Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Модель інтерпретації людської мови засобами нейронних мереж»

Студента 2м курсу, 2м групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Маслова Артема Івановича

Науковий керівник кандидат педагогічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Котенко Наталія Олексіївна

Гарант освітньої програми доктор економічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Токар Володимир Володимирович

КИЇВ – 2021

Київський національний торговельно-економічний університет

Факультет *інформаційних технологій*
Кафедра *інженерії програмного забезпечення та кібербезпеки*
Освітній ступінь *магістр*
Спеціальність *121 «Інженерія програмного забезпечення»*
Спеціалізація / освітня програма *121 «Інженерія програмного забезпечення»*

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
"10" листопада 2020 р.

Завдання
на випускну кваліфікаційну роботу студентові

Маслову Артему Івановичу

1. Тема випускної кваліфікаційної роботи: *«Модель інтерпретації людської мови засобами нейронних мереж»*
Затверджена наказом ректора від *«30» листопада 2020 р. № 3224*
2. Строк здачі студентом закінченої роботи *25 листопада 2021р.*
3. Цільова установка та вихідні дані до роботи

Мета роботи*: *аналіз і розробка методу виокремлення мовлення окремих дикторів

Об'єкт дослідження:* *рекурентні нейронні мережі

Предмет дослідження:* *розробка методу навчання рекурентних нейронних мереж і використання з ціллю сегментації звуку

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 ЗАГАЛЬНІ ВІДОМОСТІ ТА ОСНОВНІ ВИМОГИ ДО РЕАЛІЗАЦІЇ СИСТЕМИ

1.1. Загальні відомості

1.1.1. Найменування системи

1.1.2. Планові терміни початку та закінчення робіт

1.1.3. Порядок оформлення і пред'явлення результатів робіт

1.1.4. Головний бенефіціар та потенційні користувачі системи

1.2. Мета та призначення створення системи

1.2.1. Призначення системи

1.2.2. Мета створення системи

1.3. Вимоги до системи

1.3.1. Вимоги в цілому

1.3.2. Вимоги до структури

1.4. Вимоги до компонентів системи:

1.4.1. Layer (шар)

1.4.2. Dense

1.4.3. Activation layer

1.4.4. Activations

1.4.5 Losses

1.4.6 Network

1.5. Огляд рішень

1.6. Висновки до розділу 1

РОЗДІЛ 2 АНАЛІЗ ПРЕДМЕТА ДОСЛІДЖЕННЯ ТА ВИЗНАЧЕННЯ ОСНОВНИХ КОМПОНЕНТІВ СИСТЕМИ

2.1. Основні поняття і позначення

2.1.1. Теорема апроксимації

2.1.2. Прикладне застосування

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2020*	*21.09.2020*
2.	*Розробка та затвердження завдання на проєкт магістра (стац/заоч)*	*06.11.2020*	*22.12.2020*
3.	*Вступ та перелік літературних джерел*	*27.02.2021*	*27.02.2021*
4.	*Розробка технічного завдання*	*20.03.2021*	*20.03.2021*
5.	*Розділ 1. Загальні відомості та основні вимоги до реалізації системи*	*16.04.2021*	*16.04.2021*
6.	*Розділ 2. Аналіз предмета дослідження та визначення основних компонентів системи*	*24.05.2021*	*24.05.2021*
7.	*Розділ 3. Імплементація нейронної мережі*	*21.06.2021*	*21.06.2021*
8.	*Розділ 4. (Об'єднання розділів 3 та 4)*	*20.09.2021*	*20.09.2021*
9.	*Розробка програми та методики тестування*	*18.10.2021*	*18.10.2021*
10.	*Написання наукової статті*	*22.05.2021*	*22.05.2021*
11.	*Керівництво користувача*	*21.10.2021*	*21.10.2021*
12.	*Висновки та пропозиції*	*01.11.2021*	*01.11.2021*
13.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*03.11.2021*	*03.11.2021*
14.	*Підготовка автореферату та презентації доповіді*	*03.11.2021*	*03.11.2021*
15.	*Попередній захист випускного кваліфікаційного проєкту*	*22.11.2021 – 25.11.2021*	*22.11.2021*
16.	*Здача зброшурованої випускного кваліфікаційного проєкту*	*25.11.2021*	*25.11.2021*
17.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*26.11.2021*	*26.11.2021*
18.	*Підготовка до публічного захисту випускного кваліфікаційного проєкту*		

6. Дата видачі завдання «10» листопада 2020 р.

7. Науковий керівник випускного кваліфікаційного проєкту Котенко Н.О.
(прізвище, ініціали, підпис)

8. Гарант освітньої програми Токар В.В.
(прізвище, ініціали, підпис)

9. Завдання прийняв до виконання студент Маслов А.І.
(прізвище, ініціали, підпис)

11. Відгук керівника випускної кваліфікаційної роботи

Науковий керівник випускного кваліфікаційного проєкту

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

11. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента Маслова А.І.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Токар В.В.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.

(підпис, прізвище, ініціали)

«____»_____20___ р.

АНОТАЦІЯ

Дипломна робота на тему «Модель інтерпретації людської мови засобами нейронних мереж» містить 54 сторінок, 31 рисунків та 1 додаток. Список посилань включає 24 пункти.

Мета – аналіз і розробка методу виокремлення мовлення окремих дикторів.

Об'єкт – рекурентні нейронні мережі.

