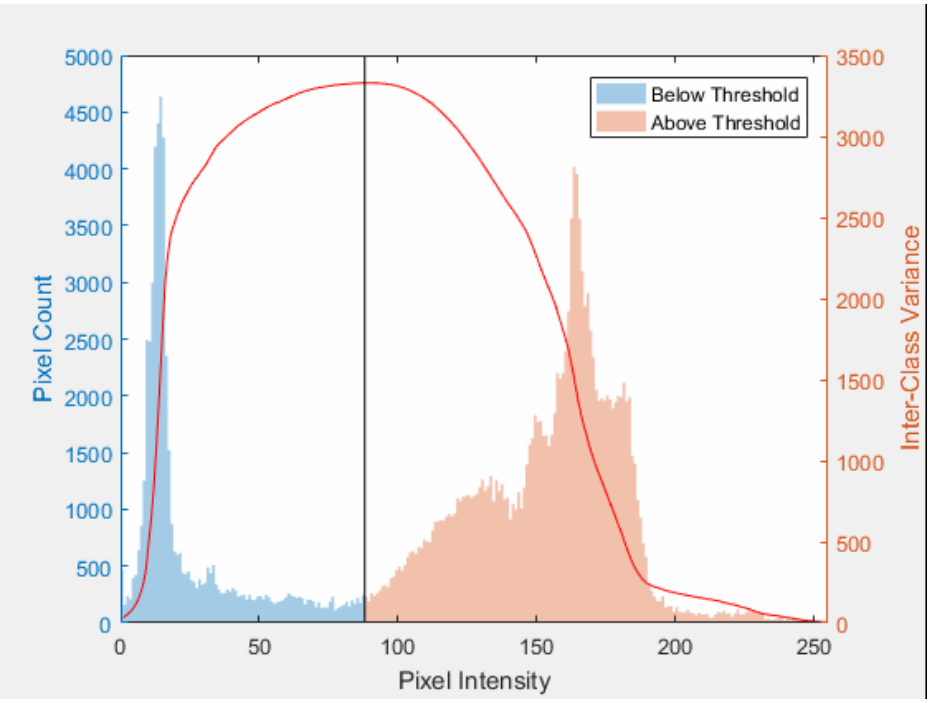
Зображення 27 - Розподіл кількості слів у рядку.

3.2. Бінаризація

Сучасні системи оптичного розпізнавання тексту вимагають проведення бінаризації[21] вхідного зображення, з метою відділення пікселів тексту від фону. Більшість методів бінаризації зображень покладаються на глобальні або локальні методи дискримінації за пороговим значенням. Ці пороги зазвичай визначаються за допомогою гістограм і статистичного аналізу розподілу яскравості. Такі підходи не враховують форму тексту, і є чутливими до шумів що можуть бути присутні на зображенні.

Метод Оцу є широко відомим методом бінаризації. Він припускає, що колір тексту та фону є однорідними. Поріг що розділяє пікселі зображення на два класи обирається так, щоб максимізувати дисперсію значення інтенсивності між класами. Оптимальне значення порогу визначається перебором всіх можливих значень, серед них обирається те, яке має найбільше значення σ^2, при цьому $\omega_0(t)$ і $\omega_1(t)$ позначають сумарні ймовірності першого і другого класів, а $\mu_0(t)$ і $\mu_1(t)$ їх середні значення за умови порогу t.

$$\sigma^2 = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2 \tag{3.1}$$



Зображення 28 - Метод Оцу визначає поріг знаходячи значення що забезпечує найбільшу міжкласову дисперсію[22].

3.3. Non-Local Means Denoising

Видалення шуму - один із найважливіших і широко використовуваних засобів у роботі з зображеннями. Його метою є відновлення оригінальних деталей зображення максимально точно, при цьому усунувши небажані шуми. Видалення шуму часто використовується як етап попередньої обробки для інших задач розпізнавання, сегментації та компресії.

Шум утворений камерами і сканувальними пристроями зазвичай є гауссовим[23]. Існує велика кількість різноманітних алгоритмів видалення шуму, більшість з яких в тому чи іншому вигляді базуються на усередненні. Одним з найпопулярніших нелінійним методом є білатеральний фільтр. У цьому методі відновлене значення кожного пікселя замінюється середнім значенням всіх пікселів певного околу. Однак, коли цей тип фільтрів сусідства використовується на природних зображеннях, повторювані структури що трапляються у зображенні можуть бути видалені разом з шумом.

