МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО

«_____«_____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-КВАЛІФІКАЦІЙНОГО РІВНЯ
«МАГІСТР»

Тема: «Програмний модуль розпізнавання голосових повідомлень з використанням нейронної мережі за технологією глибокого навчання»

Виконавець: Владислав МУРАШКО

Керівник: Дмитро КУЧЕРОВ

Нормоконтролер: Євгеній ТУПОТА

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра Комп'ютеризованих систем управління

Освітньо-кваліфікаційний рівень магістр
Спеціальність 123 "Системне програмування"
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Олександр ЛИТВИНЕНКО
«_____»_____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Мурашка Владислава Дмитровича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема проекту (роботи): «Програмний модуль розпізнавання голосових повідомлень з використанням нейронної мережі за технологією глибокого навчання»
затверджена наказом ректора від «16» вересня 2022 року №1530/ст.

2. Термін виконання проекту (роботи): з 06.09.2022 до 30.11.2022

3. Вихідні дані до проекту (роботи): мова програмування C#, Python 3, середовище розробки Visual Studio 2019, бібліотека TensorFlow

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1) аналіз доцільності голосового розпізнавання;
2) розпізнавання нейромережами за технологією глибокого навчання;
3) Проектування системи розпізнавання;
4) Тестування системи розпізнавання голосових повідомлень

5. Перелік обов'язкового графічного матеріалу:
1) алгоритм голосового розпізнавання із застосуванням нейромережі глибокого навчання;
2) екранна форма інтерфейсу системи розпізнавання об'єктів на зображенні із застосуванням нейромереж глибокого навчання;
3) Екранна форма результату навчання;
4) UML діаграма класів програмної системи розпізнавання об'єктів.

6. Календарний план-графік

№ п/п	Етапи виконання дипломного проекту	Термін виконання етапів	Примітка
1	Ознайомитись з постановкою задачі дипломного проектування	06.09.22	
2	Вивчити спеціальну літературу і технічну документацію	07.09.22	
3	Проаналізувати поняття нейромереж – основні поняття і визначення	12.09.22	
4	Написати розділ 1.	19.09.22	
5	Проаналізувати алгоритми голосового розпізнавання об’єктів із застосуванням нейромереж глибокого навчання. Спроектувати програмний модуль.	23.09.22	
6	Написати розділ 2.	05.10.22	
7	Виконати розробку програмної системи розпізнавання об’єктів на зображенні із застосуванням згорткових нейромереж	14.10.22	
9	Оформити пояснювальну записку	11.11.22	
10	Підготовити графічний демонстраційний матеріал	20.11.22	

7. Дата видачі завдання «06» вересня 2022 р

Керівник кваліфікаційної роботи _____ Дмитро КУЧЕРОВ
(підпис)

Завдання прийняв до виконання _____ Владислав МУРАШКО
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмний модуль розпізнавання голосових повідомлень з використанням нейронної мережі за технологією глибокого навчання»: 60 с., 30 рис., 19 використаних джерел, 1 додаток.

НЕЙРОННІ МЕРЕЖІ, *C#, PYTHON* 3, *RUST,* ФРЕЙМВОРК, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, *TENSORFLOW,* ГЛИБОКЕ НАВЧАННЯ

Мета дослідження – розробити програмний модуль для розпізнавання голосових повідомлень з використанням нейронної мережі за технологією глибокого навчання.

Об’єкт дослідження – опанування глибокого навчання та нейронних мереж.

Предмет дослідження – програмний модуль для розпізнавання голосових повідомлень з використанням нейронної мережі за технологією глибокого навчання.

Результатом виконання кваліфікаційної роботи є створений програмний засіб для розпізнавання голосових повідомлень з використанням нейронної мережі за технологією глибокого навчання, який забезпечує виконання команд за допомогою голосу. Додаток орієнтований на всі групи користувачів.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ММН – методи машинного навчання

МГН – методи глибокого навчання

ДКЧП – довга короткочасна пам’ять

НМ – нейронні мережі

ЗНМ – згорткові нейронні мережі

ОС – операційна система

КІ – користувацький інтерфейс

КД – користувацький досвід

ВСТУП

Сучасні інформаційні технології поступово розвиваються у всіх аспектах та напрямках. Цей процес впливає на наші повсякденні речі, наприклад, на бортовий автомобільний комп'ютер для збору й аналізу даних, надсилання інформації, також сприяє покращенню процесу обслуговування автомобіля. Ще один приклад це «розумний» будильник, який використовує додаткові датчики для аналізу фаз сну та вибирає оптимальний час для пробудження користувача. Достатньо перспективним є додавання голосового інтерфейсу до комп'ютеризованих систем, контроль промислової та побутової техніки.

Одна з актуальних проблем, яку необхідно вирішити при розробці таких систем управління є проблема недостатньої точності розпізнавання голосових команд. Удосконалення зроблено в напрямку підвищення надійності, незалежності від індивідуальні особливості голосу, знижує негативний вплив фону та шуму на якість розпізнавання. Попит на такі системи зараз обмежений через недостатньо високі параметри та проблеми, пов'язані з темою безпеки голосування. Після аналізу попиту на використання голосового керування в домогосподарстві, наприклад промислових систем було вирішено спроектувати систему, який володіє достатніми параметрами для сприйняття та обробки мови.

Щоб підвищити точність розпізнавання голосу системи було вирішено використовувати глибокі нейронні мережі (DNN), які в останні роки неодноразово демонстрував значні результати в процесах прогнозування, класифікації, розпізнавання образів, почерку і мови. Оскільки використання ГНМ та їх модифікацій у задачах розпізнавання мовлення є актуальним завданням сьогодення.

Метою дослідження є підвищення точності розпізнавання голосу за допомогою існуючих нейромережевих методів і методів, які є запропоновані в роботі.

Темою дослідження в цій роботі є процес розпізнавання голосу запропонований метод і розробка чат-бота під голосовим керуванням.

Методи дослідження. Поставлені у роботі задачі вирішувалися шляхом проведення теоретичних та експериментальних досліджень. При аналізі можливостей збільшення показників точності в розпізнаванні голосу використано основні положення математичного аналізу та комп'ютерних технологій Google App Engine, DialogFlow, TensorFlow та платформи Google Cloud.

РОЗДІЛ 1. АНАЛІЗ ДОЦІЛЬНОСТІ ГОЛОСОВОГО РОЗПІЗНАВАННЯ

1.1. Завдання голосового розпізнавання

Чи актуальні системи розпізнавання голосу в 2022 році? Техніки розпізнавання мовлення все більше стають частиною нашого життя та є практичним інструментом, управління різними електронними пристроями - регулятор гучності. На жаль, розпізнавання мовлення є дуже ресурсомістким завданням і вимагає багато обчислювальних потужностей, доступ до яких часто обмежений (наприклад, на мобільних пристроях). Крім того, процес ідентифікації займає багато часу, тоді як кінцеві користувачі пристроїв очікують швидкої реакції системи.

Розпізнавання голосу або динаміка є функцією пристрою або програмного забезпечення приймати та інтерпретувати диктовку або розуміти та виконувати сказане. Багаторівневе розпізнавання голосу набуло популярності та використання з розвитком ШІ та інтелектуальних помічників, такі як Google Assistant, Amazon Alexa, Apple Siri та Microsoft Cortana.

Системи розпізнавання голосу дозволяють споживачам взаємодіяти з технологіями, просто звернувшись до неї, зробивши запити в режимі hands-free, наприклад нагадування та інші прості завдання. Сучасні системи голосового керування обробляють і перетворюють мову з високою точністю. Для цього в алгоритмі обробки потрібно відзначити ключові слова Сьогодні сучасні інструменти дозволяють встановлювати ключові слова з використанням програмного забезпечення без апаратної прошивки.

Сучасне голосове управління може бути безконтактним, тобто для наведення пристрою у режимі для отримання голосових команд потрібно натиснути певну кнопку в програмі для мобільного телефону, ПК, приймального модулю. Однак є новіші версії програм голосового керування з можливістю введення ключового слова для переведення пристрою в режим прийому голосових команд або вже діючих команд.

1.2. Типи систем голосового розпізнавання

Розпізнавання голосу стане ключовою частиною спілкування в майбутньому. Незалежно від того, чи то запитує Alexa про час, чи навігацію системою бізнес-телефону, ви стикалися з цим раніше.

Багато компаній використовують цей новий спосіб роботи, чи то для вдосконалення власних внутрішніх процесів, чи для оновлення систем обслуговування клієнтів. Незважаючи на це, розпізнавання мовлення все ще відносно нове, і багато людей залишаються скептичними щодо того, що воно робить і як його можна використовувати.

У цьому посібнику ми обговоримо, що таке розпізнавання голосу, де його можна використовувати, які переваги воно має та чому вам варто його використовувати, якщо ви власник бізнесу.

Що таке розпізнавання голосу?

Завдяки сучасним технологіям комп’ютерне програмне забезпечення тепер може розуміти мовлення. Це програмне забезпечення може слухати те, що ви говорите, і інтерпретувати це в оцифровану версію, яка читає та аналізує.

Отже, як він це робить? Через штучний інтелект і машинне навчання. Великі обсяги даних використовуються для створення алгоритму, який можна розробити з часом. Потім штучний інтелект вивчає ці дані та визначає закономірності. Він переглядає попередні введення та фіксує те, що ви говорите. Він навіть може зрозуміти, як ви говорите, наприклад, як ви використовуєте регіональну мову.

Розпізнавання голосу означає, що ваш мобільний пристрій, розумні колонки або комп’ютер можуть слухати те, що ви говорите. Ця розширена функціональність може стати в нагоді, коли вам потрібна допомога по дому, наприклад запитайте у Amazon Alexa, якою буде сьогодні погода, щоб дізнатися, чи потрібна вам парасолька, перш ніж вирушити на роботу. Його також можна використовувати для диктування нотаток, коли у вас немає часу або фізичних засобів, щоб записати їх.

Багато компаній також використовують його для покращення обслуговування клієнтів. Абоненти можуть відповісти на певні запитання та бути направленими до потрібного агента для вирішення їхньої проблеми. Це технологія розпізнавання голосу RingCentral. Це покращує частоту вирішення першого дзвінка та гарантує, що вашим агентам не доведеться переадресовувати дзвінки в інші відділи. Це чудово для клієнтів, які швидко й ефективно вирішують свої проблеми, і чудово для бізнесу, щоб підвищити продуктивність і прийняти більше дзвінків.

Як працює розпізнавання голосу

Отже, як працює розпізнавання голосу? Що ж, він використовує технологію для оцінки біометрії вашого голосу. Це включає в себе частоту і потік вашого голосу, а також ваш акцент. Кожне слово, яке ви говорите, розбивається на сегменти з кількох тонів. Потім це оцифровується та перекладається для створення власного унікального голосового шаблону.

Штучний інтелект, глибоке навчання та машинне навчання — це сили, що стоять за розпізнаванням мовлення. Штучний інтелект використовується для розуміння розмовних слів, абревіатур і акронімів, які ми використовуємо. Потім машинне навчання збирає разом шаблони та розвиває ці дані за допомогою нейронних мереж.

Цю технологію можна використовувати для різноманітних систем, деякі складніші за інші. Наприклад, якщо ви коли-небудь дзвонили в контакт-центр свого оператора мобільного зв’язку, вас може зустріти меню з розпізнаванням

голосу. Щоб вас направили до потрібного відділу, вам потрібно вибрати варіант. Це можна зробити, промовивши номер або використовуючи клавіатуру.

Але розпізнавання голосу може зробити набагато більше, ніж це. Візьмемо, наприклад, Alexa. Цей розумний домашній помічник може відповісти на запитання, відтворити музику та вимкнути світло у вашому домі за допомогою вашого голосу.

Використання розпізнавання голосу

На сьогоднішній день 72% людей, які користуються пристроями голосового пошуку, стверджують, що вони стали частиною їх повсякденних справ. Технології швидко розвиваються, і іноді наступна «велика річ» затьмарюється новою розробкою. Але чим більше людей почуваються комфортно розмовляти по телефону та смарт-хабу, тим більше ця тенденція набирає популярності.

Це також не лише для особистого користування. У міру того, як індустрії та підприємства підключаються, тенденція використання розпізнавання голосу – лише питання часу, коли ця кількість зросте. Все більше компаній використовують системи розпізнавання голосу, щоб допомогти їм з ефективністю та точністю в обслуговуванні клієнтів.

Ось деякі з основних застосувань розпізнавання голосу:

Диктування. Технологію розпізнавання мовлення можна використовувати різними способами. Зараз багато галузей використовують розпізнавання голосу, щоб допомогти в повсякденних процесах. Наприклад, юридична галузь отримала велику користь від розпізнавання голосу. Юристи використовують його для диктування важливих зустрічей, які потім можуть записувати в документи. Це не тільки економить їхній час, але й забезпечує точний запис усієї інформації.

Це також допомагає у звичайних повсякденних заняттях. У багатьох із нас є смартфони чи домашні концентратори, які також мають віртуального помічника, і ви можете диктувати свій список покупок, щоденні завдання та майже все, що ви хочете зробити нотаткою. Це легше, а часто й продуктивніше, ніж писати це самостійно.

Доступність. Розпізнавання голосу також можна використовувати в зворотному порядку, тобто замість перетворення мови в текст ви можете перекладати текст у мову. Деякі платформи, такі як Dragon Professional від Nuance, пропонують цю функцію. Багато людей, які мають проблеми з промовою та зором, наприклад, люди з обмеженими можливостями чи вадами мови, вважають його корисним. З цієї причини його також можна використовувати в освітньому секторі.

Покупки за допомогою голосової команди. Понад 55% клієнтів придбали продукт на веб-сайті електронної комерції за допомогою розпізнавання мовлення. І, оскільки більше людей знайомляться з технологією розпізнавання голосу, це число може зрости.

Приклади систем з розпізнаванням голосу:

- Автоматизовані телефонні системи;
- Google Voice;
- Цифровий помічник;
- Автомобільний Bluetooth;

Автоматизовані телефонні системи. На робочому місці автоматизовані телефонні системи стають все більш поширеними. Візьмемо, наприклад, RingCentral Office. Ця хмарна телефонна платформа містить функцію IVR (інтерактивна голосова відповідь). Коли клієнт дзвонить, машина використовує автоматичне розпізнавання мови, щоб зрозуміти, що клієнт говорить. Потім він може спрямувати їх на голосову пошту, на внутрішній номер і навіть на

зовнішні номери. У будь-який час можна ввімкнути до 250 меню, що ідеально підходить для великих глобальних компаній.

Google Voice. Якщо ви скажете «Hey Google» на своєму пристрої Android, помічник Google Voice допоможе. Подібно до Cortana та Siri від Apple, ви можете запропонувати їй шукати різні теми, але ця система спрямовує користувачів до пошукової системи Google. Це також працює з «Google Next», найновішою розумною колонкою від Google. Більше того, ви можете точно перетворювати текст на мовлення за допомогою API на базі технології Google.

Цифровий помічник. Багато розумних пристроїв мають власного цифрового помічника. Якщо у вас є пристрій Apple, ви, ймовірно, чули про «Siri». Siri – це персональний помічник, який може розпізнати ваш голос. Ви можете попросити Siri знайти для вас запитання, надіслати комусь SMS і навіть відтворити вашу улюблену пісню. Серед інших цифрових помічників – Alexa, Cortana та Bixby.

Автомобільний Bluetooth. Наявність автомобільного Bluetooth – це не тільки зручно, але й крок у безпеці. Там, де водії могли надіслати текстове повідомлення за кермом, тепер вони можуть підключитися до свого автомобіля через Bluetooth і надіслати текстове повідомлення в режимі «вільні руки» за допомогою розпізнавання мовлення.

1.3. Принципи розпізнавання голосу

1.3.1 Приховані марковскі моделі

Стохастичний процес — це набір випадкових величин, які індексуються деякими математичними наборами. Тобто кожна випадкова величина стохастичного процесу однозначно пов’язана з елементом у наборі. Набір, який використовується для індексування випадкових змінних, називається набором індексів, а набір випадкових змінних утворює простір станів. Стохастичний процес можна класифікувати різними способами на основі простору станів, набору індексів тощо.