Предмет – розробка методу навчання рекурентних нейронних мереж і використання з ціллю сегментації звуку.

Завдання дослідження: розглянути основні принципи побудови сучасних нейронних мереж, структуру готових рішень для глибокого навчання; засвоїти основні поняття та визначення з лінійної алгебри та усвідомити їхнє геометричне представлення; на основі даних аналізу, розробити рішення для відділення цільової мови від фонових перешкод.

Методи дослідження: аналіз, синтез, моделювання, дедукція, аналогія.

У першому розділі йдеться про загальні відомості системи, мету, вимоги до мережі в цілому та до конкретних задач, що будуть виконуватися системою. Також наведений огляд готових рішень для розробки нейромереж.

У другому розділі йдеться про основні поняття лінійної алгебри та опис окремих елементів мережі. Пояснюється також безпосередній метод навчання мереж – градієнтний спуск.

У третьому розділі описуються реалізації самої мережі на мові програмування Python, описані формули, що були використані при написанні мережі.

ANNOTATION

The thesis on the topic "Human language interpretation by means of neural networks " contains 54 pages, 31 drawings, and 1 appendix. The list of links includes 24 items.

An object is the analysis and development of a method of creating and training recurrent neural networks.

The research subject is the recurrent neural networks.

The subject of study is the development of a method of teaching recurrent neural networks and use for the purpose of sound segmentation.

Research objectives: explore fundamental principles of constructing modern neural networks, inspect ready-made solutions for deep learning; master basic concepts and definitions of linear algebra, build intuition and awareness of their geometric representations; based on data analysis, develop solutions for separating target audio from background interference.

Research methods: analysis, synthesis, modeling, deduction, analogy.

The first section describes general information about the system, the goal, technical objectives that need to be accomplished, and the specific tasks the system will perform. Also, this section includes an investigation of several popular ready-made solutions for training deep learning models.

The second section introduces the main definitions of linear algebra and describes individual parts of a network. Also, the prevalent method of training models – gradient descent is presented.

The third section describes the actual implementation using Python programming language, goes in-depth about formulas that have been used for coding the network.

Зміст

ВСТУП

Що таке глибинне навчання і чому це важливо?

Моделі глибинного навчання застосовуються в широкому спектрі галузей, включаючи споживчі товари та медичні технології.

Машинне навчання, глибинне навчання, штучний інтелект – ці терміни стали синонімами сучасної епохи; терміни, які люди люблять вживати в розмові в соціальних мережах, на людях, подумки. Тим не менш, правильне розуміння цих термінів допоможе зрозуміти, як деякі найсучасніші технології світу вплинуть на життя людей.

Глибинне навчання – це підмножина машинного навчання, де штучні нейронні мережі, алгоритми, схожі за своєю подобою до людського мозку, вчяться на великій кількості даних. Глибинне навчання та всі аспекти сучасного штучного інтелекту (ШІ) використовують дані для прийняття «розумних» рішень, подібних до людських.

Для того, щоб зрозуміти це в перспективі, для автомобілів без водіїв використовується глибинне навчання, що дозволяє автомобілям розпізнавати інші транспортні засоби, знаки зупинки та навіть пішоходів. Глибинне навчання також лежить у центрі споживчих товарів, як голосовий асистент, керований розумними динаміками, технологія розпізнавання обличчя або розпізнавання рукописного тексту в смартфонах.

В цій галузі дані є ключовими і лежать в основі глибинного навчання. З розвиток інтернету вміння знаходити дані та інтерпретувати їх є цінним вмінням на ринку праці. Людина може навчитися новому вмінню через практику та досвід, моделі глибинного навчання роблять те саме через інформацію. Повернувшись до прикладу автомобіля з самостійним керуванням, комп'ютерна

модель може вивчити тисячі знаків зупинки, перш ніж отримати здатність ідентифікувати знак зупинки.

Глибинне навчання лежить на передньому плані ШІ, допомагаючи формувати інструменти, які ми використовуємо для досягнення надзвичайних рівнів точності. Успіхи глибинного навчання підштовхнули цей інструмент до того, що він перевершує людей у виконанні таких завдань, як класифікація об'єктів у зображеннях.

Такі моделі вже проникли у світ кожного, однаково запровадивши низку проривів у основних галузях, починаючи від світу побутової електроніки, закінчуючи царством аерокосмічної та оборонної діяльності.

Частіше глибинне навчання застосовується в автоматизованих програмах перекладу слуху та мови, що знаходяться в додатках та смарт-пристроях. Програми глибинного навчання допомагають цим системам розпізнавати голос і надавати точні відповіді.

На сьогоднішній день здатність виділити й розпізнати один голос з безлічі розмов, що відбуваються одночасно або голос на фоні музики, являє собою складне перцептивне завдання [1, 2]. Здатність людей робити це надихнула безліч обчислювальних спроб, причому більша частина ранніх робіт була зосереджена на використанні декількох мікрофонів і навчанні без вчителя, наприклад, незалежний компонентний аналіз [3]. Оскільки поділ мовлення з одного мікрофона мав великий стрибок в продуктивності після появи глибинних нейронних мереж, сучасні підходи зосереджуються на проблемі контрольованого поділу мовленні. У цьому завданні "одноканального поділу джерел" при наявності набору даних, що містить як змішаний звук, так і окремі голоси, завданням є навчитися розділяти нові змішані аудіодоріжки, які містить кілька невідомих дикторів.