Для вирішення цієї проблеми було запропоновано метод Non-Local Means Denoising. У цьому методі щоб модифікувати значення поточного пікселя розглядається все зображення. Подібність між певним вікном навколо центрального пікселя та вікнами сусідніх пікселів використовується для регулювання поточного пікселя. Незважаючи на те, що цей метод дає дуже хороші результати, обчислювальна складність має порядок $O(n^4)$, що значно збільшує час виконання, порівняно з більш простими алгоритмами.

Зображення 29 - Видалення шуму[24].

Алгоритми видалення шуму широко використовуються для задач оптичного розпізнавання. Одна з основних мотивацій для їх використання є чутливість деяких засобів обробки зображень таких як бінаризація чи скелетонізація до шуму. Використання засобів видалення шуму може значно покращити результат бінаризації.

3.4. Топологічна скелетонізація

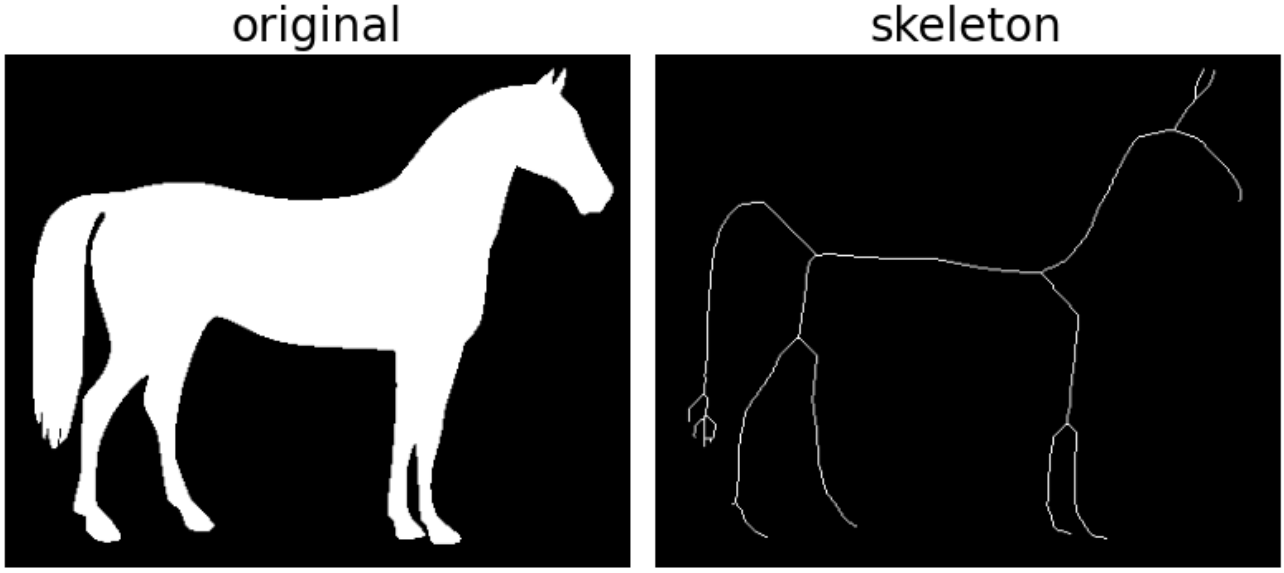
Топологічна скелетонізація - це процес обробки бінаризованого зображення з метою перетворення його на неперервну лінію товщини 1 піксель рівновіддалену від контурів об’єкта на початковому зображенні. Результат скелетонізації зазвичай підкреслює геометричні та топологічні властивості фігури, і зберігає зв’язність.

Алгоритми скелетонізації можуть бути використані для попередньої обробки тренувальних даних для систем оптичного розпізнавання тексту з метою покращення якості розпізнавання шляхом відкидання зайвої інформації, що може дати позитивний ефект на здатність моделі розпізнавання до генералізації, та збільшити її ефективність для розв’язання реальних задач.

Процес скелетонізації знищує велику кількість інформації що міститься у зображенні, що також може призвести до погіршення результатів розпізнавання. Також деякі алгоритми скелетонізації можуть додавати зайви деталі які не відповідають формі зображення. Ці алгоритми є особливо чутливими до різноманітних шумів і спотворень, тому попереднє використання алгоритмів видалення шуму є необхідним.