Коли стохастичний процес інтерпретується як час, якщо процес має кінцеву кількість елементів, таких як цілі числа, числа та натуральні числа, тоді це дискретний час.

Стохастична модель. Це процес з дискретним часом, індексований у момент часу 1, 2, 3,…, який приймає значення, які називаються станами, які спостерігаються.

Наприклад, якщо стани (S) ={hot , cold }

Ряд стану за часом => z∈ S_T

Погода на 4 дні може бути послідовною => {z1=жарко, z2 =холодно, z3 =холодно, z4 =жарко}

Моделі Маркова та приховані моделі Маркова створені для обробки даних, які можна представити як «послідовність» спостережень у часі. Приховані марковські моделі — це імовірнісні структури, де спостережувані дані моделюються як серія вихідних даних, створених одним із кількох (прихованих) внутрішніх станів.

Маркова припущення. Моделі Маркова розроблені на основі головним чином двох припущень:

1. Припущення обмеженого горизонту: ймовірність перебування в стані в момент часу t залежить лише від стану в момент часу (t-1).

$$P\left(z_t, z_{t-1}, z_{t-2}, \cdots \cdot z_1\right) = P(z_t / z_{t-1})$$

Це означає, що стан у момент часу t є достатнім підсумком минулого, щоб розумно передбачити майбутнє. Це припущення є марковським процесом порядку 1. Марковський процес порядку k припускає умовну незалежність стану z_t від станів, які перед ним мають k + 1-кратний крок.

2. Припущення стаціонарного процесу: умовний (імовірнісний) розподіл за наступним станом, враховуючи поточний стан, не змінюється з часом.

$$P(z_t / z_{t-1}) = P(z_2 / z_1) \quad ----------> t \epsilon 2 \ldots T$$

Це означає, що стани продовжують змінюватися з часом, але основний процес залишається нерухомим.

Окрім того, правильне застосування цих моделей для рішення деяких важливих прикладних задач призводить до дуже гарних результатів. Тим не менш, широке розповсюдження ПММ в задачі розпізнавання мови отримали відносно нещодавно: початково основи теорії прихованих марковських моделей були опубліковані в журналах для математиків, які не були дуже популярними серед інженерів, котрі займалися розпізнаванням мови; також, опублікована теорія не містила в собі пояснень про можливості і способах застосування ПММ в різних прикладних областях. Зараз ці моделі дуже популярні. Тим не менш крім мовного сигналу ланцюг Маркова побудований як однонаправленому процесі переходу між станами в різний час, який можливий перехід до наступного стану залежить лише від поточного стану і не залежить від того яким був процес у минулі часи. Ця вимога призводить до неправильних гістограм тривалості життя станів, а відтак сучасні системи відмовляються від них. Моделі, де ймовірність переходу до наступного стану залежить від часу, проведеного в поточному стані, що називаються гетерогенними марківськими або напівмарківські.

Отже, марківська модель звуку чи слова являє собою єдиний або кілька послідовних станів, де визначені функції щільності ймовірностей в просторі ознак і ймовірностей переходів.

1.3.2 Динамічна деформація часу (DTW)

В аналізі часових рядів динамічний час короблення (DTW) є одним з алгоритмів для вимірювання подібності між двома тимчасовими послідовностями, які можуть змінюватись у швидкості. Наприклад, подібність у ходьбі можна було виявити за допомогою DTW, навіть якщо одна людина йшла швидше за іншу, або якщо в ході спостереження відбувалися прискорення та уповільнення. DTW був застосований до тимчасових послідовностей відео, аудіо та графічних даних - дійсно, будь-які дані, які можна перетворити на лінійну послідовність, можна проаналізувати за допомогою DTW. Добре відомим додатком було автоматичне розпізнавання мови, щоб впоратися з різними мовними швидкостями. Інші програми включають розпізнавання виступаючих і розпізнавання підпису онлайн. Його також можна використовувати у додатках часткового узгодження форми.

Загалом, DTW - це метод, який обчислює оптимальну відповідність між двома заданими послідовностями (наприклад, часовими рядами) з певними обмеженнями та правилами:

- Кожен індекс з першої послідовності повинен збігатися з одним або декількома індексами з іншої послідовності та навпаки.
- Перший індекс із першої послідовності повинен збігатися з першим індексом з іншої послідовності (але це не обов'язково має бути його єдиним збігом)
- Останній індекс із першої послідовності повинен збігатися з останнім індексом з іншої послідовності (але це не обов'язково має бути його єдиним збігом)
- Відображення індексів з першої послідовності індекси з іншої послідовності має бути монотонно зростаючим, і навпаки, тобто. якщо це індекси з першої послідовності, тоді не повинно бути двох індексів в іншій послідовності, так що індекс буде зіставлений з індексом і індекс зіставляється з індексом , і навпаки;

Матч оптимальна позначається матч, який задовольняє всі обмеження та правила, і що має мінімальну вартість, де вартість обчислюється як сума абсолютних різниць, для кожної узгодженої пари індексів, між їх значеннями.

Послідовності "спотворюються" нелінійно в часовому вимірі, щоб визначити міру їхньої подоби незалежно від деяких нелінійних змін у часовому вимірі. Цей метод вирівнювання послідовностей часто використовується під час класифікації часових рядів. Хоча DTW вимірює величину, подібну до відстані між двома заданими послідовностями, він не гарантує виконання нерівності трикутника.

На додаток до вимірювання подібності між двома послідовностями створюється так званий «шлях деформації», за допомогою деформації відповідно до цього два сигнали можуть бути вирівняні в часі. Сигнал з вихідним набором точок X (вихідний), Y (вихідний) перетворюється на X (деформований), Y (деформований). Це знаходить застосування в генетичній послідовності та синхронізації звуку. У спорідненій методиці послідовності зі змінною швидкістю можуть бути усереднені з використанням цього методу, див. Розділ середньої послідовності.

Концептуально це дуже схоже алгоритм Нідлмана – Вунша.

1.3.3 Штучні нейронні мережі

Штучний інтелект (AI) досліджує та створює інтелектуальне програмне забезпечення та машини, надає конкретне рішення конкретної визначеної складної проблеми. Для виконання цих завдань він використовує такі моделі та алгоритми, як генетичні алгоритми, оптимізація роїв частинок, штучні нейронні мережі (ШНН) і гібридні моделі (дві або більше з перерахованих вище). ШНМ є гнучкими для розміщення нелінійних і нефізичних даних; однак вони потребують великого багатовимірного набору даних, щоб зменшити ризик екстраполяції. ШНМ широко використовуються в різних галузях науки через їх здатність до навчання та адаптованість до різних умов.

ШНМ — це алгоритм ML, заснований на концепції людського нейрона. Це біологічно інспірована обчислювальна модель, що складається з елементів обробки (нейронів) і зв’язків між ними з коефіцієнтами (вагами), прикріпленими до зв’язків. ШНМ надихаються структурою мозку, і з цієї причини важливо визначити основні компоненти, під якими працюють нейрон, дендрити, тіло клітини та аксон. Дендрити — це мережа, яка передає електричні сигнали до тіла клітини. Тіло клітини додає та збирає сигнали. Аксон переносить сигнал від тіла клітини до інших нейронів за допомогою довгого волокна. Коли аксон клітини вступає в контакт з дендритом іншої клітини, це називається синапсом. Тому функції нейронних мереж встановлюються через розташування нейронів та окремих синаптичних сил. На рисунку 1.1 представлена загальна схема біологічного нейрона з кожним елементом, який його складає.

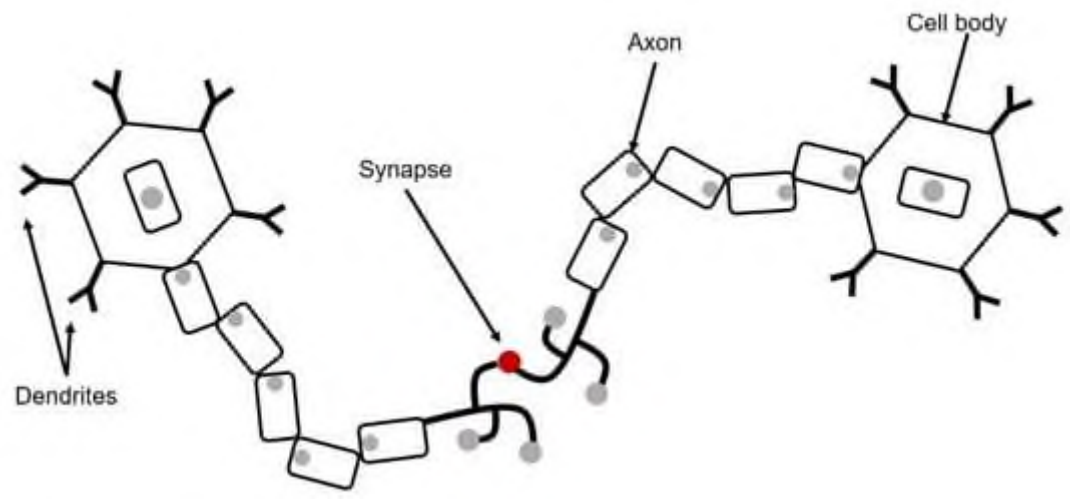


Рис. 1.1 Загальна схема біологічного нейрона

Нейронні структури розвиваються через навчання; однак вони постійно змінюються, посилюючи або послаблюючи синаптичні з’єднання. Хоча ШНМ надихаються мозком, вони не такі складні. Однак найбільша подібність полягає насамперед у тому, що обидві мережі взаємопов’язані, а функції мереж визначаються зв’язками між нейронами.

Нейрони отримують такі вхідні дані, як імпульси. Пікова швидкість, що генерується протягом певного часу, і середня швидкість генерації піків за

кілька прогонів – це деякі показники, які використовуються для опису активності нейронів. У ШНМ нейрон ідентифікується за швидкістю, з якою він генерує ці піки. Нейрон з’єднується з іншими нейронами попереднього шару через адаптивні синаптичні ваги. Знання, як правило, зберігаються як набір ваг зв’язків. Коли ці ваги з’єднань модифікуються впорядкованим чином і за допомогою відповідного методу навчання, здійснюється процес навчання. Метод навчання складається з представлення входу в мережу та бажаного виходу, коригування ваг таким чином, щоб мережа могла виробляти бажаний результат. Після тренування ваги матимуть релевантну інформацію, тоді як до тренування вона зайва і безглузда.

На рисунку 1.2 представлена проста структура нейрона. Обробка інформації в нейроні починається з вхідних даних Xn, вони зважуються та додаються перед тим, як пройти через деяку функцію активації для генерації її виходу, цей процес представлений як $\xi = \Sigma Xi \cdot Wi$. Для кожного з вихідних з'єднань це значення активації множиться на питому вагу Wn і передається наступному вузлу. Якщо він розглядає лінійну активацію, вихід буде дано як $y=\alpha(wx+b)$.

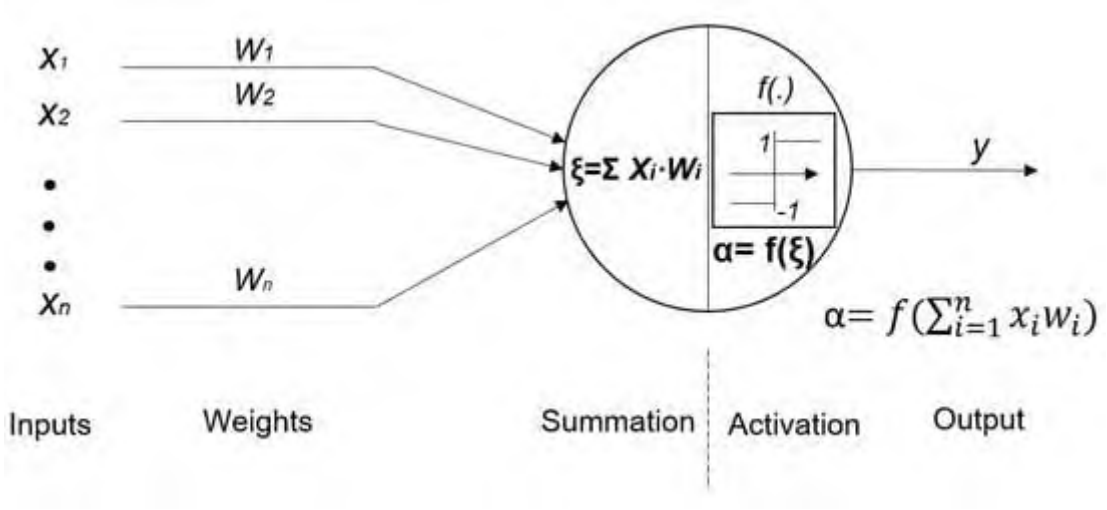


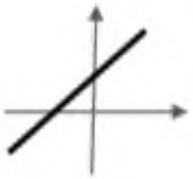
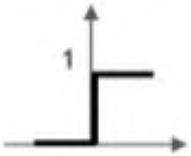
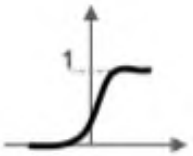
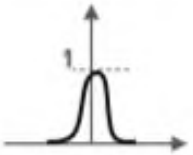
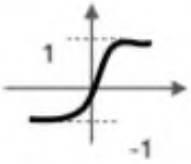
Рис. 1.2 Базова схема нейрона

Функція активації — це функція, яка отримує вхідний сигнал і створює вихідний сигнал після того, як вхід перевищить певний поріг. Тобто нейрони приймають сигнали та генерують інші сигнали. Запуск нейрона виконується

лише тоді, коли сума загальних вхідних сигналів перевищує порогову межу нейрона, тоді вихідні дані будуть передані іншому нейрону або середовищу. Ця порогова межа визначає, чи активований нейрон чи ні, найпоширенішими функціями активації або передачі є лінійна, бінарна ступінчаста, кусково-лінійна, сигмоподібна, гауссова та гіперболічна дотична функції .

Таблиця 1.1 показує функції активації, які зазвичай використовуються в мережах мереж. Поведінка нейронів визначається цими функціями. Якщо він передає функцію, яка є лінійною, а мережа є багатошаровою, її можна представити як одношарову мережу, оскільки вона є добутком вагових матриць кожного шару та створюватиме лише додатні числа у всьому діапазоні дійсних чисел. З іншого боку, нелінійні функції передачі (сигмоїдна функція) між шарами дозволяють декільком шарам надавати нові можливості, регулюючи вагові коефіцієнти для отримання мінімальної помилки в кожному наборі з’єднань між шарами. Лінійні функції зазвичай використовуються у вхідному та вихідному рівнях, тоді як нелінійні функції активації можуть використовуватися для прихованого та вихідного рівнів.

Таблиця 1.1

Name	Graphic	Function
Linear		$\varphi(\varphi) = \varphi \cdot \varphi + \varphi$
Binary step		if $\varphi \geq 0$, then $\varphi(\varphi) = 1$, if $\varphi < 0$, then $\varphi(\varphi) = 0$,
Piecewise linear		if $\varphi \geq \varphi_{\varphi\varphi\varphi}$, then $\varphi(\varphi) = 1$, if $\varphi_{\varphi\varphi\varphi} > \varphi >$ $\varphi_{\varphi\varphi\varphi}$, then $\varphi(\varphi) = \varphi \cdot \varphi +$ φ, if $\varphi \leq \varphi_{\varphi\varphi\varphi}$, then $\varphi(\varphi) = 0$,
Sigmoid		$\varphi(\varphi) = \frac{1}{1+\varphi^{-\varphi\varphi}}$, interval (0,1)
Gaussian		$\varphi(\varphi) = \varphi^{-\varphi^2}$, interval (0,1]
Hyperbolic tangent		$\varphi(\varphi) = \frac{2}{1+\varphi^{-2\varphi}} - 1$, interval [-1,1]

Найбільш використовуваними нелінійними функціями активації є сигмоподібний і гіперболічний тангенси. В основному використовуються гіперболічний та сигмоїдний дотичні, оскільки вони є диференційованими та роблять їх сумісними з алгоритмом зворотного поширення. Обидві функції активації мають криву «S», тоді як їхній вихідний діапазон різний. Сигмоїдна функція є найбільш використовуваною функцією активації в ШНМ. Ця функція змінюється від 0 до +1, хоча функція активації іноді намагається коливатися

між −1 і +1, і в цьому випадку функція активації приймає антисиметричну форму відносно початку координат, визначаючи її як функцію гіперболічного тангенса.