Метою дослідження випускної кваліфікаційної роботи є аналіз і розробка методу виокремлення мовлення окремих дикторів.

Об’єктом дослідження є рекурентні нейронні мережі.

Предметом – розробка методу навчання рекурентних нейронних мереж і використання з ціллю сегментації звуку.

Завдання дослідження:

- Розглянути основні принципи побудови сучасних нейронних мереж.
- Розглянути та проаналізувати структуру готових рішень для навчання нейронних мереж.
- Засвоїти основні поняття та визначення з лінійної алгебри та усвідомити їхнє геометричне представлення;
- На основі даних аналізу, розробити рішення для відділення цільової мови від фонових перешкод.

Методи дослідження:

- Аналіз – дозволяє розкласти досліджуваний матеріал на одиниці, вивчити окремі частини елементів.
- Синтез – з'єднує частини в ціле, відтворюючи єдине, нове, що взаємодіє з окремих частин.
- Моделювання – переносить реальний об'єкт в створювану модель.
- Дедукція – організовує необхідний перехід від загальних явищ до приватних.
- Аналогія – підводить логічний висновок, через якого знання про один предмет виникають на підставі подібностей з схожими предметами.

РОЗДІЛ 1
ЗАГАЛЬНІ ВІДОМОСТІ ТА ОСНОВНІ ВИМОГИ ДО РЕАЛІЗАЦІЇ СИСТЕМИ

1.1. Загальні відомості

1.1.1. Найменування системи

Повне найменування: бібліотека для виокремлення джерел звуку.

Скорочене найменування системи: voice filter, бібліотека, API, фреймворк.

1.1.2. Планові терміни початку та закінчення робіт

Початок роботи: липень 2021 р.

Закінчення роботи: грудень 2021 р.

1.1.3. Порядок оформлення і пред'явлення результатів робіт

Програмне забезпечення створюється у вигляді функціонуючого комплексу на базі засобів обчислювальної техніки та електротехнічного устаткування у визначені строки. Роботи поетапно в наступній послідовності: етап, зміст роботи і результат.

Після реалізації цих пунктів, результат проведених робіт пред'являється за допомогою презентації.

1.1.4. Головний бенефіціар та потенційні користувачі системи

Головні користувачі: спеціалісти по обробці даних, аналітики даних, інженери машинного навчання, інженери, вчені різних галузей, бізнес-аналітики.

1.2. Мета та призначення створення системи

1.2.1. Призначення системи

При наявності набору даних, що містить як змішаний звук, так і окремі голоси, завданням є навчитися розділяти змішані аудіодоріжки, на кілька окремих дикторів.

1.2.2. Мета створення системи

Метою даної роботи є розробка адаптивного методу навчання рекурентних нейронних мереж для задач класифікації, регресії і прогнозування, з можливістю задавати структуру мережі і параметри всіх необхідних евристик.

1.3. Вимоги до системи

1.3.1. Вимоги в цілому

Кінцевий продукт має представляти з себе високорівневий API нейронних мереж. Він має бути розроблений з акцентом на експериментування та прототипування; спроєктований для уможливлення швидких експериментів з мережами глибинного навчання; зосереджено на тому, щоби він був зручним в користуванні, модульним та розширюваним.

Знаючи загальні вимоги, можна виокремити конкретні першочергові вимоги до структури.

1.3.2. Вимоги до структури

- Має містить численні втілення широко вживаних нейромережевих будівельних блоків, таких як шари, функції витрат та передавальні функції, оптимізатори та деякі інші складові для спрощення кодування, потрібного для написання глибинно-нейромережевого коду;
- має містити підтримку рекурентних мереж.

1.4. Вимоги до компонентів системи:

Основна задача запропонованої системи – це можливість створювати і навчати рекурентні мережі для вирішення різного роду задач в тому числі для сегментації звуку.

1.4.1. Layer (шар)

Клас Layer має бути основним компонентом системи. Найменшим компонентом звичайної нейронної мережі зазвичай являється нейрон, але це поняття ввели лише для поверхневої аналогії з нейронами у мозку людини. В реалізації нейронних мереж кожний окремий компонент векторизується як для простоти роботи так і для ефективності в обчислювані.

Layer має буде батьківським класом - інтерфейсом для всіх реалізацій будь-яких шарів.

1.4.2. Dense

Основний шар в більшості проєктів. Реалізує базовий шар нейронів тісно пов’язаних, які можуть самостійно навчатися. Має містити собі функції прямого та зворотнього поширення. Приклад двох таких шарів наведено на рисунку нижче.

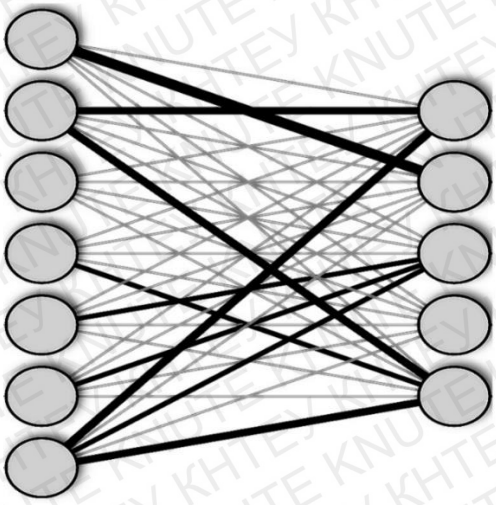


Рис. 1.1 Fully connected layer

Важливим моментом є можливість створювати довільну кількість шарів і в кожному окремому шарі задавати кількість вузлів.