Але при цьому, оптичне розпізнавання рукописного тексту є однією з більш привабливих галузей для застосування таких алгоритмів через порівняно високу якість зазвичай відксанованих тренувальних зображень, прості геометричні форми символів та легке виділення фону.

Для попередньої обробки бінаризованих тренувальних зображень моделі оптичного розпізнавання тексту в цій роботі був використаний алгоритм Zhang-Suen з метою скелетонізації тексту на зображеннях. Алгоритм полягає в ітеративному видаленні пікселів біля країв об’єкта, що добре працює для фігур зі сталою товщиною, таких як текст.



Зображення 30 - Приклад топологічної скелетонізації[25].

3.5. Висновки до третього розділу

Датасет IAM Database є одним з найбільших за обсягом відкритих англомовних датасетів для оптичного розпізнавання рукописного тексту, він широко використовується в дослідженнях на тему оптичного розпізнавання тексту.

Задача розпізнавання рукописного тексту є сприятливою для використання різноманітних методів попередньої обробки завдяки високій якості тренувальних зображень і легко роздільному фону.

Серед меотодів попередньої обробки одним з найбільш перспективних є скелетонізація, що дозволяє прибрати значну частину варіативності у стилі написання, і залишити лише примітивні форми.

Оцінка отриманих результатів

4.1. Метрики якості розпізнавання тексту

Оцінка якості розпізнавання тексту є дещо складнішою порівняно з задачами регресії чи класифікації тому що звичні метрики для оцінки схожості між бажаним та отриманим результатом не є застосовними.

Метрика відстані Левенштейна дозволяє визначити відмінності між двома текстовими рядками, і обчислюється як мінімальна можлива кількість операцій вставки, заміни або видалення символів необхідних для перетворення одного рядка на інший. Відстань Левенштейна може бути визначина рекурсивно за формулою:

$$\text{lev(a,b)} = \begin{cases} |\text{a}| & \text{if } |\text{b}|=0, \\ |\text{b}| & \text{if } |\text{a}|=0, \\ \text{lev(tail(a),tail(b))} & \text{if a[0]=b[0],} \\ 1 + \min \begin{cases} \text{lev(tail(a),b)} \\ \text{lev(a,tail(b))} \\ \text{lev(tail(a),tail(b))} \end{cases} & \text{otherwise.} \end{cases} \tag{4.1}$$

Для оцінки якості розпізнавання моделі були використані метрики Character Error Rate (CER) та Word Error Rate(WER)[21] що базуються на відстані Левенштейна. Розглянемо випадок порівняння вихідного текстового рядка з очікуваним. Позначимо кількість вставок C_{ins}, кількість видалень C_{del}, кількість замін C_{sub}, необхідних для перетворення одного рядка в інший. Тоді загальна кількість

помилок визначається як:

$$C_{error} = C_{ins} + C_{del} + C_{sub} \tag{4.2}$$

Кількість правильних символів $C_{correct}$ визначається як різниця загальної кількості символів C_{all} та кількості помилок C_{error}. Оскільки кількість помилок може бути більшою за кількість символів, ця метрика може бути від'ємною.

$$C_{correct} = C_{all} - C_{error} \tag{4.3}$$

Метрика CER може бути порахована як відношення $C_{correct}$ до загальної кількості символів. Метрика WER визначається аналогічно.

4.2. Огляд результатів

У процесі виконання роботи було здійснено навчання нейронної моделі на базі архітектури MDLSTM використовуючи дані з датасету IAM для тренування, валідації та тестування. Була натренована нейронна модель що містить три приховані шари MDLSTM з загальної кількістю параметрів приблизно 700,000.

Для порівняння також були натреновані дві альтернативні версії моделі з використанням додаткової обробки вхідних даних: одна з використанням видалення шуму алгоритмом Non-Local Means Denoising, та інша з видаленням шуму і скелетонізацією засобами бібліотеки scikit-image.

Використання алгоритму видалення шуму дало незначне покращення результату, в поєднанні зі скелетонізацією, приріст був більш суттєвим.