Пряма реалізація функцій сигмоїди та гіперболічного тангенса в апаратному забезпеченні є недоцільною через її експоненціальний характер. Існує декілька різних підходів до апаратної апроксимації функцій активації, таких як кусково-лінійна апроксимація. Апроксимації лінійної частини є повільними, але вони є найпоширенішим способом реалізації функцій активації. Крім того, тут використовується ряд лінійних сегментів для апроксимації тригерної функції. Кількість і розташування цих сегментів вибирається таким чином, щоб помилки і час обробки були мінімізовані.

Функцію активації Гауса можна використовувати, коли потрібен точніший контроль над діапазоном активації. Крім того, він може рівномірно виконувати апроксимації неперервної функції різних змінних.

ШНМ класифікуються за різними критеріями, можна встановити два типи:

- Нейронні мережі прямого зв'язку (FFNN);
- рекурентні нейронні мережі (в дискретному часі) або диференціальні (в безперервному часі);
- Нейронні мережі прямого зв'язку.

Нейрон є основним компонентом НМ. Нейрони з'єднані один з одним за допомогою синаптичної ваги. Розглядаючи нейронну мережу з трьома шарами, такими як на малюнку 3: вхідний рівень, прихований шар і вихідний рівень, проміжний рівень вважається самоорганізованою картою Кохонена, яка складається з двох рівнів блоків обробки (вхідного та вихідного), залежно від складності мережі (у кожній мережі може бути декілька прихованих рівнів). У FFNN інформація просувається від вхідних вузлів до прихованих вузлів і від прихованих вузлів до вихідних вузлів. Коли вхідний шаблон подається в

мережу, одиниці у вихідному шарі конкурують між собою, і виграє вихідний блок, чиї ваги вхідних з’єднань найближчі до вхідного шаблону, кількість нейронів у вхідному та вихідному шарах. дорівнює кількості входів і виходів задачі. Метод навчання можна розділити на два етапи, перший етап полягає у визначенні нейрона прихованого шару, ваговий вектор якого є першим вхідним вектором, а другий відноситься до процесу навчання. Спочатку буде обчислено евклідову відстань між входом і ваговим вектором першого нейрона. Якщо відстань перевищує заздалегідь визначене порогове значення відстані, створюється новий нейрон прихованого шару шляхом призначення вхідних даних як вектора ваги. В іншому випадку вхідний шаблон належить цьому нейрону. Під час навчання кожен шаблон, поданий у мережу, вибирає найближчий нейрон на евклідовій мірі відстані, змінюючи вектор ваги переможця, а топологічні сусіди малюють їх у напрямку вхідних даних, ваги, що залишають нейрон-переможець, і його сусіди коригуються за допомогою метод градієнтного спуску. Прямі НМ поділяються на дві категорії залежно від кількості шарів: одношарові або багаторівневі.

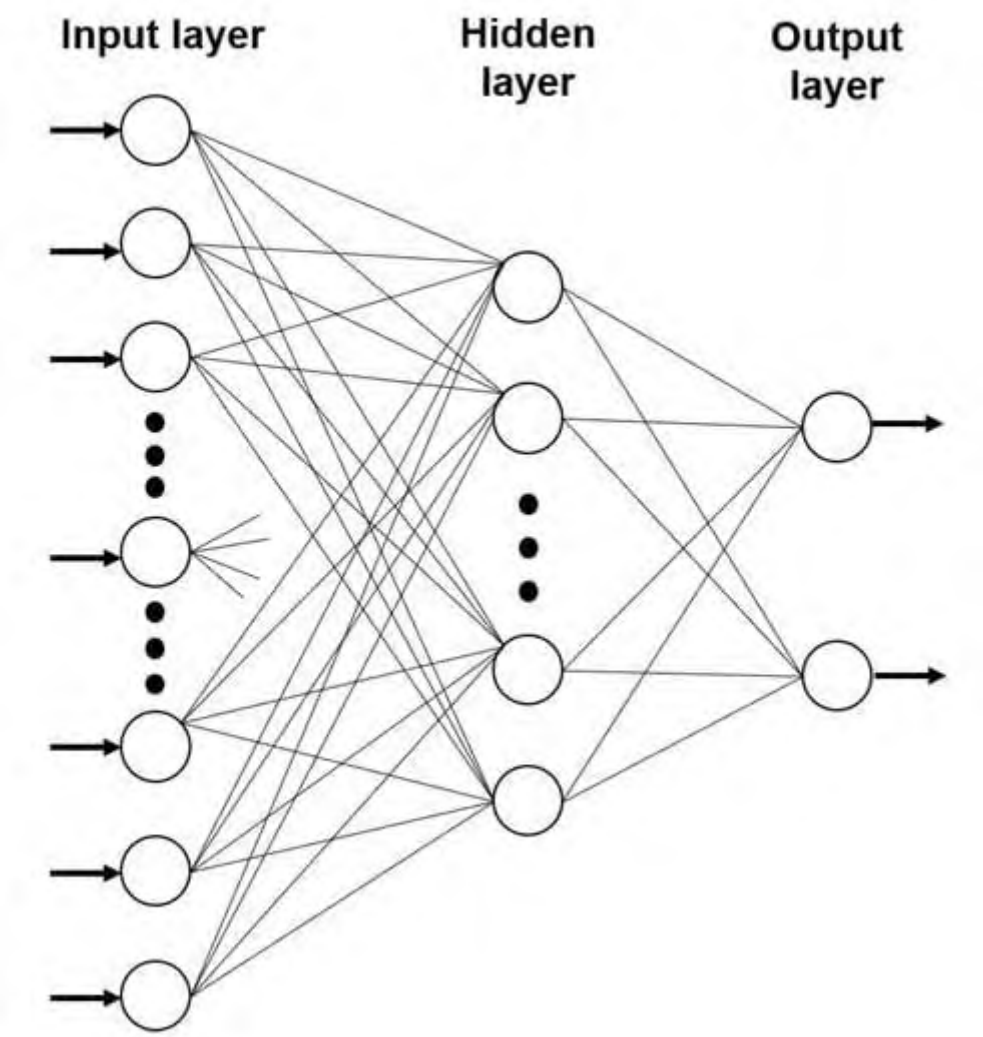


Рис. 1.3 Структура нейронної мережі прямого зв’язку

Зворотне розповсюдження (ВР) — це тип навчання ШНМ, який використовується для реалізації контрольованого навчання, завдань, для яких відома репрезентативна кількість вибіркових вхідних і правильних вихідних даних. ВР виводиться з різниці між бажаним і прогнозованим виходом; це обчислюється та поширюється назад. Спочатку ініціалізуються ваги мережі до невеликої випадкової ваги, представляється векторний набір вхідних даних у мережу, вхідні дані поширюються для генерації виходу, що називається фазою випередження вхідних даних, і помилка порівняння оціненого чистого виходу з розрахований бажаний вихід.

1.4 Показники якості розпізнавання голосу

World Error Rate (WER) є загальноприйнятою метрикою ефективності системи розпізнавання мовлення.

Метод визначення показника WER полягає у вирівнюванні двох текстових рядків – першим є результат розпізнавання, а другим саме той рядок, який мав бути сказаний. Дане вирівнювання реалізовується за допомогою алгоритму динамічного програмування з обчисленням відстані Левенштейна. Відстань Левенштейна представляє собою “вартість” редагування даних (мініальна кількість або зважена сума операцій редагування) для перетворення першого рядка у другий з найменшою кількістю операцій заміни (S), видалення (D) та вставки (I) слів:

$$WER = (S + D + I) / N$$

де N – кількість слів в розпізнаній фразі.

Для оцінювання результатів автоматичного розпізнавання мовлення використовується показник, як відсоток коректно розпізнаних слів (WCR – Word Correctly Recognized), який не враховує помилкові вставки слів, отриманих системою:

$$WCR = H/T * 100\%, H = N – D – S \quad (54)$$

де H – кількість правильно розпізнаних слів.

Висновок

Перший розділ був присвячений огляду задачі розпізнавання мови та голосового управління, способам її рішення. В цьому розділі класифіковано системи розпізнавання мови за різними критеріями, розглянули різні підходи до поставленої проблеми та проаналізували три найбільш успішних методи, які використовуються в сучасних системах розпізнавання мови.

У комп'ютерних науках проблеми штучного інтелекту розглядаються з позицій проектування експертних систем та баз знань. Під базами знань розуміється сукупність даних і правил виведення, що допускають логічний висновок і осмислену обробку інформації.

В цілому дослідження проблем штучного інтелекту в комп'ютерних науках спрямовані на створення, розвиток і експлуатацію інтелектуальних інформаційних систем, а питання підготовки користувачів і розробників таких систем вирішуються фахівцями інформаційних технологій.

РОЗДІЛ 2. РОЗПІЗНАВАННЯ НЕЙРОМЕРЕЖАМИ ЗА ТЕХНОЛОГІЄЮ ГЛИБОКОГО НАВЧАННЯ

2.1 Типова структура нейромережі з глибоким навчанням

Архітектури коннекціонізму існують більше 70 років, але нові архітектури та графічні процесори (GPU) вивели їх на передній план штучного інтелекту. Глибоке навчання — це не єдиний підхід, а радше клас алгоритмів і топологій, які можна застосувати до широкого спектру проблем.

Хоча глибоке навчання, безумовно, не є чимось новим, воно переживає вибуховий розвиток через перетин багаторівневих нейронних мереж і використання графічних процесорів для прискорення їх виконання. Великі дані також сприяли цьому зростанню. Оскільки глибоке навчання ґрунтується на навчанні нейронних мереж прикладами даних і винагородою за їхній успіх, чим більше даних, тим краще побудувати ці структури глибокого навчання.

Кількість архітектур і алгоритмів, які використовуються в глибинному навчанні, є великою та різноманітною. У цьому розділі розглядаються шість архітектур глибокого навчання за останні 20 років. Примітно, що довготривала короткочасна пам’ять (LSTM) і згорточні нейронні мережі (CNN) є двома найстарішими підходами в цьому списку, але також двома найбільш використовуваними в різних програмах.

Архітектури глибокого навчання класифікуються на контрольоване та неконтрольоване навчання та представлено кілька популярних архітектур глибокого навчання: згорткові нейронні мережі, рекурентні нейронні мережі (RNN), довготривала короткочасна пам’ять/контрольована рекурентна одиниця (GRU), самоорганізуюча карта (SOM). , автокодери (AE) і обмежена машина Больцмана (RBM). Він також дає огляд мереж глибоких переконань (DBN) і мереж глибокого стекування (DSN).

Штучна нейронна мережа (ШНН) є базовою архітектурою глибинного навчання. На основі ШНМ було винайдено кілька варіантів алгоритмів. Щоб дізнатися про основи глибокого навчання та штучних нейронних мереж, прочитайте вступ до статті про глибоке навчання.

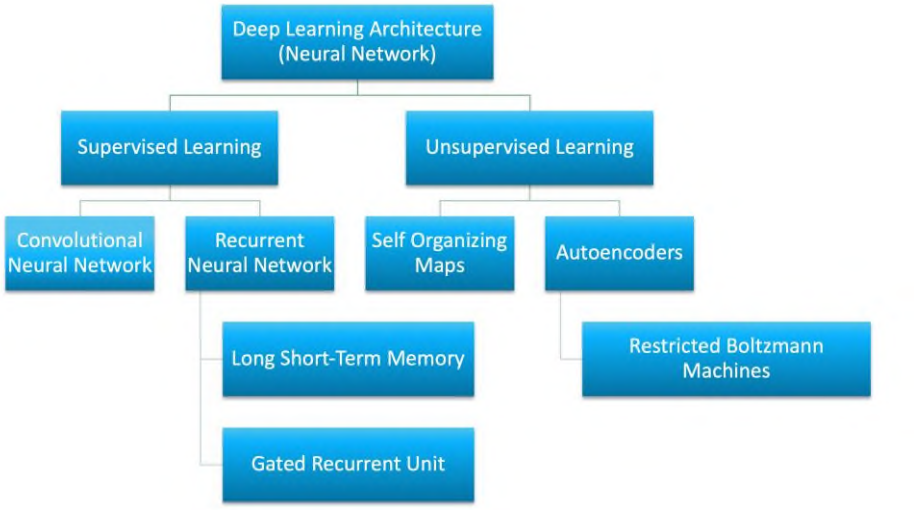


Рис. 2.1 Архітектура мережі з глибоким навчанням

Глибоке навчання під наглядом.

Контрольоване навчання відноситься до проблемного простору, де ціль, яку потрібно передбачити, чітко позначена в даних, які використовуються для навчання.

У цьому розділі ми представляємо на високому рівні дві найпопулярніші керовані архітектури глибокого навчання – згорточні нейронні мережі та рекурентні нейронні мережі, а також деякі їх варіанти.

Згорткові нейронні мережі.

CNN — це багатошарова нейронна мережа, яка була біологічно натхненна зоровою корою тварин. Архітектура особливо корисна в програмах обробки зображень. Перший CNN створив Ян ЛеКун; у той час архітектура була зосереджена на розпізнаванні рукописних символів, таких як інтерпретація поштового індексу. Будучи глибокою мережею, ранні рівні

розпізнають функції (наприклад, грані), а пізніші рівні рекомбінують ці функції в атрибути вищого рівня вхідних даних.

Архітектура LeNet CNN складається з кількох рівнів, які реалізують виділення ознак, а потім класифікацію (див. наступне зображення). Зображення ділиться на сприйнятливі поля, які надходять у згортковий шар, який потім витягує функції з вхідного зображення. Наступним кроком є об'єднання, яке зменшує розмірність вилучених функцій (через зменшення вибірки), зберігаючи при цьому найважливішу інформацію (як правило, за допомогою максимального об'єднання). Потім виконується ще один крок згортки та об'єднання, який надходить у повністю зв'язаний багатошаровий перцептрон. Кінцевий вихідний рівень цієї мережі – це набір вузлів, які ідентифікують особливості зображення (у цьому випадку вузол на ідентифіковане число). Ви навчаєте мережу за допомогою зворотного поширення.

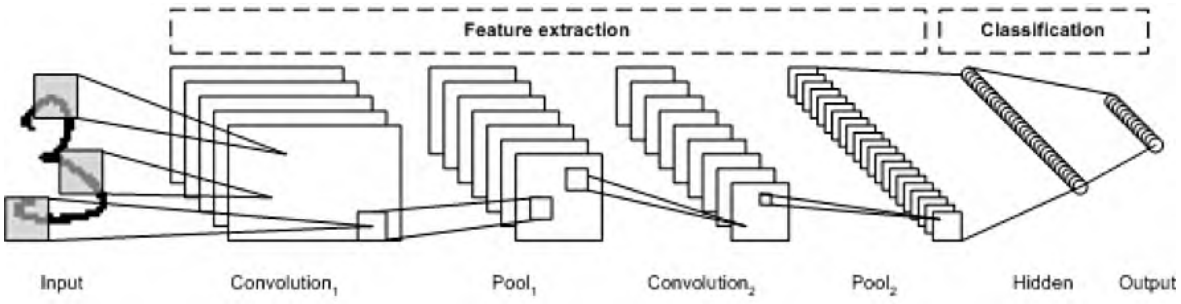


Рис. 2.2 Архітектура CNN

Використання глибоких рівнів обробки, згорток, об'єднання та повністю підключеного рівня класифікації відкрило двері для різноманітних нових застосувань нейронних мереж глибокого навчання. Окрім обробки зображень, CNN успішно застосовувався для розпізнавання відео та різних завдань у рамках обробки природної мови.

Приклади застосування: розпізнавання зображень, аналіз відео та обробка природної мови.

Рекурентні нейронні мережі.