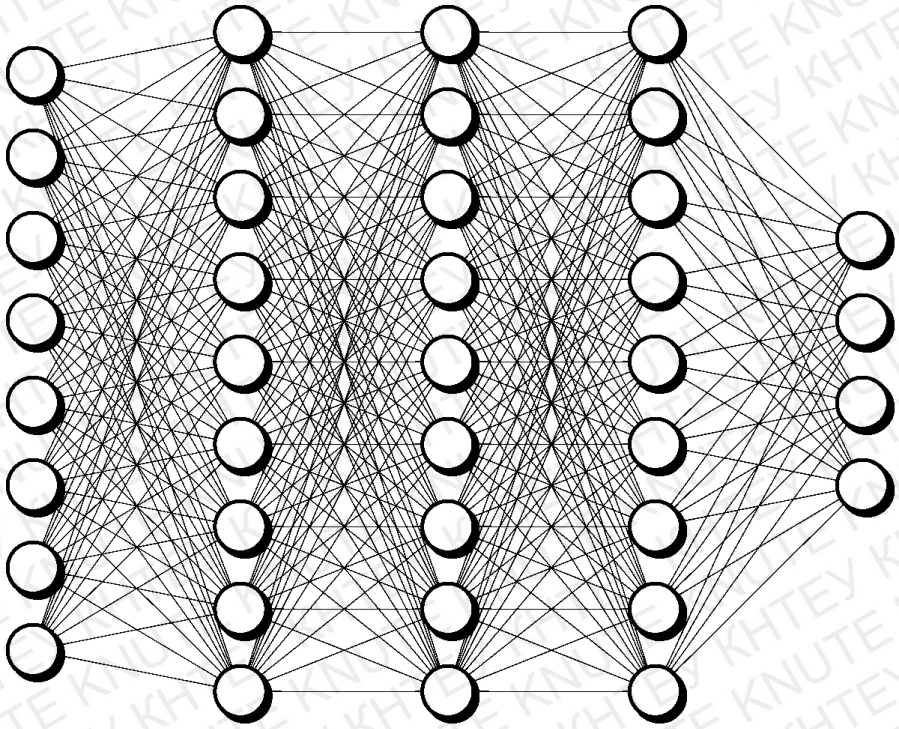


Рис. 1.2. Приклад кінцевої мережі

1.4.3. Activation layer

Ніяка нейронна мережа не обходиться без функції активації. З лінійними функціями активації або взагалі навіть без них мережа не зможе навчитися складним відображенням під час навчання. Оскільки нейрон представляє з себе лише лінійну функцію, то скільки б нейронів, скільки б шарів інженер не використав у своїй моделі – вона все одно буде одною лінійною залежністю між вхідними ознаками.

Звідси походить і їхня важливість. Функції активації додають нелінійності нейронній мережі.

1.4.4. Activations

Має бути присутня реалізація всіх широковживаних функція активації, наприклад: сигмоїда, гіперболічний тангенс, ReLU.

1.4.5 Losses

Функції витрат для обчислення ступеня похибки. Прикладами можуть служити: середнє квадратичне відхилення (mean square error) або абсолютне середнє відхилення (mean absolute error).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \quad (1.1)$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \quad (1.2)$$

1.4.6 Network

Основний клас для об’єднання безлічі різних шарів в одну послідовну мережу. Має реалізовувати методи для навчання, передбачення, прохід в прямому напрямі, прохід у зворотньому, має підтримувати різні види функції витрат.

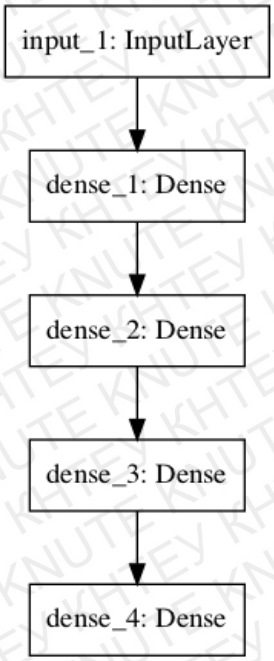


Рис. 1.3. Схематичний приклад об'єднання шарів у мережу

1.5. Огляд рішень

TensorFlow

TensorFlow — відкрита програмна бібліотека для машинного навчання для цілої низки задач, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та навчати нейронні мережі для виявлення та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди. Цю бібліотеку наразі застосовують як для досліджень, так і для розробки продуктів Google [4].

Згідно з офіційним сайтом її також використовують такі компанії, як: Airbnb, AMD, NVidia, UBER, DropBox, Ebay, Google, Snapchat, Intel, and Twitter.

Переваги Tensorflow:

- Підтримка багатьох мов програмування.
- Відносно не складний процес побудови моделей.
- Потужний інструмент для візуалізації даних – TensorBoard.
- Гарна документація.

Theano

Theano – це бібліотека та оптимізувальний компілятор Python для маніпулювання математичними виразами та обчислення їх, особливо матричнозначних. Обчислення в Theano виражаються NumPy-ським синтаксисом і компілюються для ефективного виконання на архітектурі або ЦП, або ГП [5].

Переваги:

- Тісна інтеграція з NumPy .
- Оптимізація швидкості та стабільності.