Модель	Character rate	Word rate
MDRNN	89.29	69.33
MDRNN denoise	89.85	69.97
MDRNN denoise and skeletonize	91.27	70.84

4.3. Висновки до четвертого розділу

Навчання моделей нейронних мереж з метою оптичного розпізнавання тексту вимагає значної кількості тренувальних даних та спеціальних підходів до попередньої обробки даних і оцінки отриманих результатів.

Процес попередньої обробки вхідних даних може вплинути на якість результуючої моделі, що було продемонстровано в процесі тренування моделі оптичного розпізнавання на базі MDLSTM. Запропоновані в роботі методи видалення шуму та скелетонізації були здатні покращити результативність моделі на тестових даних.

Висновки

З розвитком цифрових технологій та збільшенням обчислювальної потужності електронних обчислювальних пристроїв, актуальність задачі оптичного розпізнавння рукописного тексту стрімко зростає.

В цій роботі була розглянута одна з передових архітектур рекурентних нейронних мереж та її застосування для вирішення задачі оптичного розпізнавання рукописного тексту. Також були запропоновані ефективні способи попередньої обробки тренувальних даних засновані на скелетонізації вхідних зображень, і описане їх практичне використання.

Отримана модель може бути використана як складова частина загального програмного рішення для оптичного розпізнавання тексту, аналізу документів, або інших пов’язаних задач.

Запропонований в цій роботі метод попередньої обробки з використанням скелетонізації виявився ефективним, і може бути застосованим для тренування інших моделей що працюють з подібними даними.

Бібліографія

[1] Recognition system overview, `https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5`

[2] U.-V. Marti, H. Bunke. *The IAM-database: an English sentence database for offline handwriting recognition*. Department of Computer Science, University of Bern, Neubrückstrasse 10, 3011 Bern, Switzerland, 2001

[3] Nafiz Arica and Fatos T. Yarman-Vural. *An Overview of Character Recognition Focused onOff-Line Handwriting*. IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews), 31(2), 216–233.

[4] J. Illingworth and J. Kittler. *A Survey of the Hough Transform*. Department of Electronics and Electrical Engineering, University of Surrey, Guildford GU2 5XH, United Kingdom, 1988

[5] R. Hecht-Nielsen. *Theory of the Backpropagation Neural Network*. Proceedings of the International Joint Conference on Neural Networks 1, 593–611, 1989. Neural Networks for Perception, 65–93.

[6] Yacim Joseph Awoamim, Boshoff Douw Gert Brand. *Impact of artificial neural networks training algorithms on accurate prediction of property values*. Department of Construction Economics, Faculty of Engineering, Built Environment and Information Technology, University of Pretoria, Hatfield 0028, South Africa, 2018

[7] M.W. Gardner S.R. Dorling. *Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences*. School of Environmental Sciences, University of East Anglia, Norwich, Norfolk NR4 7TJ, UK, 1998

[8] Kurt Hornik, Maxwell Stinchcombe, Halbert White. *Multilayer feedforward networks are universal approximators*. Technische Universität Wien Austria, University of California, San Diego USA, 1989

[9] Xiaoxue Chen, Lianwen Jin, Yuanzhi Zhu, Canjie Luo, Tianwei Wang. *Text Recognition in the Wild: A Survey*. College of Electronic and Information Engineering, South China University of Technology, China, 2020

[10] Robin M. Schmidt. *Recurrent Neural Networks (RNNs):A gentle Introduction and Overview*. Department of Computer ScienceEberhard-Karls-University TübingenTübingen, Germany, 2019

[11] Recurrent Neural Networks cheatsheet, `https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks`

[12] Sepp Hochreiter, Jürgen Schmidhuber. *Long Short-Term Memory*. Fakult¨at f¨ur Informatik, Technische Universit¨at M¨unchen, 80290 M¨unchen, Germany, 1997

[13] Felix A. Gers, Jurgen Schmidhuber, Fred Cummins *Learning to Forget: Continual Prediction with LSTM*. 9th International Conference on Artificial Neural Networks: ICANN '99, 1999

[14] Understanding LSTM Networks, `https://colah.github.io/posts/2015-08-Understanding-LSTMs`

[15] Alex Graves, Santiago Fernández, Faustino Gomez, Jürgen Schmidhuber. *Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks*. ICML '06: Proceedings of the 23rd international conference on Machine learning, 2006