RNN є однією з основоположних мережевих архітектур, з яких будуються інші архітектури глибокого навчання. Основна відмінність між типовою

багаторівневою мережею та повторюваною мережею полягає в тому, що рекурентна мережа може мати з'єднання, які повертаються на попередні рівні (або на той самий рівень), а не на повні з'єднання прямого зв'язку. Цей зворотний зв'язок дозволяє RNN зберігати пам'ять про минулі вхідні дані та вчасно моделювати проблеми.

RNN складаються з багатого набору архітектур (далі ми розглянемо одну популярну топологію під назвою LSTM). Ключовою відмінністю є зворотний зв'язок у мережі, який може проявлятися з прихованого рівня, вихідного рівня або деякої їх комбінації.

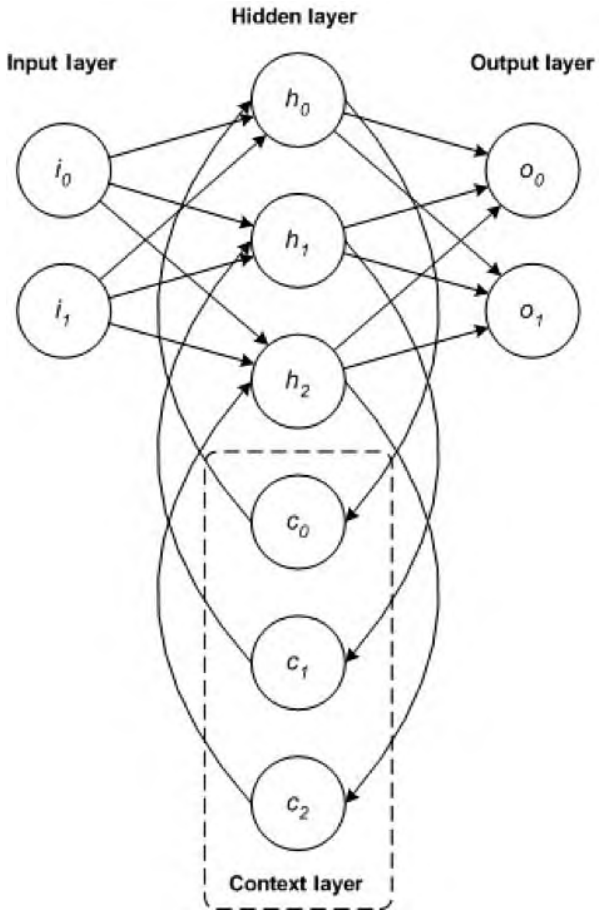


Рис. 2.3 Рекурентна нейронна мережа

RNN можна розгорнути в часі та навчити за допомогою стандартного зворотного поширення або за допомогою варіанту зворотного поширення, який називається зворотним поширенням у часі (BPTT).

Приклади програм: розпізнавання мовлення та розпізнавання рукописного тексту.

Мережі LSTM.

LSTM був створений у 1997 році Hochreiter і Schimdhuber, але в останні роки він став популярним як архітектура RNN для різних застосувань. Ви знайдете LSTM у продуктах, якими користуєтеся щодня, наприклад у смартфонах. IBM застосувала LSTM в IBM Watson® для розпізнавання розмовного мовлення.

LSTM відійшов від типової архітектури нейронної мережі на основі нейронів і натомість представив концепцію комірки пам’яті. Комірка пам’яті може зберігати своє значення протягом короткого чи тривалого часу залежно від своїх вхідних даних, що дозволяє комірці запам’ятовувати те, що є важливим, а не лише своє останнє обчислене значення.

Комірка пам'яті LSTM містить три ворота, які контролюють те, як інформація надходить у комірку або з неї. Вхідні ворота контролюють, коли нова інформація може надходити в пам'ять. Шлюз забуття контролює, коли наявна частина інформації забута, дозволяючи комірці запам’ятовувати нові дані. Нарешті, вихідний вентиль контролює, коли інформація, яка міститься в комірці, використовується у виводі з комірки. Комірка також містить ваги, які керують кожним воротами. Алгоритм навчання, зазвичай BPTT, оптимізує ці ваги на основі результуючої помилки виведення мережі.

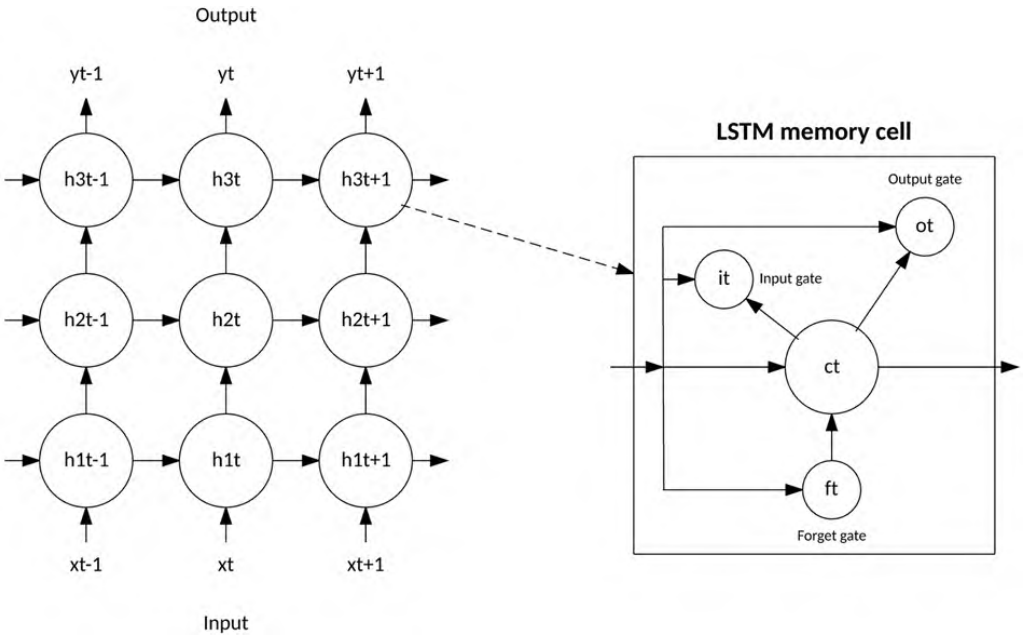


Рис. 2 4 LSTM мережа

Нещодавні застосування CNN і LSTM створили системи субтитрів до зображень і відео, в яких зображення або відео підписується природною мовою. CNN реалізує обробку зображень або відео, а LSTM навчено перетворювати вихідні дані CNN на природну мову.

Приклади застосування: системи субтитрів до зображень і відео.

Мережі GRU.

У 2014 році було введено спрощення LSTM, яке називається рекурентним блоком зі стробами. Ця модель має два вентилі, позбувшись вихідного вентиля, наявного в моделі LSTM. Ці ворота — це ворота оновлення та ворота скидання. Шлюз оновлення вказує, скільки вмісту попередньої комірки потрібно зберегти. Шлюз скидання визначає, як об'єднати новий вхід із попереднім вмістом клітинки. GRU може змоделювати стандартну RNN, просто встановивши ворота скидання на 1 і оновлення на 0.

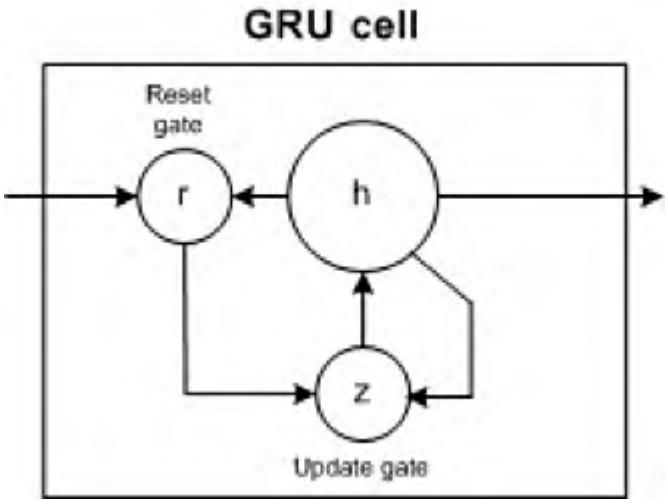


Рис. 2.5 GRU мережа

GRU є простішим, ніж LSTM, його можна навчити швидше та може бути більш ефективним у своєму виконанні. Однак LSTM може бути більш виразним і з більшою кількістю даних може призвести до кращих результатів.

Приклади застосування: стиснення тексту природною мовою, розпізнавання рукописного тексту, розпізнавання мовлення, розпізнавання жестів, підписи до зображень.

Глибоке навчання без нагляду.

Неконтрольоване навчання відноситься до проблемного простору, де в даних, які використовуються для навчання, немає цільової мітки.

У цьому розділі розглядаються три неконтрольовані архітектури глибокого навчання: самоорганізовані карти, автокодери та обмежені машини Больцмана. Ми також обговорюємо, як будуються мережі глибоких переконань і мережі глибокого стекування на основі базової неконтрольованої архітектури.

Самоорганізовані карти.

Самоорганізована карта (SOM) була винайдена доктором Теуво Кохоненом у 1982 році та була широко відома як карта Кохонена. SOM — це неконтрольована нейронна мережа, яка створює кластери набору вхідних даних шляхом зменшення розмірності вхідних даних. SOM багато в чому відрізняються від традиційної штучної нейронної мережі.

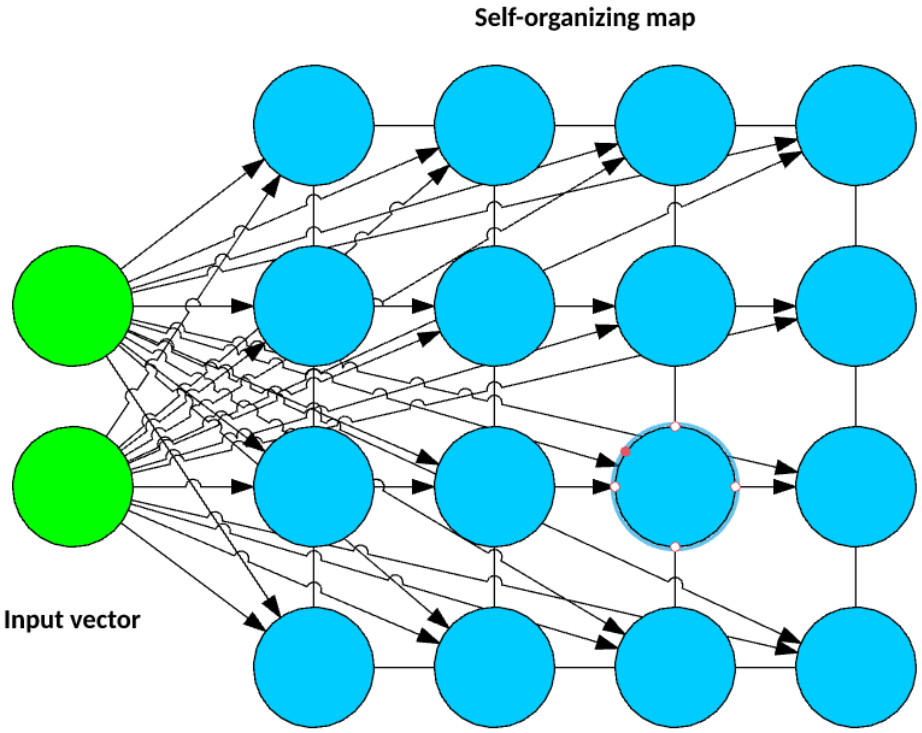


Рис. 2.6 Самоорганізована карта

Перша суттєва відмінність полягає в тому, що ваги служать характеристикою вузла. Після нормалізації входів спочатку вибирається випадковий вхід. Випадкові ваги, близькі до нуля, ініціалізуються для кожної функції вхідного запису. Ці ваги тепер представляють вхідний вузол. Кілька комбінацій цих випадкових ваг представляють варіації вхідного вузла. Обчислюється евклідова відстань між кожним із цих вихідних вузлів і вхідним

вузлом. Вузол із найменшою відстанню оголошується як найточніше представлення вхідних даних і позначається як найкраща відповідна одиниця або BMU. З цими BMU як центральними точками інші одиниці обчислюються подібним чином і призначаються кластеру, відстань від якого знаходиться. Радіуси точок навколо ваг BMU оновлюються на основі близькості. Радіус зменшено.

Далі, у SOM не застосовується функція активації, і оскільки немає цільових міток для порівняння, немає концепції обчислення помилки та зворотного поширення.

Приклади застосування: зменшення розмірності, кластеризація високовимірних вхідних даних у 2-вимірний вихід, сяючий результат оцінки та візуалізація кластерів.

Автокодери.

Хоча історія винаходу автокодерів туманна, перше відоме використання автокодерів було виявлено Лекуном у 1987 році. Цей варіант ШНМ складається з 3 рівнів: вхідного, прихованого та вихідного.

Спочатку вхідний рівень кодується у прихований шар за допомогою відповідної функції кодування. Кількість вузлів у прихованому шарі значно менша за кількість вузлів у вхідному шарі. Цей прихований шар містить стиснуте представлення вихідного введення. Вихідний рівень спрямований на реконструкцію вхідного рівня за допомогою функції декодера.

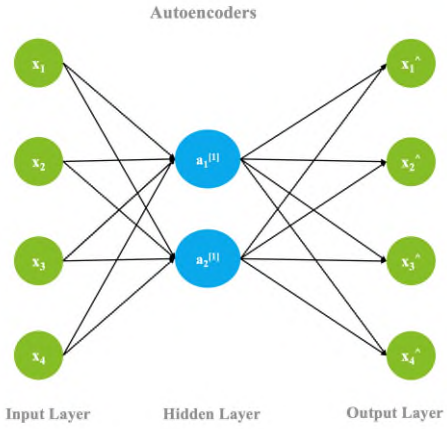


Рис. 2.7 Автокодери

Під час фази навчання різниця між вхідним і вихідним шарами обчислюється за допомогою функції помилки, а ваги коригуються, щоб мінімізувати помилку. На відміну від традиційних методів неконтрольованого навчання, де немає даних для порівняння виходів, автокодери безперервно навчаються за допомогою зворотного розповсюдження. З цієї причини автокодери класифікуються як самоконтрольовані алгоритми.

Приклади застосування: зменшення розмірності, інтерполяція даних і стиснення/декомпресія даних.

2.2 Алгоритми навчання нейромереж

Проблема навчання.

Задача навчання формулюється як мінімізація індексу втрат f. Це функція, яка вимірює продуктивність нейронної мережі на наборі даних. Індекс втрат включає, загалом, помилку та умови регулярізації. Термін помилки оцінює, як нейронна мережа відповідає набору даних. Термін регулярізації запобігає переобладнанню, контролюючи складність моделі.

Функція втрат залежить від адаптивних параметрів (зміщення та синаптичні ваги) у нейронній мережі. Ми можемо згрупувати їх в один n-вимірний ваговий вектор w. Наступний малюнок представляє функцію втрат f(w).

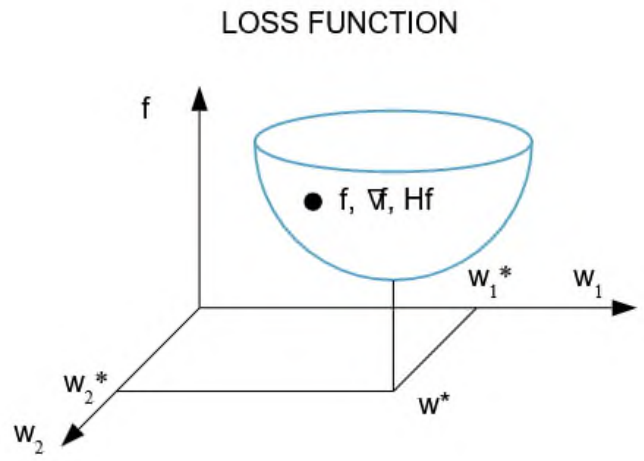


Рис. 2.8 Функція втрат

Як бачимо, мінімум функції втрат припадає на точку w∗. У будь-якій точці A, ми можемо обчислити першу та другу похідні функції втрат.