- Обширна кількість юніт-тестів та самоперевірок – це дозволяє виявити та діагностувати багато видів помилок на ранніх етапах розробки.

Недоліки:

- Оновлюється рідко і тільки силами розробників в open source, через це швидко втрачає актуальність і «старіє».

PyTorch

PyTorch — відкрита бібліотека машинного навчання на основі бібліотеки Torch, що використовують для таких застосувань, як комп'ютерне бачення та обробка природної мови. Розробляє її переважно група дослідження штучного інтелекту компанії Facebook [6].

Переваги:

- Синтаксис нагадує синтаксис Python’а
- Підтримує динамічні обчислювальні графи
- Уможливлює швидке експериментування та прототипування
- Гарна підтримка хмарними технологіями
- Легко дебажити

Scikit-learn

Бібліотека Scikit-learn - найпоширеніший вибір для вирішення завдань класичного машинного навчання. Вона надає широкий вибір алгоритмів навчання з учителем і без вчителя [7].

Переваги:

- Велика кількість користувачів.
- Працює на основі декількох поширених математичних бібліотек, і легко інтегрує їх один з одним.

- Обширна документація.
- Містить алгоритми класичного машинного навчання.
- Просто у використанні, гарно підходить тих хто тільки починає.

Недоліки:

- Бібліотека розроблена більше з акцентом на shallow learning або алгоритми класичного машинного навчання або нейронні мережі малих розмірів.

Keras

Keras - відкрита нейромережева бібліотека, написана на мові Python. Вона являє собою надбудову над фреймворками Deeplearning4j, TensorFlow і Theano. Націлена на оперативну роботу з мережами глибинного навчання [8].

Переваги:

- Являється front-end'ом для багатьох інших бібліотек розробки нейронних мереж.
- Зручна в користуванні
- Високорівневий API

Недоліки:

- Це лише «надбудова» над іншими бібліотеками
- Недостатня документація

1.6. Вимоги до розробки сучасних нейронних мереж

Провівши дослідження з вивчення існуючих рішень для розробки нейронних мереж, можна виокремити загальні характеристику присутні всім фреймворкам глибоко навчання і на їхній основі визначити остаточні вимоги.

- Зручність у користуванні. Фреймворк має пропонувати послідовні та прості високорівневі API, що мінімізують кількість дій користувача, необхідних для випадків загального використання.
- Модульність. Модель – це послідовність або графік самостійних, повністю налаштовуваних модулів, які можна підключити разом з якомога меншими обмеженнями. Зокрема, нейронні шари, функції витрат, оптимізатори, функції активації – це окремі модулі, які можна комбінувати для створення нових моделей.
- Легка розширюваність. Нові модулі легко додавати (як нові класи та і функції). Можливість легко створювати нові модулі дозволяє отримати повну виразність, що робить бібліотеку придатною для передових досліджень.
- Робота з Python. Немає окремих файлів конфігурації моделей у декларативному форматі. Моделі описані в коді Python, який є компактним, простішим у налагодженні та дозволяє легке розширення.
- Інтеграція поширених математичних бібліотек (NumPy, SciPy) та бібліотек роботи з даними: обробки, візуалізації, preprocessing, postprocessing, роботи з файловою системою.

1.6. Висновки до розділу 1

Отже, підводячи підсумок, хотілося б наголосити керівні принципи проекту.

Метою даної роботи є розробка алгоритму навчання рекурентних нейронних мереж для задач які вирішує навчання з вчителем, а саме: класифікації та регресії; використання напрацювань для вирішення проблеми розпізнавання диктора поміж зайвих шумів або інших мовців.

Проведений аналіз готових бібліотек глибокого навчання дозволив виокреслити структуру даної розробки.

Бібліотека має представляти з себе високорівневий API нейронних мереж. Вона має бути розроблена для швидкого експериментування та прототипування; спроєктована для уможливлення швидких експериментів з мережами глибинного навчання; фреймворк має надавати можливість задавати структуру мережі і параметри всіх необхідних змінних (гіперпараметрів); має бути зосереджений на зручності в користуванні, бути модульним та розширюваним.

Уміння переходити від ідеї до результату з найменшою можливою затримкою є ключовим для хорошого дослідження.

РОЗДІЛ 2
АНАЛІЗ ПРЕДМЕТА ДОСЛІДЖЕННЯ ТА ВИЗНАЧЕННЯ ОСНОВНИХ КОМПОНЕНТІВ СИСТЕМИ

Нейронні мережі – це набір алгоритмів, змодельованих за зразком мозку людини, які розроблені для розпізнавання шаблонів. Вони інтерпретують сенсорні дані через своєрідне машинне сприйняття, позначаючи чи групуючи вхідні дані. Шаблони, які вони розпізнають, є чисельними, містяться у векторах, у які повинні бути переведені всі дані з реального світу, наприклад: зображення, звук, текст чи часовий ряд.

Нейронні мережі допомагають класифікувати та кластеризувати. Їх можна розглядати як відповідний шар поверх даних користувача. Вони допомагають групувати немарковані дані відповідно до подібності серед прикладів, які подаються на вхід мережі, і вони класифікують дані, коли у них є мічений набір даних для навчання. Нейронні мережі також можуть витягувати ознаки, які подаються в інші алгоритми кластеризації чи класифікації; тому глибокі нейронні мережі можуть бути компонентами більш великих додатків машинного навчання, що включають алгоритми навчання з підкріплення, класифікації та регресії.