[16] Hannun. *Sequence Modeling with CTC*. Distill, 2017

[17] Explanation of Connectionist Temporal Classification, `https://sid2697.github.io/Blog_Sid/algorithm/2019/10/19/CTC-Loss.html`

[18] Alex Graves, Santiago Fernández, Jürgen Schmidhuber. *Multi-Dimensional Recurrent Neural Networks*. IDSIA Galleria 2, 6928 Manno, Switzerland, 2013

[19] M. Schuster and K. K. Paliwal. *Bidirectional recurrent neural networks*. IEEETransactions on Signal Processing, 45:2673–2681, November 1997

[20] Romain Karpinski, Devashish Lohani, Abdel Belaid. *Metrics for Complete Evaluation of OCR Perfor-mance*. IPCV'18 - The 22nd Int'l Conf on Image Processing, Computer Vision, & Pattern Recognition,Jul 2018, Las Vegas, United States

[21] Zohra Saidane and Christophe Garcia. *Robust Binarization for Video Text Recognition*. Orange Labs4, rue du Clos Courtel BP 9122635512 Cesson S´evign´e Cedex - France, 2007

[22] Otsu's method, https://en.wikipedia.org/wiki/Otsu%27s_method

[23] Venkateswarlu Karnati, Mithun Uliyar, Sumit Dey. *Fast Non-Local algorithm for image denoising*. 2009 16th IEEE International Conference on Image Processing (ICIP)

[24] Non-local means, https://en.wikipedia.org/wiki/Non-local_means

[25] Scikit-image docs Skeletonize, https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.

[26] Multi Dimensional Recurrent Networks, https://github.com/areiner222/MDLSTM/blob/master/README.md

ДОДАТОК А
Код моделі MDLSTM

```
import tensorflow as tf
from tensorflow.contrib.rnn import RNNCell, LSTMStateTuple
from tensorflow.contrib.rnn.python.ops.core_rnn_cell_impl import _linear


def ln(tensor, scope=None, epsilon=1e-5):
""" Layer normalizes a 2D tensor along its second axis """
assert (len(tensor.get_shape()) == 2)
m, v = tf.nn.moments(tensor, [1], keep_dims=True)
if not isinstance(scope, str):
scope = ''
with tf.variable_scope(scope + 'layer_norm'):
scale = tf.get_variable('scale',
shape=[tensor.get_shape()[1]],
initializer=tf.constant_initializer(1))
shift = tf.get_variable('shift',
shape=[tensor.get_shape()[1]],
initializer=tf.constant_initializer(0))
ln_initial = (tensor - m) / tf.sqrt(v + epsilon)

return ln_initial * scale + shift


class MultiDimensionalLSTMCell(RNNCell):
"""
Adapted from TF's BasicLSTMCell to use Layer Normalization.
Note that state_is_tuple is always True.
"""

def __init__(self, num_units, forget_bias=0.0, activation=tf.nn.tanh):
self._num_units = num_units
self._forget_bias = forget_bias
self._activation = activation

@property
def state_size(self):
return LSTMStateTuple(self._num_units, self._num_units)
```

```
@property
def output_size(self):
return self._num_units

def __call__(self, inputs, state, scope=None):
"""Long short-term memory cell (LSTM).
@param: inputs (batch,n)
@param state: the states and hidden unit of the two cells
"""
with tf.variable_scope(scope or type(self).__name__):
c1, c2, h1, h2 = state

# change bias argument to False since LN will add bias via shift
concat = _linear([inputs, h1, h2], 5 * self._num_units, False)

i, j, f1, f2, o = tf.split(value=concat, num_or_size_splits=5, axis=1)

# add layer normalization to each gate
i = ln(i, scope='i/')
j = ln(j, scope='j/')
f1 = ln(f1, scope='f1/')
f2 = ln(f2, scope='f2/')
o = ln(o, scope='o/')

new_c = (c1 * tf.nn.sigmoid(f1 + self._forget_bias) +
c2 * tf.nn.sigmoid(f2 + self._forget_bias) + tf.nn.sigmoid(i) *
self._activation(j))

# add layer_normalization in calculation of new hidden state
new_h = self._activation(ln(new_c, scope='new_h/')) * tf.nn.sigmoid(o)
new_state = LSTMStateTuple(new_c, new_h)

return new_h, new_state


def multi_dimensional_rnn_while_loop(rnn_size, input_data, context_wind_shape,
   dims=None, scope_n="layer1"):
"""Implements naive multi dimension recurrent neural networks
@param rnn_size: the hidden units
@param input_data: the data to process of shape [batch,h,w,channels]
@param context_wind_shape: [height,width] of the windows
```

```
@param dims: dimensions to reverse the input data,eg.
dims=[False,True,True,False] => true means reverse dimension
@param scope_n : the scope
returns [batch,h/sh[0],w/sh[1],channels*sh[0]*sh[1]] the output of the lstm
"""
with tf.variable_scope("MultiDimensionalLSTMCell-" + scope_n):
cell = MultiDimensionalLSTMCell(rnn_size)

# shape = input_data.get_shape().as_list()

# Get the symbolic shape of the data
# shape: (batch_size, height, width, input_dim)
shape = tf.shape(input_data)
batch_size, h, w, inp_dim = tf.unstack(shape)

# Add padding if the height and width is not evenly divisible by the context
   window sizes
# Symbolic pad function
def pad(value, axis, context_size):
shp = tf.shape(value)
m = tf.mod(shp[axis], context_size)
pad_amt = tf.cond(
tf.not_equal(m, 0),
lambda: context_size - m,
lambda: tf.constant(0)
)

pad_shape = tf.unstack(shp)
pad_shape[axis] = pad_amt

padding = tf.zeros(shape=pad_shape)

return tf.concat([value, padding], axis=axis)

# Pad the height
input_data_padded = pad(input_data, 1, context_wind_shape[0])

# Pad the width
input_data_padded = pad(input_data_padded, 2, context_wind_shape[1])

# Calculate the reduced height and width dimensions to account for context
```