Вектор градієнта групує перші похідні,

$$\nabla_i f(\mathbf{w}) = \frac{\partial f}{\partial w_i},$$

для i=1,…,n.

Аналогічно, матриця Гессе групує другі похідні,

$$\mathbf{H}_{i,j} f(\mathbf{w}) = \frac{\partial^2 f}{\partial w_i \partial w_j},$$

для i, j=0,1,…

Широко досліджується проблема мінімізації неперервних і диференційованих функцій багатьох змінних. Ми можемо безпосередньо застосувати багато звичайних підходів до цієї проблеми для навчання нейронних мереж.

Одновимірна оптимізація.

Хоча функція втрат залежить від багатьох параметрів, велике значення тут мають одновимірні методи оптимізації. Дійсно, вони дуже часто використовуються в процесі навчання нейронної мережі.

Багато навчальних алгоритмів спочатку обчислюють напрям навчання d а потім швидкість навчання η, яка мінімізує втрати в цьому напрямку, f(η). Наступний малюнок ілюструє цю одновимірну функцію.

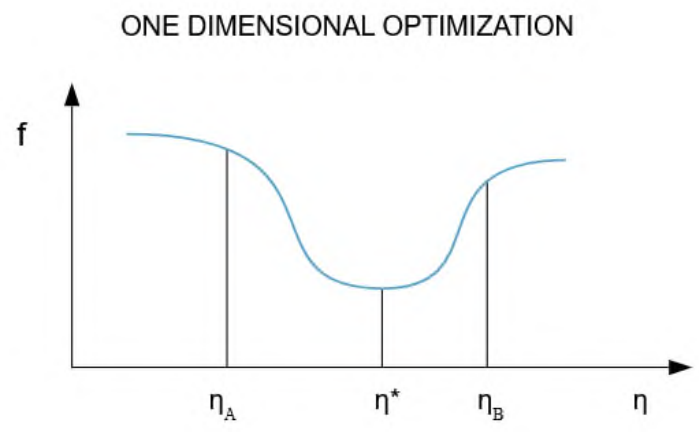


Рис. 2.9 Одновимірна оптимізація

Точки η1 і η2 визначають інтервал, який містить мінімум f, η∗.

У зв'язку з цим одновимірні методи оптимізації здійснюють пошук мінімуму одновимірних функцій. Одними з найбільш використовуваних є золотий переріз і метод Брента. Обидва зменшують мінімальну дужку, доки відстань між зовнішніми точками не стане меншою за визначений допуск.

Багатовимірна оптимізація.

Задача навчання для нейронних мереж формулюється як пошук вектора параметрів w∗ при якому функція втрат f приймає мінімальне значення. Необхідна умова стверджує, що якщо нейронна мережа має мінімум функції втрат, то градієнт є нульовим вектором.

Функція втрат є, загалом, нелінійною функцією параметрів. Отже, неможливо знайти закриті алгоритми навчання для мінімумів. Замість цього ми розглядаємо пошук у просторі параметрів, що складається з послідовності кроків. На кожному кроці втрати будуть зменшуватися шляхом коригування параметрів нейронної мережі.

Таким чином, щоб навчити нейронну мережу, ми починаємо з деякого вектора параметрів (часто вибирається навмання). Потім ми генеруємо послідовність параметрів, щоб функція втрат зменшувалася на кожній ітерації алгоритму. Зміна втрат між двома кроками називається декрементом втрат. Алгоритм навчання зупиняється, коли задовольняється задана умова або критерій зупинки.

Градієнтний спуск (GD)

Градієнтний спуск - найпростіший алгоритм навчання. Для цього потрібна інформація з вектора градієнта, тому це метод першого порядку.

Позначимо f(w(i))=f(i) і ∇f(w(i))=g(i). Метод починається в точці w(0) і, поки не буде задоволено критерій зупинки, рухається від w(i) до w(i+1) у напрямку навчання d(i)=−g(i). Таким чином, метод градієнтного спуску повторюється таким чином:

$$w^{(i+1)} = w^{(i)} - \mathbf{g}^{(i)}\eta^{(i)},$$

для i=0,1,….

Параметр η – це курс навчання. Це значення можна або встановити на фіксоване значення, або знайти за допомогою одновимірної оптимізації вздовж напрямку навчання на кожному кроці. Оптимальне значення для швидкості навчання, отримане мінімізацією лінії на кожному наступному кроці, як правило, є кращим. Однак багато програмних засобів все ще використовують лише фіксоване значення для швидкості навчання.

На наступному малюнку – діаграма діяльності тренувального процесу з градієнтним спуском. Як ми бачимо, алгоритм покращує параметри в два етапи: спочатку він обчислює напрямок тренування градієнтного спуску. По-друге, він знаходить відповідну швидкість навчання.

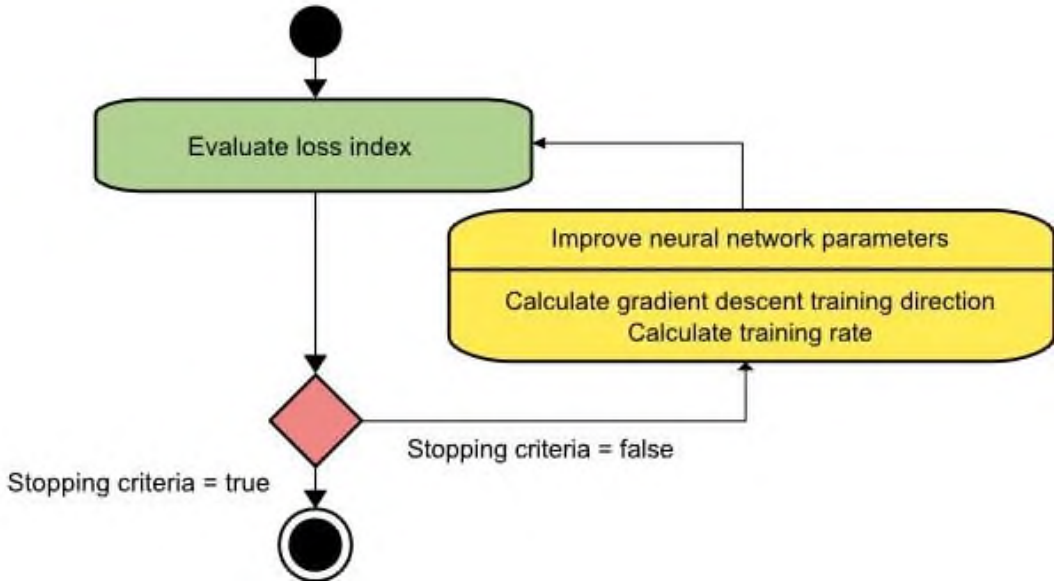


Рис. 2.10 Діаграма діяльності тренувального процесу градієнтного спуску

Алгоритм навчання градієнтного спуску має серйозний недолік, оскільки вимагає багатьох ітерацій для функцій, які мають довгі, вузькі долинні структури. Дійсно, низхідний градієнт — це напрямок, у якому функція втрат спадає найшвидше, але це не обов’язково призводить до найшвидшої конвергенції. Наведене нижче зображення ілюструє цю проблему.

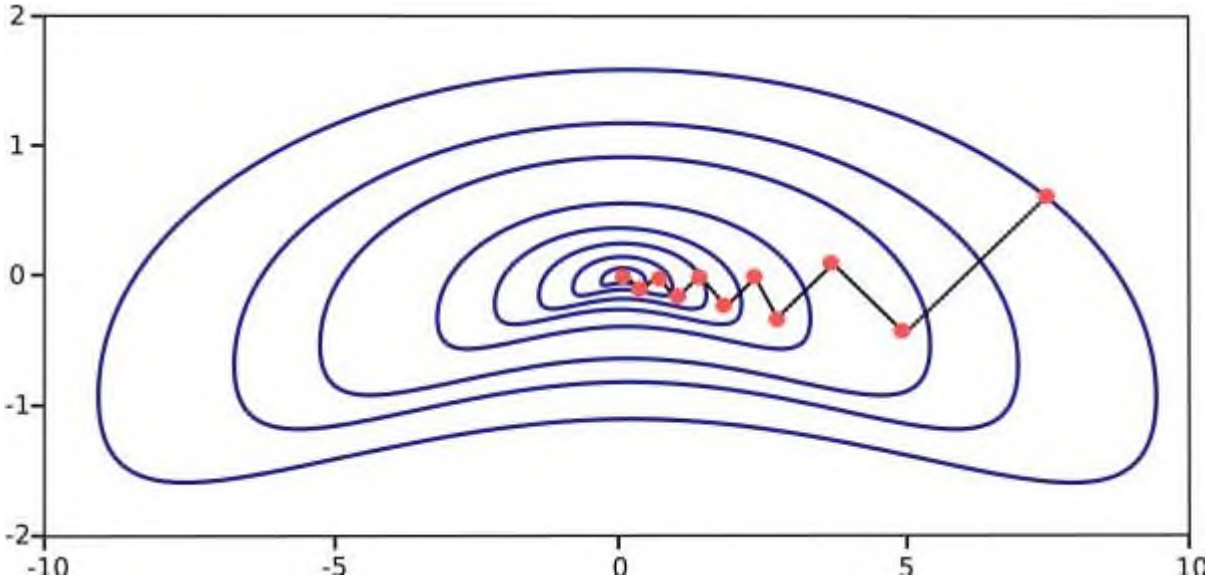


Рис. 2.11 Ітерації функцій

Градієнтний спуск — це рекомендований алгоритм, коли у нас є масивні нейронні мережі з багатьма тисячами параметрів. Причина в тому, що цей метод зберігає лише вектор градієнта (розмір n) і не зберігає матрицю Гессе (розмір n2).

Метод Ньютона (НМ)

Метод Ньютона є алгоритмом другого порядку, оскільки він використовує матрицю Гессе. Метою цього методу є пошук кращих напрямків навчання за допомогою других похідних функції втрат.

Позначимо f(w(i))=f(i)

, ∇f(w(i))=g(i) і Hf(w(i))=H(i). Розглянемо квадратичну апроксимацію f при w(0) за допомогою розкладання в ряд Тейлора

$$f = f^{(0)} + g^{(0)} \cdot (\mathbf{w} - \mathbf{w}^{(0)}) + 0.5 \cdot (\mathbf{w} - \mathbf{w}^{(0)})^2 \cdot \mathbf{H}^{(0)}$$

H(0) — це матриця Гессе f, обчислена в точці w(0). Встановлюючи g рівним 0 для мінімуму f(w), ми отримуємо наступне рівняння

$$g = g^{(0)} + \mathbf{H}^{(0)} \cdot (\mathbf{w} - \mathbf{w}^{(0)}) = 0$$

Тому, починаючи з вектора параметрів w(0), метод Ньютона виконує наступні ітерації

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{H}^{(i)-1} \cdot \mathbf{g}^{(i)}$$

для i=0,1,….

Вектор H(i)−1·g(i) відомий як крок Ньютона. Зверніть увагу, що ця зміна параметра може рухатися до максимуму, а не до мінімуму. Це відбувається, якщо матриця Гессе не є позитивно визначеною. Таким чином, метод Ньютона не гарантує зниження індексу втрат на кожній ітерації. Щоб запобігти цьому, рівняння методу Ньютона зазвичай модифікують так:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - (\mathbf{H}^{(i)-1} \cdot \mathbf{g}^{(i)})\eta$$

для i=0,1,….

Швидкість навчання, η, може бути встановлена на фіксоване значення або знайдена шляхом мінімізації лінії. Вектор d(i)=H(i)−1·g(i) тепер називається напрямком навчання Ньютона.

На наступному малюнку зображено діаграму стану процесу навчання за методом Ньютона. Параметри покращуються шляхом отримання спочатку напрямку навчання, а потім відповідного курсу навчання.

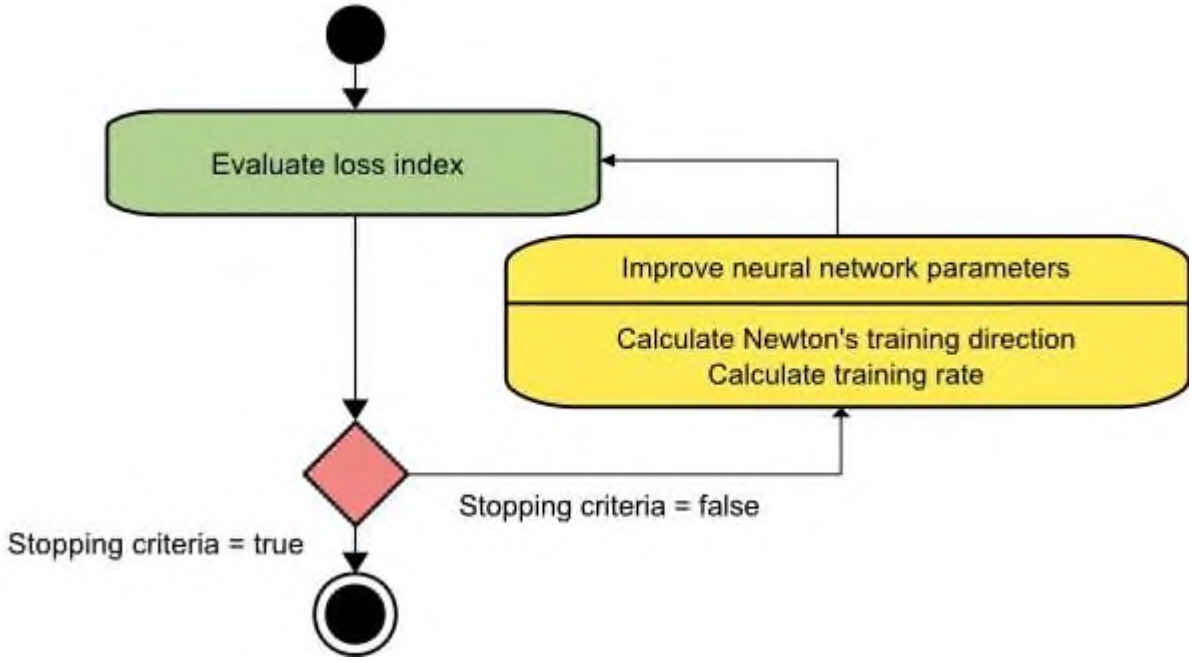


Рис. 2.12 Діаграма стану процесу навчання методом Ньютона

На малюнку нижче показано ефективність цього методу. Як ми бачимо, метод Ньютона вимагає менше кроків, ніж градієнтний спуск, щоб знайти мінімальне значення функції втрат.

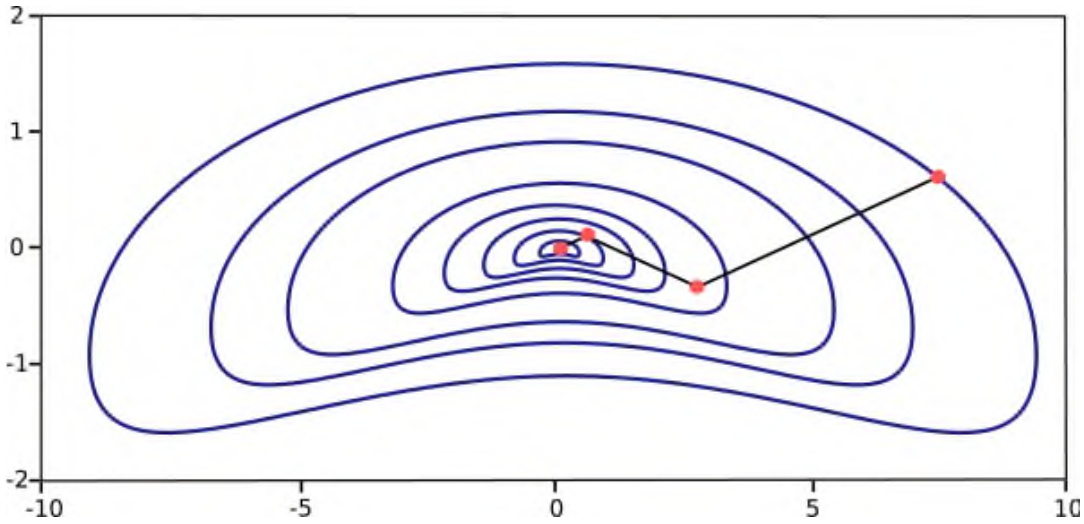


Рис. 2.13 Ефективність методу Ньютона

2.3 Порівняльна характеристика програм глибокого навчання

Глибоке навчання нещодавно набуло величезної популярності, а різноманітні архітектури глибокого навчання роблять цю сферу ще більш поширеною. Для підтримки реалізації кожної з цих архітектур для різних варіантів використання доступні кілька фреймворків. Незважаючи на те, що кожен із цих фреймворків має свої плюси та мінуси, вибір правильного фреймворку глибокого навчання на основі вашого індивідуального робочого навантаження є важливим першим кроком, який має зробити кожен розробник, спеціаліст із глибокого навчання чи науковець із даних.