Далі в розділі буде йтися про деякі поняття нейронних мереж, втому числі рекурентних, як вони працюють, яка у них структура, як вони навчаються. Всі ці теоретичні знання є необхідними для написання своїх нейронних мереж.

2.1. Основні поняття і позначення

Нейронні мережі насправді представляють з себе складні алгоритми і функції, тому початковий, найпростіший термін який необхідно знати для розуміння роботи мереж це означення функції.

Функція в математиці — це правило, яке кожному елементу з першої множини – області визначення ставить у відповідність елемент з іншої множини – області значень. Часто цю другу множину називають цільовою множиною чи образом функції чи відображення.

Відображення f, яке ставить у відповідність кожному елементові множини A єдиний елемент множини B позначається як $f: \mathrm{A} \to \mathrm{B}$ (тобто f відображує A в B).

2.1.1. Теорема апроксимації

Не всяку множину A можна точно відобразити у множину B. Мережа тільки знаходить кореляції між даними. Навіть якщо точно розв’язку не існує, функція може безкінечно близько наблизитися до правильного результату. Доказом цього є універсальна теорема апроксимації [9] — теорема, доведена Джорджем Цибенком в 1989 році, яка стверджує, що штучна нейронна мережа прямого зв'язку (у яких зв'язки не утворюють циклів) з одним прихованим шаром може апроксимувати будь-яку неперервну функцію багатьох змінних з будь-якою точністю. Умовами є достатня кількість нейронів прихованого шару, вдалий підбір $w_1, w_2, \dots w_N, \alpha, \theta$, де

- w_1 – ваги між вхідними нейронами і нейронами прихованого шару
- α – це ваги між зв'язками від нейронів прихованого шару і вихідним нейроном
- θ – це коефіцієнт «упередженості» для нейронів прихованого шару.

Алгоритм навчання може наблизити невідому функцію $f(x) = y$ між будь-яким входом x і будь-яким виходом y, якщо вважати, що вони взагалі пов'язані (наприклад, за допомогою кореляції чи причинного зв'язку). У процесі навчання нейронна мережа знаходить правильну f або правильну манеру перетворення x на у, будь то $f(x) = 7 + 6x$ або $f(x) = 4x - 0.9$.

2.1.2. Прикладне застосування

Декілька прикладів того, що глибоке навчання може зробити:

Класифікація

Усі завдання класифікації залежать від уже розмічених наборів даних; тобто такий набір даних, де кожному вхідному значенню X співвідноситься уже відоме правильне вихідне значення Y щоб нейронна мережа дізналася про співвідношення міток і даних. Це відомо як навчання з учителем.

- Виявити обличчя, визначити людей на зображеннях, розпізнати міміку (сердитий, радісний)
- Визначити предмети на зображеннях (знаки зупинки, пішохідні переходи, дорожня розмітка)
- Розпізнати жести
- Розпізнати голоси, розпізнати ораторів, перевести мовлення у текст, розпізнати почуття у голосах
- Класифікувати текст як спам (в електронних листах) або шахрайський (у страхових вимогах); розпізнати почуття в тексті (відгуки клієнтів)

Будь-які мітки, які можуть створювати люди, будь-які результати, які мають цінність і які співвідносяться з даними, можуть використовуватись для навчання нейронної мережі.

Кластеризація

Кластеризація чи групування - це виявлення подібності. Для вирішення проблем виявлення подібності використовуються дані без міток. Такі дані – це більшість даних у світі. Навчання без позначок називається навчання без учителя.

- Пошук: порівняння документів, зображень чи звуків для знаходження подібних даних.
- Виявлення аномалії: протилежність знаходження подібності – виявлення аномалій чи незвичної поведінки. У багатьох випадках незвична поведінка сильно співвідноситься з речами, які ви хочете виявити та запобігти, наприклад, шахрайству.

2.2 Рекурентні нейронні мережі

Люди не починають думати з чистого аркуша що секунди. Читаючи текст, людина розуміє кожне слово, ґрунтуючись на розумінні попереднього слова. Люди не викидають з голови все і не починають думати з нуля. Людські думки мають сталість.

Традиційні нейронні мережі не мають цієї властивості, і в цьому їхній головний недолік. Наприклад, потрібно класифікувати події, що відбуваються у фільмі. Незрозуміло, як традиційна нейронна мережа могла б використати міркування про попередні події фільму, щоб отримати інформацію про наступні.

Вирішити цю проблему допомагають рекурентні нейронні мережі (Recurrent Neural Networks, RNN). Це мережі, що містять зворотні зв'язки та дозволяють зберігати інформацію.

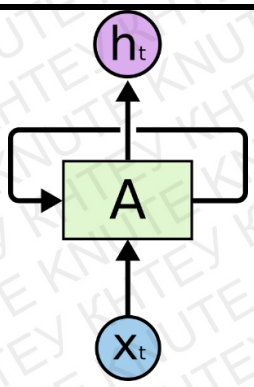


Рис. 2.1 RNN

Рекурентні нейронні мережі містять зворотні зв'язки.

На рис. 2.1 фрагмент нейронної мережі A набуває вхідного значення x_t і повертає значення h_t. Наявність зворотного зв'язку дозволяє передавати інформацію від кроку мережі до іншого.

Рекурентну мережу можна розглядати як кілька копій однієї і тієї ж мережі, кожна з яких передає інформацію наступної копії.