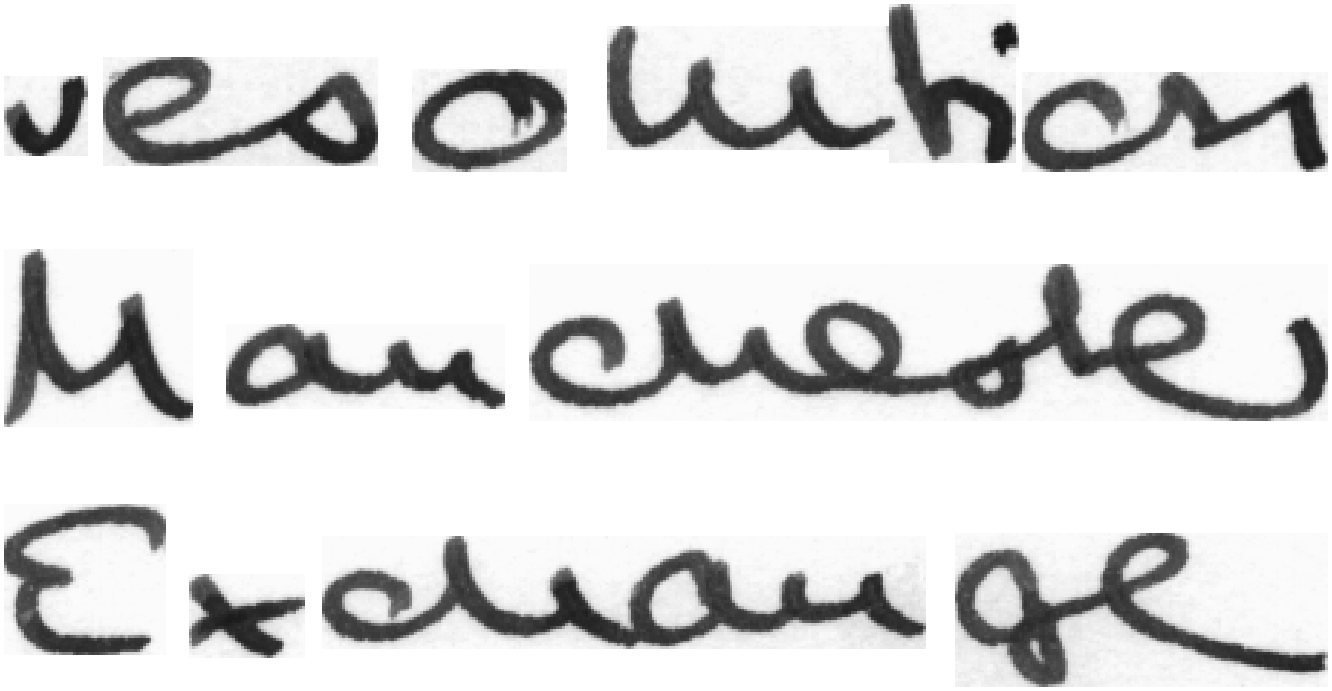
```
   windows
h_red, w_red = tf.unstack(tf.shape(input_data_padded[0, :, :, 0]))
h_red, w_red = h_red / context_wind_shape[0], w_red / context_wind_shape[1]

# Recalculate the feature dimension to account for the size of context window
context_features_size = context_wind_shape[1] * context_wind_shape[0] *
   input_data_padded.get_shape().as_list()[-1]

# Reshape input data to group the features in a context window
x = tf.reshape(input_data, [batch_size, h_red, w_red, context_features_size])

# Perform reversing of dimensions
if dims is not None:
assert dims[0] is False and dims[3] is False
x = tf.reverse(x, dims)

# Shuffle dimensions to look like (height, width, batch_size,
   context_wind_feature_size)
x = tf.transpose(x, [1, 2, 0, 3])
x = tf.reshape(x, [-1, batch_size, context_features_size])
# x = tf.split(axis=0, num_or_size_splits=h * w, value=x)

# sequence_length = tf.ones(shape=(batch_size,), dtype=tf.int32) * shape[0]
inputs_ta = tf.TensorArray(dtype=tf.float32, size=h_red * w_red,
   name='input_ta')
inputs_ta = inputs_ta.unstack(x)
states_ta = tf.TensorArray(dtype=tf.float32, size=h_red * w_red + 1,
   name='state_ta', clear_after_read=False)
outputs_ta = tf.TensorArray(dtype=tf.float32, size=h_red * w_red,
   name='output_ta')

# initial cell and hidden states
states_ta = states_ta.write(h * w, LSTMStateTuple(tf.zeros([batch_size,
   rnn_size], tf.float32),
tf.zeros([batch_size, rnn_size], tf.float32)))

# helper methods for getting the index of the above state and the left state
def get_up(t_, w_):
# state above is one row back from the current index
return t_ - tf.constant(w_)
```

```
def get_last(t_, w_):
# state to the left is just one index back from the current index
return t_ - tf.constant(1)