Рис. 2.14 Фреймворки глибокого навчання

У цьому розділі наведено огляд шести найпопулярніших фреймворків глибокого навчання: TensorFlow, Keras, PyTorch, Caffe, Theano та Deeplearning4j. За останні кілька років три з цих фреймворків глибокого навчання — Tensorflow, Keras і PyTorch — набрали обертів завдяки своїй простоті у використанні, широкому використанню в академічних дослідженнях, а також комерційному коду та розширюваності. У цій статті ми також порівняємо TensorFlow і PyTorch.

TensorFlow

У контексті машинного навчання тензор відноситься до багатовимірного масиву, який використовується в математичних моделях, які описують нейронні мережі. Іншими словами, тензор зазвичай є узагальненням матриці або вектора вищої розмірності.

За допомогою простої нотації, яка використовує ранг для відображення кількості вимірів, тензори дозволяють представляти складні _n_-вимірні вектори та гіперфігури як _n_-вимірні масиви. Тензори мають дві властивості: тип даних і форму.

TensorFlow — це платформа глибокого навчання з відкритим кодом, яка була випущена наприкінці 2015 року під ліцензією Apache 2.0. Відтоді він став одним із найпоширеніших фреймворків глибокого навчання у світі (зважаючи на кількість проектів GitHub на його основі).

TensorFlow бере свій початок від Google DistBelief, запатентованої системи глибокого навчання, розробленої проектом Google Brain. Google розробив TensorFlow з нуля для розподіленої обробки та оптимальної роботи на спеціально розробленій інтегральній схемі (ASIC) від Google, яка називається Tensor Processing Unit (TPU) у своїх виробничих центрах обробки даних. Ця конструкція робить TensorFlow ефективним для програм глибокого навчання.

Фреймворк може працювати на CPU, GPU або TPU на серверах, настільних комп’ютерах і мобільних пристроях. Розробники можуть розгорнути TensorFlow на кількох операційних системах і платформах локально або в хмарі. Багато розробників вважають, що TensorFlow має кращу підтримку

для розподіленої обробки та більшу гнучкість і продуктивність для комерційних додатків, ніж аналогічні фреймворки глибокого навчання, такі як Torch і Theano, які також здатні апаратно прискорювати та широко використовуються в академічних колах.

Нейронні мережі глибокого навчання зазвичай складаються з багатьох шарів. Вони передають дані або виконують операції між шарами за допомогою багатовимірних масивів. Тензор протікає між шарами нейронної мережі, отже, назва TensorFlow.

Основною мовою програмування для TensorFlow є Python. C++, мова Java® і програмний інтерфейс (API) Go також доступні без гарантій стабільності, як і багато прив'язок сторонніх розробників для C#, Haskell, Julia, Rust, Ruby, Scala, R і навіть PHP. Google має оптимізовану для мобільних пристроїв бібліотеку TensorFlow-Lite для запуску програм TensorFlow на Android.

У цьому розділі наведено огляд системи TensorFlow, включно з перевагами та застосуваннями фреймворку.

Переваги TensorFlow.

TensorFlow пропонує розробникам:

- Завзяте виконання. TensorFlow 2 підтримує швидке виконання, за допомогою якого операції оцінюються негайно, а конкретні значення повертаються без побудови графіків. Це допомагає при побудові моделей і налагодженні моделей.
- Обчислювальна графова модель. TensorFlow використовує графіки потоку даних, які називаються орієнтованими графами, для вираження обчислювальних моделей. Це робить його інтуїтивно зрозумілим для розробників, які можуть легко візуалізувати те, що відбувається на рівнях нейронної мережі за допомогою вбудованих інструментів, і вдосконалювати свої моделі нейронної мережі, налаштовуючи параметри та конфігурації в інтерактивному режимі.

- Простий у використанні API. Розробники Python можуть використовувати необроблений низькорівневий API або базовий API TensorFlow для розробки власних моделей або використовувати бібліотеки API вищого рівня для вбудованих моделей. TensorFlow має багато вбудованих і доданих бібліотек, і можна накласти структуру глибокого навчання вищого рівня, таку як Keras, щоб діяти як API високого рівня. Багато попередніх API було або видалено, або оновлено до TensorFlow 2.0.
- Гнучка архітектура. Головною перевагою використання TensorFlow є те, що він має модульну, розширювану та гнучку конструкцію. Розробники можуть легко переміщувати моделі між процесорами CPU, GPU або TPU з невеликими змінами коду. Незважаючи на те, що TensorFlow спочатку був розроблений для широкомасштабного розподіленого навчання та висновків, розробники також можуть використовувати TensorFlow для експериментів з іншими моделями машинного навчання та системної оптимізації існуючих моделей.
- Розподілена обробка. Google Brain з нуля розробив TensorFlow для розподіленої обробки на спеціальному ASIC TPU. Крім того, TensorFlow може працювати на кількох ядрах GPU NVIDIA. Розробники можуть скористатися перевагами архітектури ЦП x64 на базі Intel Xeon і Xeon Phi або архітектури ЦП ARM64. TensorFlow може працювати на мультиархітектурних і багатоядерних системах, а також як розподілений процес, який виділяє інтенсивну обробку як робочі завдання. Розробники можуть створювати кластери серверів TensorFlow і розподіляти обчислювальний графік між цими кластерами для навчання. TensorFlow може виконувати розподілене навчання як синхронно, так і асинхронно, як у межах графа, так і між графами, а також може ділитися загальними даними в пам’яті чи між мережевими обчислювальними вузлами.

- Продуктивність. Продуктивність часто є спірною темою, але більшість розробників розуміють, що будь-яка структура глибокого навчання залежить від основного апаратного забезпечення для оптимальної роботи для досягнення високої продуктивності з низькими витратами енергії. Як правило, власна платформа розробки будь-якої фреймворку забезпечує найкращу оптимізацію. TensorFlow найкраще працює на TPU Google, але йому вдається досягти високої продуктивності на різних платформах — не лише на серверах і настільних комп'ютерах, а й на вбудованих системах і мобільних пристроях. Фреймворк також підтримує дивовижну кількість мов програмування. Хоча інший фреймворк, який працює нативно, як-от IBM Watson на платформі IBM, іноді може перевершувати TensorFlow, він усе ще є фаворитом розробників, оскільки проекти штучного інтелекту (ШІ) можуть охоплювати платформи та мови програмування, націлені на численні кінцеві програми, усі з яких мають створювати послідовні результати.

Додатки TensorFlow.

У цьому розділі розглядаються додатки, у яких добре працює TensorFlow. Очевидно, оскільки Google використовував свою власну версію TensorFlow для текстового та голосового пошуку, мовного перекладу та пошуку зображень, основні переваги TensorFlow полягають у класифікації та висновках. Наприклад, Google реалізував RankBrain, механізм, який ранжує результати пошуку Google, у TensorFlow.

TensorFlow можна використовувати для покращення розпізнавання мовлення та синтезу мовлення шляхом диференціації кількох голосів або фільтрації мовлення в середовищах із високим рівнем шуму, імітації голосових шаблонів для більш природного звучання тексту в мовлення. Крім того, він обробляє структуру речень різними мовами для кращого перекладу. Його можна використовувати для розпізнавання зображень і відео, а також для

класифікації об’єктів, орієнтирів, людей, настроїв або діяльності. Це призвело до значного покращення пошуку зображень і відео.

Завдяки гнучкому, розширюваному та модульному дизайну TensorFlow не обмежує розробників окремими моделями чи програмами. Розробники використовували TensorFlow для впровадження не лише алгоритмів машинного й глибокого навчання, але й статистичних і загальних обчислювальних моделей. Для отримання додаткової інформації про додатки та надані моделі перегляньте тематичні дослідження TensorFlow.

Keras

Keras — це бібліотека глибокого навчання на основі Python, яка відрізняється від інших фреймворків глибокого навчання. Keras функціонує як специфікація API високого рівня для нейронних мереж. Він може служити і як інтерфейс користувача, і для розширення можливостей інших систем глибокого навчання, на яких він працює.

Keras починався як спрощений інтерфейс для академічно популярного фреймворку Theano. Відтоді Keras API став частиною Google TensorFlow. Keras офіційно підтримує Microsoft Cognitive Toolkit (CNTK), Deeplearning4J і Apache MXNet. Завдяки такій широкій підтримці Keras отримав позицію де-факто інструменту для міграції між фреймворками. Розробники можуть переносити не лише алгоритми та моделі нейронних мереж глибокого навчання, але й попередньо підготовлені мережі та ваги.

Keras — це пакет Python з відкритим кодом, випущений за ліцензією Массачусетського технологічного інституту (MIT), де Франсуа Шолле, Google, Microsoft та інші учасники володіють деякими авторськими правами на програмне забезпечення.

Інтерфейс Keras дозволяє швидко створювати прототипи моделей нейронних мереж у дослідженнях. API простий у вивченні та використанні та має додаткову перевагу легкого перенесення моделей між фреймворками.

Оскільки Keras автономний, ви можете використовувати його без взаємодії з серверною платформою, на якій він працює. Keras має власні структури даних графів для визначення обчислювальних графіків: він не покладається на структури даних графіків базової внутрішньої структури. Цей підхід звільняє вас від необхідності вчитися програмувати серверну структуру, тому Google вирішила додати Keras API до свого ядра TensorFlow.

Однак у Keras 2.4.0 компанія Keras припинила роботу з мультисервером Keras і зосередилася виключно на TensorFlow.

Переваги Keras.

Отже, навіщо використовувати Keras? Є ряд переваг, зокрема:

- Кращий досвід користувача (UX) для програм глибокого навчання. Keras API є зручним для користувача. API добре розроблений, об’єктно-орієнтований і гнучкий, що забезпечує кращий досвід роботи з користувачем. Дослідники можуть визначати нові моделі глибокого навчання без необхідності працювати з потенційно складними серверними частинами, що призводить до простішого та компактнішого коду.
- Бездоганна інтеграція Python. Keras — це нативний пакет Python, який забезпечує легкий доступ до всієї екосистеми науки про дані Python. Наприклад, Python scikit-learn API також може використовувати моделі Keras. Розробники, знайомі з серверними частинами, такими як TensorFlow, також можуть використовувати Python для розширення Keras.
- Великий портативний обсяг роботи та потужна база знань. Дослідники вже деякий час використовують Keras із системою Theano. Результатом є великий обсяг роботи та потужна база знань спільноти, яку розробники глибокого навчання можуть легко перенести з внутрішніх систем Theano на серверні частини TensorFlow. Навіть обважнювачі можна переносити між задніми частинами, що означає, що попередньо навчені моделі можуть

легко міняти місцями задні кінці лише кількома налаштуваннями. Дослідження Кераса та Теано залишаються актуальними для TensorFlow та інших серверних систем. Крім того, Keras робить багато навчальних ресурсів, документацію та зразки коду безкоштовно доступними.

Демістифікація зв'язку між TensorFlow 2 і Keras

Keras і TensorFlow часто помилково вважаються конкурентними фреймворками. Keras — це високорівневий API для розробки моделей нейронних мереж, який не обробляє низькорівневі обчислення. Для цих низькорівневих обчислень Keras покладався на інші серверні механізми, такі як Theano, Tensorflow і CNTK. Однак, згідно з останнім випуском Keras, Keras головним чином зосередиться на інтеграції з API ядра TensorFlow, продовжуючи підтримувати виправлення для Theano/CNTK.

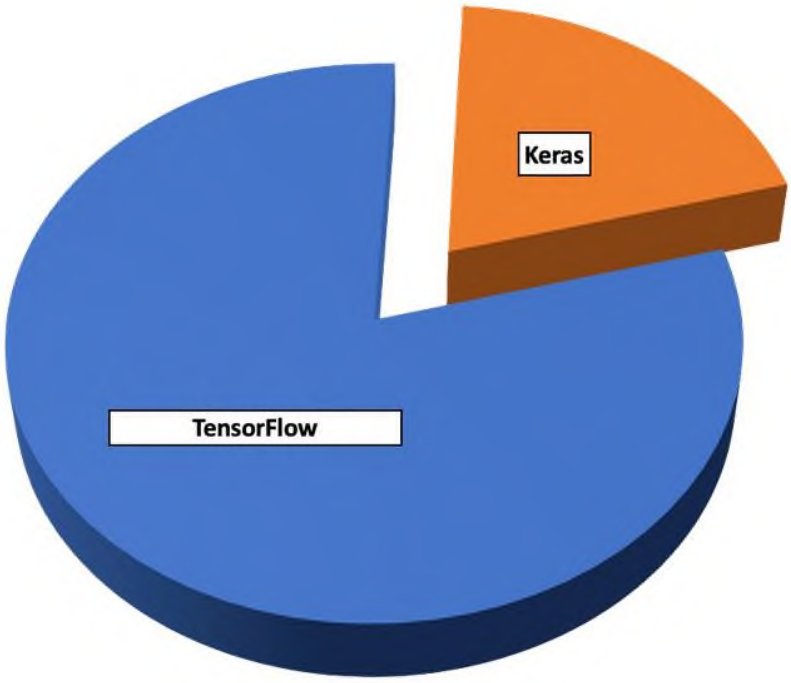


Рис. 2.15 Демістифікація зв'язку між TensorFlow 2 і Keras

PyTorch

PyTorch — це пакет Python з відкритим кодом, випущений за модифікованою ліцензією Berkeley Software Distribution. Авторські права на PyTorch належать Facebook, Дослідницькому інституту Idiap, Нью-Йоркському

університету (NYU) та NEC Labs America. Хоча Python є мовою вибору для науки про дані, PyTorch є відносно новачком на арені глибокого навчання.

У цьому розділі наведено огляд системи PyTorch і надано додаткові відомості про такі теми:

- Фон PyTorch
- Переваги використання PyTorch
- Типові програми PyTorch

Алгоритми нейронної мережі зазвичай обчислюють піки або спади функції втрат, при цьому більшість із них використовує для цього функцію градієнтного спуску. У Torch, попереднику PyTorch, пакет Torch Autograd, наданий Twitter, обчислює градієнтні функції. Torch Autograd базується на Python Autograd.

Основним випадком використання PyTorch є дослідження. Facebook використовує PyTorch для інноваційних досліджень і переходить на Caffe2 для виробництва. Формат Open Neural Network Exchange дозволяє користувачам конвертувати моделі між PyTorch і Caffe2 і скорочує час затримки між дослідженнями та виробництвом.

Пакети Python, такі як Autograd і Chainer, використовують техніку, відому як автоматичне диференціювання на основі стрічки, для обчислення градієнтів. Таким чином, ці пакети сильно вплинули та надихнули на дизайн PyTorch.

Автоматична диференціація на основі стрічки працює так само, як магнітофон, оскільки вона записує виконані операції, а потім відтворює їх для обчислення градієнтів — метод, також відомий як автоматична диференціація у зворотному режимі. PyTorch Autograd має одну з найшвидших реалізацій цієї функції.

Використовуючи цю функцію, користувачі PyTorch можуть довільно налаштовувати свої нейронні мережі без накладних витрат або штрафів за затримку. У результаті, на відміну від більшості відомих фреймворків, користувачі PyTorch можуть динамічно будувати графіки, а швидкість і

гнучкість фреймворку полегшують дослідження та розробку нових алгоритмів глибокого навчання.