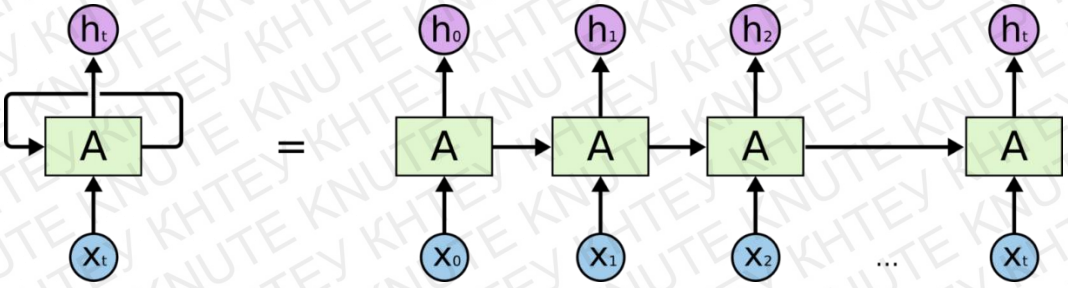


Рис. 2.2 Розгорнута рекурентна нейронна мережа

Те, що RNN нагадують ланцюжок, свідчить, що вони тісно пов'язані з послідовностями та списками. RNN – природна архітектура нейронних мереж для роботи з даними такого типу.

За останні кілька років RNN з неймовірним успіхом застосували до цілого ряду завдань: розпізнавання мовлення, мовне моделювання, переклад, розпізнавання зображень. Чимала роль цих успіхах належить LSTM – модифікації рекурентної нейронної мережі, що у багатьох завданнях значно

перевищує стандартну версію. Майже всі вражаючі результати RNN досягнуто саме за допомогою LSTM.

Проблема довготривалих залежностей

Одна з привабливих ідей RNN полягає в тому, що вони потенційно вміють пов'язувати попередню інформацію з поточним завданням, наприклад, знання про попередній кадр відео можуть допомогти в розумінні поточного кадру. Якби RNN мали таку здатність, вони були б надзвичайно корисні. На практиці це залежить від деяких обставин.

Іноді для виконання завдання потрібна лише нещодавня інформація. Наприклад, для мовної моделі, яка намагається передбачити наступне слово виходячи з попередніх. Якщо потрібно передбачити останнє слово у реченні “хмари пливуть небом”, ширший контекст не потрібен; тут досить очевидно, що останнім словом буде “небом”. У цьому випадку, коли дистанція між актуальною інформацією та місцем, де вона знадобилася невелика, RNN можуть навчитися використовувати інформацію з минулого.

Бувають випадки, коли потрібно більше контексту. Припустимо, потрібно передбачити останнє слово у тексті “Я виріс у Франці … Я говорю по-французьки”. Найближчий контекст передбачає, що останнім словом буде назва мови, але щоб встановити, якої саме мови, нам потрібен контекст Франції з більш віддаленого минулого. Таким чином, розрив між актуальною інформацією та точкою її застосування може стати дуже великим.

На жаль, у міру зростання цієї відстані RNN втрачають здатність зв'язувати інформацію. Теоретично проблеми з обробкою довгострокових залежностей у RNN бути не повинно. Людина може акуратно підбирати параметри мережі для вирішення штучних завдань такого типу. На жаль, практично навчити RNN цим

параметрам здається неможливим. LSTM в свою чергу має свої шляхи вирішення таких проблем.

2.3 Мережі LSTM

Довга короткочасна пам'ять (ДКЧП, англ. Long short-term memory, LSTM) – особливий різновид архітектури рекурентних нейронних мереж, здатна до вивчення довгострокових залежностей, такі мережі чудово вирішують цілу низку різноманітних завдань і нині широко використовуються.

LSTM спеціально розроблені щоб уникнути проблеми довготривалої залежності. Запам'ятовування інформації на довгі періоди часу – це їхня звичайна поведінка, а не щось, чого вони намагаються навчитися.

Будь-яка рекурентна нейронна мережа має форму ланцюжка повторюваних модулів нейронної мережі. У звичайній RNN структура одного такого модуля дуже проста, наприклад, він може бути одним шаром з функцією активації tanh (гіперболічний тангенс). Модуль, що повторюється, в стандартній RNN складається з одного шару.

Структура LSTM також нагадує ланцюжок, але модулі виглядають інакше. Замість одного шару нейронної мережі вони містять чотири, і ці шари взаємодіють особливим чином. Модель, що повторюється, в LSTM мережі складається з чотирьох взаємодіючих шарів.