# initialize step counters for the multi-dimensional while loop
time = tf.constant(0)
zero = tf.constant(0)

# body of while loop
def body(time_, outputs_ta_, states_ta_):

# if not in first row, state up is one row back. otherwise it's the 0 state we
   have a placeholder for
state_up = tf.cond(tf.less_equal(tf.constant(w_red), time_),
lambda: states_ta_.read(get_up(time_, w_red)),
lambda: states_ta_.read(h_red * w_red))

# if not the first column, state is one index back. otherwise it's the 0 state
   we hav ea placeholderfor
state_last = tf.cond(tf.less(zero, tf.mod(time_, tf.constant(w_red))),
lambda: states_ta_.read(get_last(time_, w_red)),
lambda: states_ta_.read(h_red * w_red))

# Combine the cell states for the up and left states into a tuple
current_state = state_up[0], state_last[0], state_up[1], state_last[1]

# Run the multi-dimensional cell step
out, state = cell(inputs_ta.read(time_), current_state)

# Write the output and the cell state
outputs_ta_ = outputs_ta_.write(time_, out)
states_ta_ = states_ta_.write(time_, state) # bc multi-dim, need to actually
   record states

return time_ + 1, outputs_ta_, states_ta_

# while loop conditional
def condition(time_, outputs_ta_, states_ta_):
return tf.less(time_, tf.constant(h_red * w_red))

result, outputs_ta, states_ta = tf.while_loop(condition, body, [time,
```

```
   outputs_ta, states_ta],
parallel_iterations=1)

# stack the outputs and states
outputs = outputs_ta.stack()
states = states_ta.stack()

# reshape (do we need this?)
y = tf.reshape(outputs, [h_red, w_red, batch_size, rnn_size])

# put the batch back as the first dimension
y = tf.transpose(y, [2, 0, 1, 3])

# reverse back on dims if we had reversed originally
if dims is not None:
y = tf.reverse(y, dims)

# returns the hidden outputs and the states
return y, states
```