Частина цієї продуктивності забезпечується модульною конструкцією, яка використовується в ядрі PyTorch. PyTorch реалізує більшість тензорних і нейронних мережевих серверів для процесорів і графічних процесорів як окремі та компактні модулі на основі C, з інтегрованими бібліотеками прискорення математики для підвищення швидкості.

PyTorch бездоганно інтегрується з Python і використовує імперативний стиль кодування. Крім того, дизайн зберігає розширюваність, яка зробила популярним його попередника на основі Lua. Користувачі можуть програмувати за допомогою C/C++ із розширенням API, що базується на інтерфейсі зовнішніх функцій C (cFFI) для Python.

Caffe

Ще один популярний фреймворк глибокого навчання — Caffe. Спочатку Caffe було розроблено як частину Ph.D. дисертацію, але тепер випущено за ліцензією Berkeley Software Distribution. Caffe підтримує широкий спектр архітектур глибокого навчання, включаючи CNN і LSTM, але особливо не підтримує RBM або DBM (хоча Caffe2 включатиме таку підтримку).

Caffe використовувався для класифікації зображень та інших додатків зору, і він підтримує прискорення на основі графічного процесора за допомогою бібліотеки NVIDIA CUDA Deep Neural Network. Caffe підтримує Open Multi-Processing (OpenMP) для розпаралелювання алгоритмів глибокого навчання в кластері систем. Caffe та Caffe2 написані на C++ для підвищення продуктивності та пропонують інтерфейс Python та MATLAB для глибокого навчання та виконання.

Theano

Theano — це низькорівнева бібліотека Python, яка використовується для виконання завдань глибокого навчання, пов’язаних із визначенням, оптимізацією та оцінкою математичних виразів. Хоча він має досить вражаючу обчислювальну продуктивність, є багато скарг щодо інтерфейсу та повідомлень

про помилки. Тому Theano в основному використовується в поєднанні з оболонками, такими як Keras, Lasagne і Blocks, трьома фреймворками високого рівня, спрямованими на швидке створення прототипів.

Deeplearning4j

Deeplearning4j — популярна структура глибокого навчання, яка орієнтована на технологію Java, але включає інтерфейси програмування додатків для інших мов, таких як Scala, Python і Clojure. Фреймворк випущений за ліцензією Apache і включає підтримку RBM, DBN, CNN і RNN. Deeplearning4j також включає розподілені паралельні версії, які працюють з Apache Hadoop і Spark (фреймворки обробки великих даних).

Deeplearning4j застосовувався для вирішення різних проблем, включаючи виявлення шахрайства у фінансовому секторі, системи рекомендацій, розпізнавання зображень і кібербезпеку (виявлення мережевих вторгнень). Фреймворк інтегрується з CUDA для оптимізації GPU і може поширюватися з OpenMP або Hadoop.

Висновки

Проблематика розпізнавання голосу досить специфічна, оскільки потребує спеціально підготовлених наборів даних, що розмічені автоматично чи мануально, та архітектури нейронної мережі яка здатна розрізняти маленькі, міжкласово подібні, різноманітні, тісно скупчені зображення.

Ми можемо використовувати велику кількість даних, які стали доступними завдяки швидкому зростанню мережі Інтернет. Два таких джерела даних - це анонімізовані запити на google.com та на інших Інтернет-сайтах. Вони допомагають покращити автоматичне розпізнавання мовлення за допомогою великих мовних моделей: голосовий пошук використовує перший, тоді як транскрипція мовлення YouTube значно виграє від другої. Ці методи допомагають заощадити багато часу та не формувати багатомільйонні словники для кожного проекту.

РОЗДІЛ 3. ПРОЕКТУВАННЯ НЕЙРОМЕРЕЖІ

3.1 Вибір структури нейромережі

Нейронне структуроване навчання (NSL) - це нова парадигма навчання для навчання нейронних мереж за рахунок використання структурованих сигналів на додаток до вхідних даних. Структура може бути явною, представленою графом, або неявною, спричиненою ворожим обуренням.

Структуровані сигнали зазвичай використовуються для подання відносин або подібності між зразками, які можуть бути позначені або не позначені. Тому при використанні цих сигналів під час навчання нейронної мережі використовують як розмічені, так і нерозмічені дані, що може підвищити точність моделі, особливо коли кількість розмічених даних відносно невелика . Крім того, було показано, що моделі, навчені за допомогою вибірок, створених шляхом додавання ворожих обурень, стійкі до зловмисних атак, які призначені для того, щоб ввести в оману прогноз або класифікацію моделі.

NSL узагальнює навчання нейронних графів, а також змагальне навчання. Платформа NSL у TensorFlow надає розробникам такі прості у використанні API та інструменти для навчання моделей за допомогою структурованих сигналів:

- API-інтерфейси Keras для забезпечення навчання за допомогою графіків (явна структура) та ворожих збурень (неявна структура).
- Операції та функції TF для забезпечення навчання зі структурою при використанні API TensorFlow нижчого рівня.

- Інструменти для побудови графіків та побудови вхідних даних для навчання

Включення структурованих сигналів здійснюється лише під час навчання. Таким чином, продуктивність робочого процесу обслуговування/висновку залишається незмінною. Більше інформації про нейронному структурованому навчанні можна знайти в описі нашого фреймворку. Щоб розпочати роботу, ознайомтеся з нашим посібником зі встановлення , а для практичного ознайомлення з NSL ознайомтеся з нашими навчальними посібниками.

3.2 Вибір середовища розробки

Microsoft Visual Studio — один з продуктів компанії Майкрософт, який представляє з себе інтегровану середу розробки програмного забезпечення і ряд інших інструментальних засобів. Даний продукт дозволяє розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в не керованому, так і керованому кодах для всіх платформ, підтримуваних Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework і Microsoft Silverlight.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторингу коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і як відладчик машинного рівня. Решта вбудовуваних інструментів включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду, додавання нових наборів інструментів, наприклад, для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування або інструментів для інших аспектів циклу розробки програмного забезпечення. Загальний вигляд вікна,

яке відображається при завантаженні Visual Studio, початок роботи та створення нового проекту (рисунок 3.1).

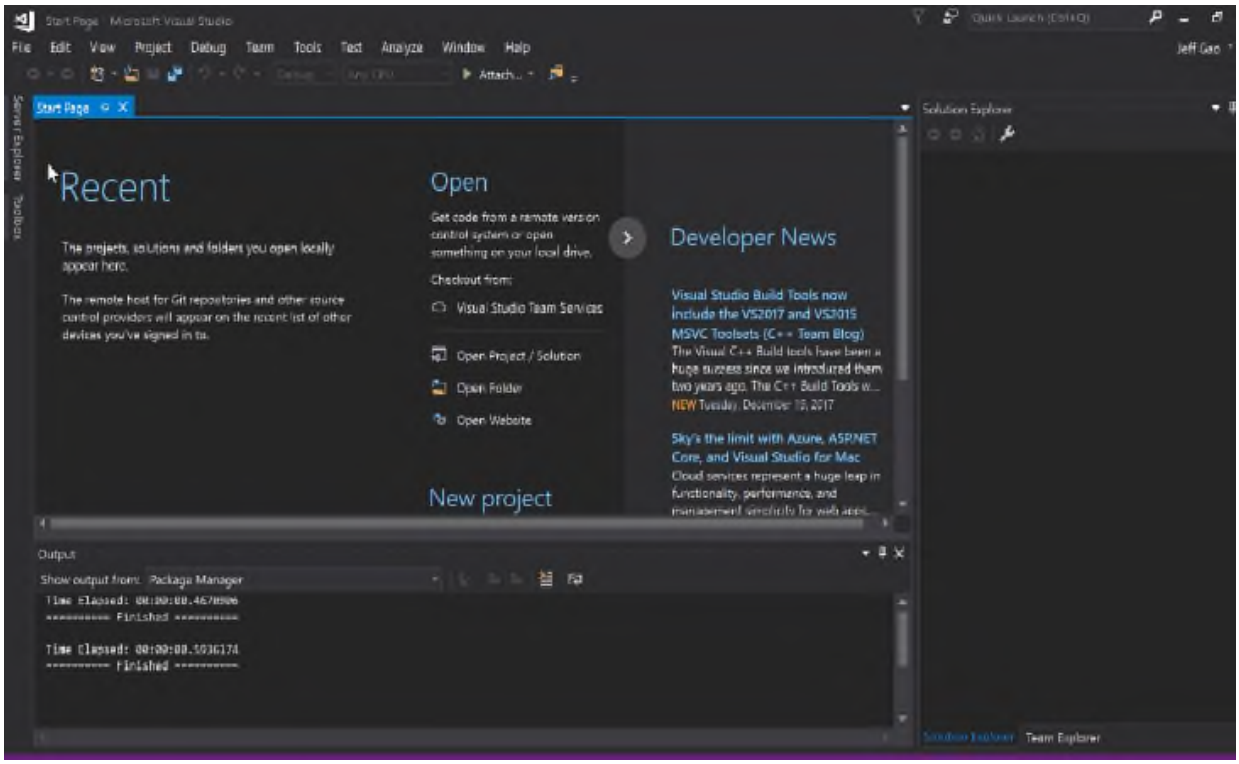


Рис. 3.1 Інтерфейс середовища розробки

3.3 Вибір алгоритму навчання

Обчислювальна графова модель. TensorFlow використовує графіки потоку даних, які називаються орієнтованими графами, для вираження обчислювальних моделей. Це робить його інтуїтивно зрозумілим для розробників, які можуть легко візуалізувати те, що відбувається на рівнях нейронної мережі за допомогою вбудованих інструментів, і вдосконалювати свої моделі нейронної мережі, налаштовуючи параметри та конфігурації в інтерактивному режимі.

3.4 Сценарій роботи з нейромережею

Тепер давайте перейдемо до обговорення точних етапів роботи нейронної мережі.

- Спочатку набір даних повинен бути поданий на вхідний рівень, який потім буде перенесено на прихований шар.

- Зв’язки, які існують між двома шарами, випадково призначають ваги вхідним даних.
- Зміщення додається до кожного входу. Зсув – це константа, яка використовується в моделі, щоб найкраще відповідати заданим даним.
- Зважена сума всіх вхідних даних буде надіслана до функції, яка використовується для визначення активного стану нейрона шляхом обчислення зваженої суми та додавання зсуву. Ця функція називається функцією активації.
- Вузли, які повинні запускатися для вилучення функцій, вирішуються на основі вихідного значення функції активації.
- Остаточний вихід мережі потім порівнюється з необхідними позначеними даними нашого набору даних, щоб обчислити кінцеву помилку вартості. Помилка вартості насправді говорить нам, наскільки «погана» наша мережа. Тому ми хочемо, щоб помилка була якомога меншою.
- Вага регулюється шляхом зворотного поширення, що зменшує похибку. Цей процес зворотного поширення можна розглядати як центральний механізм, якому навчаються нейронні мережі. По суті, він точно налаштовує ваги глибокої нейронної мережі, щоб зменшити собівартість.

Простіше кажучи, під час навчання нейронної мережі ми зазвичай обчислюємо втрату (значення помилки) моделі та перевіряємо, зменшено вона чи ні. Якщо помилка перевищує очікуване значення, ми повинні оновити параметри моделі, такі як ваги та значення зміщення. Ми можемо використовувати модель, коли втрати нижчі за очікувану межу похибки.

3.5 Кодування нейромережі

Щоб побудувати просту нейронну мережу, нам потрібен набір даних, і тут ми будемо використовувати набір даних «fashion-mnist», який уже присутній у keras. Набір даних містить два набори: навчальний набір і тестовий набір, навчальний набір містить 60 000 прикладів даних, а тест — 10 000 прикладів даних. Ми також можемо сказати, що ми маємо дані про 70 000 зображень розміром 28x28 (28 пікселів у ширину та 28 пікселів у висоту), з яких 60 000 знаходяться на стадії навчання, а 10 000 – на тестуванні. Ці зображення мають відтінки сірого, на кожному з яких показано q0 можливий тип одягу.

Крок 1. Імпорт бібліотеки

```
import tensorflow as tf
from tensorflow import keras
```

Рис 3.2 Імпорт бібліотеки

Крок 2. Завантаження набору даних

```
(x_train_data, y_train_data), (x_val_data, y_val_data) = keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

Here in the above we are loading the dataset, also we are splitting them into training set and validation set. x_train_data and y_train_data is here to build our model in which x_train_data will consist of image pixel data for 60,000 clothes whereas the y_train_data will consist of classes i.e clothing type for x_train_data. Similarly the x_val_data and y_val_data is used for testing or validation of our model.

Рис. 3.3 Завантаження набору даних

Крок 3. Попередня обробка та створення набору даних

```
def preprocessing_function(x_new, y_new):
  x_new = tf.cast(x_new, tf.float32) / 255.0
  y_new = tf.cast(y_new, tf.int64)

  return x_new, y_new

def func_creating_dataset(xs_data, ys_data, num_classes=10):
  ys_data = tf.one_hot(ys_data, depth=num_classes)
  return tf.data.Dataset.from_tensor_slices((xs_data, ys_data)) \
    .map(preprocessing_function) \
    .shuffle(len(ys_data)) \
    .batch(128)
```

Рис. 3.4 Створення набору даних

Крок 4 - Створення нейронної мережі

```
dataset_training = func_creating_dataset(x_train_data, y_train_data)
dataset_val = func_creating_dataset(x_val_data, y_val_data)
```

Рис. 3.5 Створення нейронної мережі

Крок 5 - Побудова моделі

```
My_model = keras.Sequential([
    keras.layers.Reshape(target_shape=(28 * 28,), input_shape=(28, 28)),
    keras.layers.Dense(units=256, activation='relu'),
    keras.layers.Dense(units=192, activation='relu'),
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dense(units=10, activation='softmax')
])
```

Рис. 3.6 Побудова моделі

Крок 6. Навчання моделі

```
My_model.compile(optimizer='adam',
                 loss=tf.losses.CategoricalCrossentropy(from_logits=True),
                 metrics=['accuracy'])

history = My_model.fit(
    dataset_training.repeat(),
    epochs=10,
    steps_per_epoch=500,
    validation_data=dataset_val.repeat(),
    validation_steps=2
)
```

```
Epoch 1/10
500/500 [==============================] - 8s 12ms/step - loss: 0.6743 - accuracy: 0.7643 - val_loss: 0.4322 - val_accuracy: 0.8633
Epoch 2/10
500/500 [==============================] - 6s 11ms/step - loss: 0.3679 - accuracy: 0.8660 - val_loss: 0.3940 - val_accuracy: 0.8555
Epoch 3/10
500/500 [==============================] - 6s 12ms/step - loss: 0.3317 - accuracy: 0.8773 - val_loss: 0.3207 - val_accuracy: 0.9023
Epoch 4/10
500/500 [==============================] - 6s 12ms/step - loss: 0.3024 - accuracy: 0.8877 - val_loss: 0.3989 - val_accuracy: 0.8672
Epoch 5/10
500/500 [==============================] - 6s 12ms/step - loss: 0.2863 - accuracy: 0.8916 - val_loss: 0.3594 - val_accuracy: 0.8750
Epoch 6/10
500/500 [==============================] - 6s 12ms/step - loss: 0.2766 - accuracy: 0.8983 - val_loss: 0.4284 - val_accuracy: 0.8438
Epoch 7/10
500/500 [==============================] - 6s 11ms/step - loss: 0.2571 - accuracy: 0.9025 - val_loss: 0.2932 - val_accuracy: 0.8711
Epoch 8/10
500/500 [==============================] - 6s 12ms/step - loss: 0.2456 - accuracy: 0.9074 - val_loss: 0.1954 - val_accuracy: 0.9297
Epoch 9/10
```

Рис. 3.7 Навчання моделі

Крок 7 -Створення прогнозів

```
Make_predictions = My_model.predict(dataset_val)
Make_predictions
```

```
array([[8.21499402e-09, 1.98965409e-07, 1.17530963e-09, ...,
        2.39849047e-04, 3.75632886e-10, 9.99724329e-01],
       [4.75593419e-11, 1.32912683e-13, 1.02271674e-12, ...,
        3.79424998e-11, 1.00000000e+00, 1.16047255e-10],
       [6.36480451e-08, 1.22093002e-06, 1.70876291e-08, ...,
        2.29782687e-04, 4.84631757e-09, 9.99749601e-01],
       ...,
       [3.27635678e-13, 7.38353267e-11, 6.17022167e-14, ...,
        2.07368976e-06, 5.78794384e-15, 9.99997735e-01],
       [3.67982114e-15, 5.02441912e-18, 2.04865252e-15, ...,
        4.30711580e-18, 1.00000000e+00, 3.65123112e-20],
       [3.21948218e-10, 9.99999881e-01, 1.29861441e-10, ...,
        6.70993209e-12, 9.32399991e-11, 6.58160956e-11]], dtype=float32)
```

Рис. 3.8 Створення прогнозів

Висновки

У цьому розділі ми обговорили деякі важливі факти про нейронні мережі, наприклад, що таке нейронні мережі, як вони працюють, типи нейронних мереж і кілька варіантів їх використання. Власне, нейронні мережі на сьогоднішній день можна вважати найпомітнішим напрямком досліджень у галузі інформатики.

Існує купа моделей нейронних мереж, таких як CNN і RNN. Tesla є однією з компаній, які активно використовують нейронні мережі. Їхня машина Tesla, яка є автопілотом, нещодавно досягла величезного успіху завдяки кращим результатам, отриманим за допомогою нейронних мереж.

Нейронна мережа — це вражаюча технологія, яка відповідає за величезні прориви в усьому: від розпізнавання обличчя до подорожей.

РОЗДІЛ 4. ТЕСТУВАННЯ НЕЙРОМЕРЕЖІ

4.1. Підготовка вихідних даних для тестування

Завантаження набору даних.

Завантаження та підготовка набору даних MNIST. Перетворення даних вибірки з цілих чисел у числа з плаваючою комою (*Рис 4.1*):

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Рис 4.1 Завантаження набору даних

Побудова моделі машинного навчання.

Створення моделі tf.keras.Sequential шляхом накладання шарів.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

Рис. 4.2 Побудова моделі машинного навчання

Для кожного прикладу модель повертає вектор оцінок логітів або логарифмічних шансів по одному для кожного класу.

```
predictions = model(x_train[:1]).numpy()
predictions
```

```
array([[ 0.2760778 , -0.39324787, -0.17098302,  1.2016621 , -0.03416392,
         0.5461229 , -0.7203061 , -0.41886678, -0.59480035, -0.7580608 ]],
      dtype=float32)
```

Рис. 4.3 Вектор оцінки логітів

Функція tf.nn.softmax перетворює ці логіти на ймовірність для кожного класу:

```
tf.nn.softmax(predictions).numpy()
```

```
array([[0.11960829, 0.06124588, 0.0764901 , 0.30181262, 0.08770514,
        0.15668967, 0.04416083, 0.05969675, 0.05006609, 0.04252464]],
      dtype=float32)
```

Рис. 4.4 Перетворення логітів

Визначення функції втрат для навчання з використанням losses.SparseCategoricalCrossentropy, яка приймає вектор логітів та індекс True та повертає скалярну втрату для кожного прикладу.

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

Рис. 4.5 Визначення функції втрат

Ця втрата дорівнює негативній логарифмічній ймовірності істинного класу: втрата дорівнює нулю, якщо модель впевнена у правильному класі.

Ця ненавчена модель дає ймовірності, близькі до випадкових (1/10 для кожного класу), тому початкова втрата має бути близька до -tf.math.log(1/10) ~= 2.3.

```
loss_fn(y_train[:1], predictions).numpy()
```

```
1.8534881
```

Рис. 4.6 Початкова втрата

Налаштування перед початком навчання та компіляція моделі за допомогою Model.compile. Встановлення класу optimizer на adam, встановлення loss на функцію loss_fn , яку ви визначено раніше, та вказання метрики, яка оцінюватиметься для моделі, встановивши параметр metrics на accuracy.

```
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

Рис. 4.7 Налаштування перед початком навчання

4.2 Функціональне тестування

Хоча TF2.0 і Keras забезпечують набір добре реалізованих алгоритмів. У багатьох випадках нам все одно потрібно бруднити руки та впроваджувати власні моделі та шари. Реалізації передбачають набір математичних перетворень. Це робить необхідним перевірку правильності. Модульне тестування підходить для цього, коли ми хочемо перевірити правильність нашої власної реалізації.

Модуль-тест TF 2.0 дуже схожий на модульний тест Python. Все починається з класу TestCase. Наступний код є мінімальною конфігурацією модульного тесту. У цьому тестовому випадку ми перевіримо правильність шарів Keras Dense, перевіривши його результат.

У наступному коді ми спочатку визначаємо клас DenseLayerTest, який успадковує tf.test.TestCase. Потім ми реалізуємо метод setUp. Цей метод викликається на початку кожного тестового методу.

Ми реалізуємо один тестовий метод під назвою testDenseLayerOutput. Цей метод тестування перевіряє, чи щільний шар дає нам правильний результат.

AssertAllEqual надається в tf.test.TestCase, цей метод перевіряє, чи дорівнює очікуваний вихід обчисленому.

```
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf
import numpy as np


class DenseLayerTest(tf.test.TestCase):

    def setUp(self):
        super(DenseLayerTest, self).setUp()
        self.my_dense = tf.keras.layers.Dense(2)
        self.my_dense.build((2, 2))

    def testDenseLayerOutput(self):
        self.my_dense.set_weights([
            np.array([[1, 0],
                      [2, 3]]),
            np.array([0.5, 0])
        ])
        input_x = np.array([[1, 2],
                            [2, 3]])
        output = self.my_dense(input_x)
        expected_output = np.array([[5.5, 6.],
                                    [8.5, 9]])

        self.assertAllEqual(expected_output, output)


tf.test.main()
```

Рис. 4.7. Клас DenseLayerTest

Результат роботи запущеного коду:

```
Running tests under Python 3.7.4: /Users/rl/anaconda3
/envs/machine_learning/bin/python
[ RUN      ] DenseLayerTest.testDenseLayerOutput
[       OK ] DenseLayerTest.testDenseLayerOutput
[ RUN      ] DenseLayerTest.test_session
[  SKIPPED ] DenseLayerTest.test_session
----------------------------------------------------------------------
Ran 2 tests in 0.023s

OK (skipped=1)
```

Рис. 4.8 Результат роботи коду

DenseLayerTest.DenseLayerTest.testDenseLayerOutput, наш тест пройдено. (Не потрібно турбуватися про сесію DenseLayerTest.test_session, оскільки вона була ініційована методом setUp.)

Тут показано простий варіант використання assertAllEqual. Існує повний список assert методів на tf.test.TestCase.

Реалізація нетривіального рівня Tensorflow 2.0 і проведення модульного тестування.

У випадку використано Multi-head Attention, який є найважливішим рівнем, що використовується в моделі Transformer. Transformer добре застосовувався в багатьох завданнях моделювання мови. НЛП.

Нижче наведено реалізацію цього рівня на основі цього документа. scaled_dot_product_attention — це допоміжний метод для обчислення презентації на основі уваги.

```
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf

class MultiHeadAttention(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads, debug_mode):
        super(MultiHeadAttention, self).__init__()
        self.num_heads = num_heads
        self.d_model = d_model

        assert d_model % self.num_heads == 0

        self.depth = d_model // self.num_heads

        self.wq = tf.keras.layers.Dense(d_model)
        self.wk = tf.keras.layers.Dense(d_model)
        self.wv = tf.keras.layers.Dense(d_model)

        self.dense = tf.keras.layers.Dense(d_model)

        self.debug_mode = debug_mode

    def split_heads(self, x, batch_size):
        """Split the last dimension into (num_heads, depth).
        Transpose the result such that the shape is (batch_size, num_heads, seq_len, depth)
        """
        x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
        return tf.transpose(x, perm=[0, 2, 1, 3])
```

Рис. 4.9. Реалізація нетривіального рівня Tensorflow 2.0

Висновки

Після демонстрації роботи програми, оглянуто напрямки подальшої роботи над програмою. Розробка має учбовий та науковий характер, оскільки для комерційного використання найбільш логічно будувати нейронні мережі, виходячи із поведінки реальних користувачів та працювати над системою, що дозволить запропонувати, наприклад, опції для голосового тестування. Однак, задача розпізнавання елементів голосу методами машинного навчання має велике прикладне значення, якщо воно відбувається з високою точністю.

ВИСНОВКИ

У роботі вирішується актуальне завдання – розробка системи розпізнавання голосу з високою точністю розпізнавання голосу з використанням нейронних мереж.

Основними науково-теоретичними та практичними результатами є:

1. Проведено огляд відомих принципів побудови нейронної мережі та методів реалізації, що дозволило виявити сильні та слабкі сторони існуючих рішень та виділити рішення, які можна вважати прототипами.
2. Спільне використання повторюваних НМ і сервісів Google Cloud дозволяє поєднати переваги обох методів: забезпечити високу точність відтворення того чи іншого сигналу, якщо це необхідно.
3. Був створений план, у якому голосові команди активували пристрої.
4. Розробка системи розпізнавання голосу у вигляді чат-бота.
5. Розробка системи голосового керування на основі відповідей чат-бота на запити користувачів. пов'язані з проектом,
6. У результаті маркетингового аналізу можливостей впровадження запропонованих науково-технічних рішень і пропозицій та оцінки можливостей їх реалізації на ринку можна визначити, що розроблений план комерціалізації ринку можливий і може стати прибутковим комерційним проектом. Збільшення попиту на однакові товари збільшує обсяг закупівлі тих самих пристроїв, але створює жорстку конкурентну ситуацію для виходу на ринок.
7. Враховуючи поточну ринкову ситуацію, проект має великий потенціал для реалізації, що вимагає нових потужних та економічних рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ ISO 9241-10-2001 Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 10. Принципи діалогу (ISO 9241 10:1996, IDT). – Держстандарт України, 2002. Чинний.
2. Roto, Virpi. Web Browsing on Mobile Phones – Characteristics of User Experience. автореф. дис. докт. техн. наук : 2006 — 22 с. Режим доступу: https://www.researchgate.net/figure/User-experience-definition-by-Hassenzahl-Tractinsky-2006_fig5_27516496
3. Hartson R. The UX Book: Process and Guidelines for Ensuring a Quality User Experience / R. Hartson, P. Pardha. – Waltham, USA: Elsevier Inc, 2012. – 937 с. – (3).
4. Vermeeren, Arnold & Law, Lai-Chong & Roto, Virpi & Obrist, Marianna & Hoonhout, Jettie & Väänänen, Kaisa. User experience evaluation methods: Current state and development needs. NordiCHI 2010: Extending Boundaries - Proceedings of the 6th Nordic Conference on Human-Computer Interaction. 521-530. 10.1145/1868914.1868973.
5. Graham Dove, Kim Halskov, Jodi Forlizzi, and John Zimmerman. UX Design Innovation: Challenges for Working with Machine Learning as a Design Material. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17). Association for Computing Machinery, New York, NY, USA, 278–288. DOI: https://doi.org/10.1145/3025453.3025739
6. Michael Chromik, Florian Lachner, and Andreas Butz. ML for UX? - An Inventory and Predictions on the Use of Machine Learning Techniques for UX Research. Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society. Association for Computing Machinery, New York, NY, USA, Article 57, 2020, p. 1–11. DOI: https://doi.org/10.1145/3419249.3420163

7. Bonastre, L., & Granollers, T. A Set Of Heuristics for User Experience Evaluation in E-commerce Websites. *ACHI 2014*. Режим доступу: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.672.2794&rep=rep1&type=pdf

8. V.Alberti UI design principles. 2018 – 49 с. Режим доступу: http://ska-sdp.org/sites/default/files/attachments/ska-tel-sko-0000787-02_uidesignprinciples.pdf

9. Bader, Frederik & Schön, Eva-Maria & Thomaschewski, Jörg. Heuristics Considering UX and Quality Criteria for Heuristics. International Journal of Interactive Multimedia and Artificial Intelligence №4, 2017, p. 48-53. 10.9781/ijimai.2017.05.001.

10. Hu, G., Zhu, L., & Yang, J. AppFlow: Using Machine Learning to Synthesize Robust, Reusable UI Tests. Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, с. 269–282. https://doi.org/10.1145/3236024.3236055

11. Claire Kayacik, Sherol Chen, Signe Nørly, Jess Scon Holbrook, Adam Roberts, & Doug Eck. Identifying the intersections: User experience + research scientist collaboration in a generative machine learning interface. ACM CHI Conference, 2019. Режим доступу: https://dl.acm.org/doi/pdf/10.1145/3290607.3299059

12. K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett and D. Poshyvanyk, "Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps," in IEEE Transactions on Software Engineering, vol. 46, № 2, pp. 196-221, 2020, Режим доступу: https://arxiv.org/pdf/1802.02312.pdf

13. Robinson, J., Lanius, C., & Weber, R. (2018). The Past, Present, and Future of UX Empirical Research. *Commun. Des. Q. Rev, 5(3), 10–23.* https://dl.acm.org/doi/pdf/10.1145/3188173.3188175

14. T. A. Nguyen and C. Csallner, "Reverse Engineering Mobile Application User Interfaces with REMAUI (T)," 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015, pp. 248-259, DOI: 10.1109/ASE.2015.32.

15. Carmen Krahe, Antonio Bräunche, Alexander Jacob, Nicole Stricker, Gisela Lanza, Deep Learning for Automated Product Design, Procedia CIRP, Volume 91, 2020, с. 3-8, Режим доступу: https://doi.org/10.1016/j.procir.2020.01.135.

16. Georges, V., Courtemanche, F., Senecal, S., Baccino, T., Fredette, M., & Leger, P.-M. (2016). UX Heatmaps: Mapping User Experience on Visual Interfaces. Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, с. 150-160 Режим доступу: https://doi.org/10.1145/2858036.2858271

17. ДСТУ ISO 13407:2007 Людино-центричні процеси проектування діалогових систем (EN ISO 13407:1999, IDT). − Київ: Держстандарт України, 2009. Чинний.

18. Su C. How Do You Use Deep Learning to Identify UI Components? [Електронний ресурс] / Chuan Su. – 2021. – Режим доступу до ресурсу: https://www.alibabacloud.com/blog/how-do-you-use-deep-learning-to-identify-ui-components_597859

19. Li, Yuanchun, Ziyue Yang, Yao Guo, and Xiangqun Chen. "Humanoid: A deep learning-based approach to automated black-box android app testing." In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1070-1073. IEEE, 2019.

20. Moran, Kevin, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. "Machine learning-based prototyping of graphical user interfaces for mobile apps." IEEE Transactions on Software Engineering 46, no. 2 (2018): 196-221.

21. Kumar, Anurendra, Keval Morabia, Jingjin Wang, Kevin Chen-Chuan Chang, and Alexander Schwing. "CoVA: Context-aware Visual Attention for Webpage Information Extraction." arXiv preprint arXiv:2110.12320 (2021).

22. Pernice, K., K. Whitenton, and J. Nielsen. "How People Read Online: The Eyetracking Evidence." Fremont, USA: Nielsen Norman Group (2014).

23. Georges, Vanessa, François Courtemanche, Sylvain Senecal, Thierry Baccino, Marc Fredette, and Pierre-Majorique Leger. "UX heatmaps: mapping user experience on visual interfaces." In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, pp. 4850-4860. 2016.

24. Liao, Shengcai, Anil K. Jain, and Stan Z. Li. "A fast and accurate unconstrained face detector." IEEE transactions on pattern analysis and machine intelligence 38, no. 2 (2015): 211-223.

25. Liu, Thomas F., Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. "Learning design semantics for mobile apps." In Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology, pp. 569-579. 2018.

26. Massaro, Alessandro, Daniele Giannone, Vitangelo Birardi, and Angelo Maurizio Galiano. "An Innovative Approach for the Evaluation of the Web Page Impact Combining User Experience and Neural Network Score." Future Internet 13, no. 6 (2021): 145.

27. Pernice, K., K. Whitenton, and J. Nielsen. “How People Read Online: The Eyetracking Evidence” 2nd edition, Fremont, USA: Nielsen Norman Group (2018).