ДОДАТОК Б

Приклади зображень з датасету IAM



ДОДАТОК В

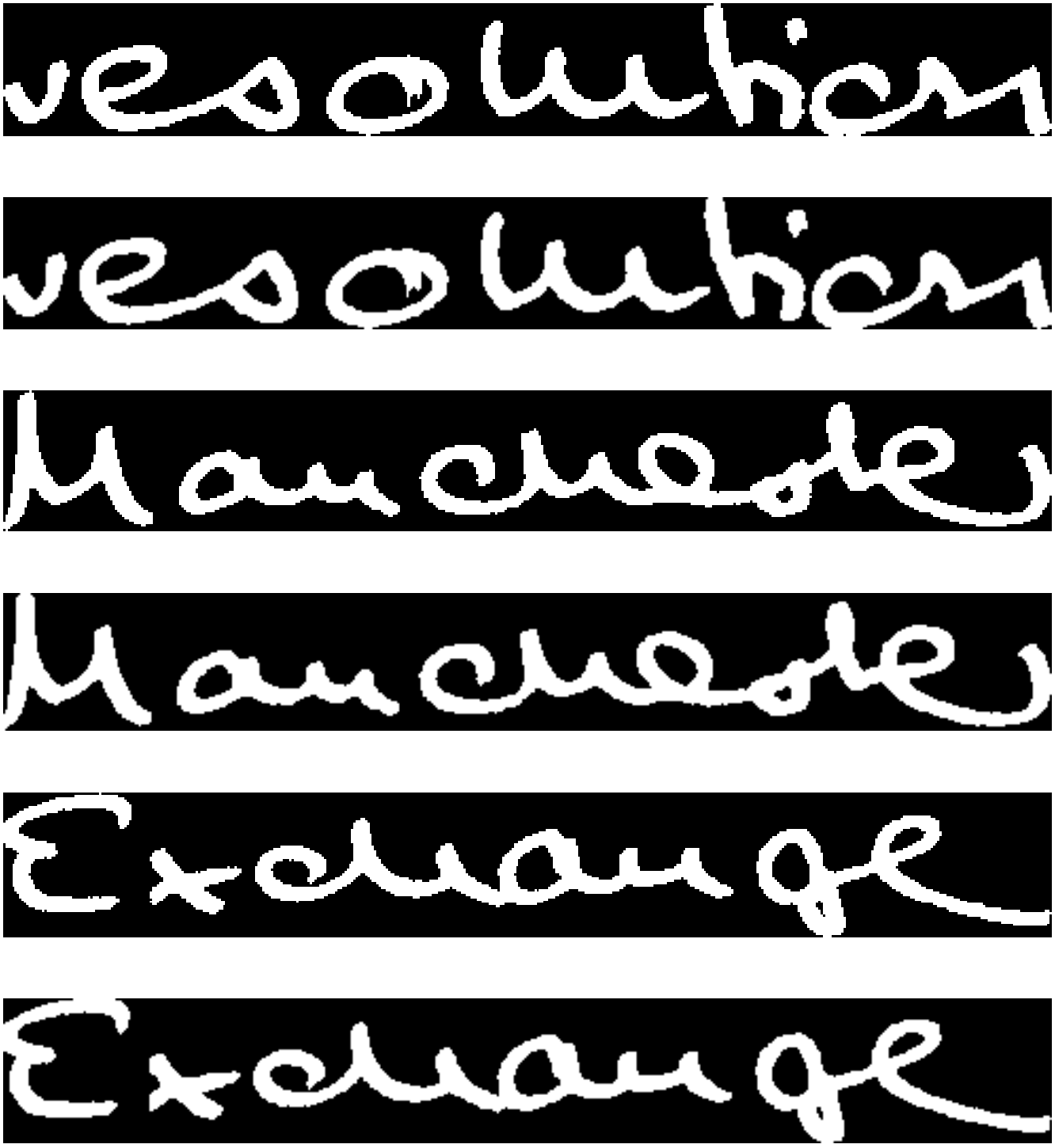
Приклади Бінаризації зображень

ДОДАТОК Г

Приклади усунення шуму

ДОДАТОК Д

Приклади усунення шуму бінаризованих зображень



ДОДАТОК Е

Приклади скелетонізації