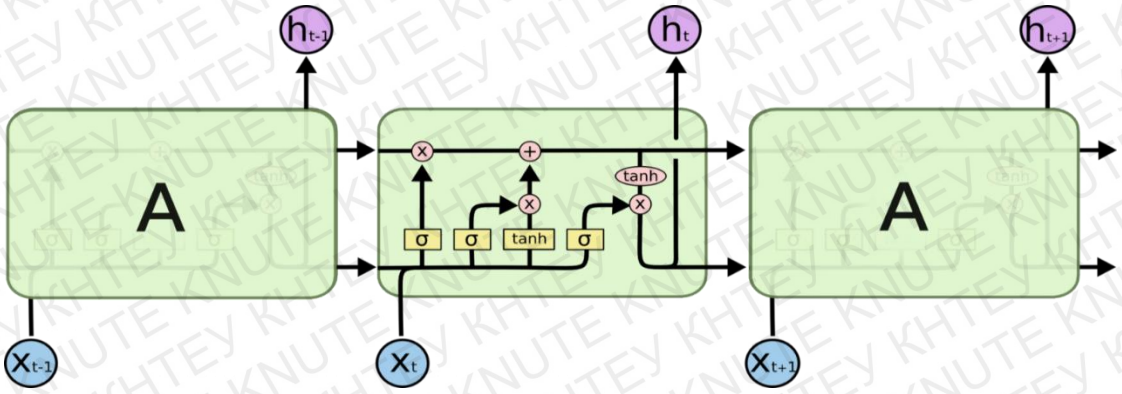


Рис. 2.3 LSTM ланцюжок

Ключовий компонент LSTM – це стан комірки (cell state) – це горизонтальна лінія, що проходить по верхній частині схеми.

Стан комірки нагадує конвеєрну стрічку. Вона проходить безпосередньо через весь ланцюжок, беручи участь лише у кількох лінійних перетвореннях. Інформація може легко текти по ній, не зазнаючи змін.

Тим не менш, LSTM може видаляти інформацію зі стану комірки; цей процес регулюється структурами, які називаються вентилями (gates). Вентилі дозволяють пропускати інформацію на підставі деяких умов. Вони складаються з шару сигмоїдальної нейронної мережі та операції по елементного множення. Сигмоїдальний шар повертає числа від нуля до одиниці, які позначають, яку частку кожного блоку інформації слід пропустити далі через мережу. Нуль у разі означає “не пропускати нічого”, одиниця – “пропустити все”.

У LSTM три таких фільтри, що дозволяють захищати та контролювати стан комірки.

Покроковий аналіз LSTM

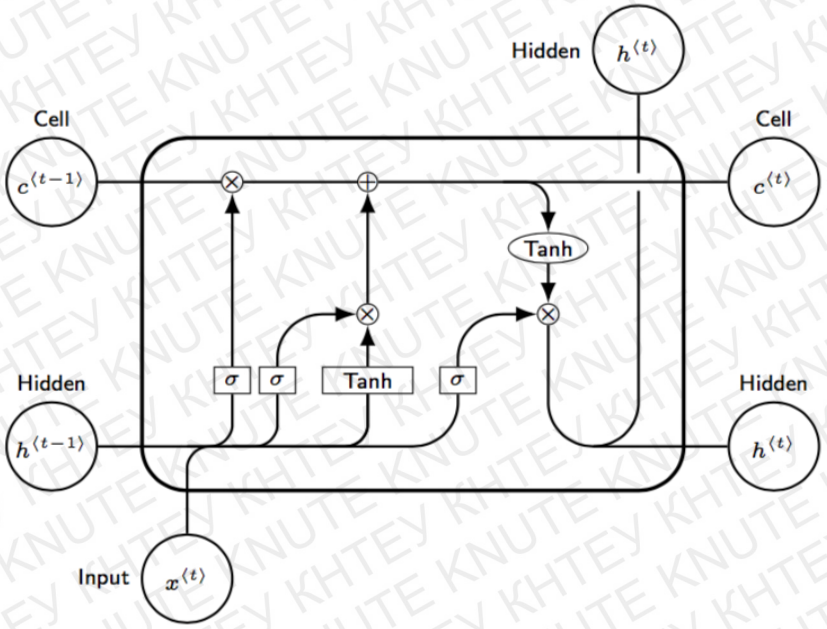


Рис. 2.4 LSTM вузол

Перший крок у LSTM – визначити, яку інформацію можна викинути зі стану комірки. Це рішення приймає сигмоїдальний шар, званий вентилем забування (forget gate layer). Він дивиться на h_{t-1} і x_t і повертає число від 0 до 1 для кожного значення з комірки C_{t-1}. 1 означає "повністю зберегти", а 0 - "повністю викинути".

$$f_t = \sigma(W_t * [h_{t-1}, x_t] + b_f) \quad (2.1)$$

Наступний крок – вирішити, яка нова інформація зберігатиметься у комірці. Цей етап складається із двох частин. Спочатку сигмоїдальний шар під назвою вхідний вентиль (input layer gate) визначає, які значення слід оновити. Потім tanh-шар будує вектор нових значень-кандидатів C_t, які можна додати до комірки.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \quad (2.2)$$

$$\tilde{C}_t = tanh(W_C * [h_{t-1}, x_t] + b_C) \quad (2.3)$$

Далі потрібно замінити старий стан комірки C_{t-1} на новий стан C_t. Спочатку треба помножити старий стан на f_t, забуваючи про те, що було вирішено забути. Потім треба додати $i_t * C_t$ – нові значення-кандидати.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (2.4)$$

Зрештою, потрібно вирішити, яку інформацію треба отримати на виході. Вихідні дані будуть засновані на стані комірки але до них будуть використані деякі фільтри. Спочатку застосовується сигмоїдальний шар, який вирішує, яка інформація буде виведена з h_t. Потім значення з комірки проходять через tanh-шар, щоб отримати на виході значення з діапазону від -1 до 1, і перемножуються

з вихідними значеннями сигмоїдального шару, що дозволяє виводити тільки необхідну інформацію.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (2.5)$$

$$h_t = o_t * (C_t) \quad (2.6)$$

Функції активації

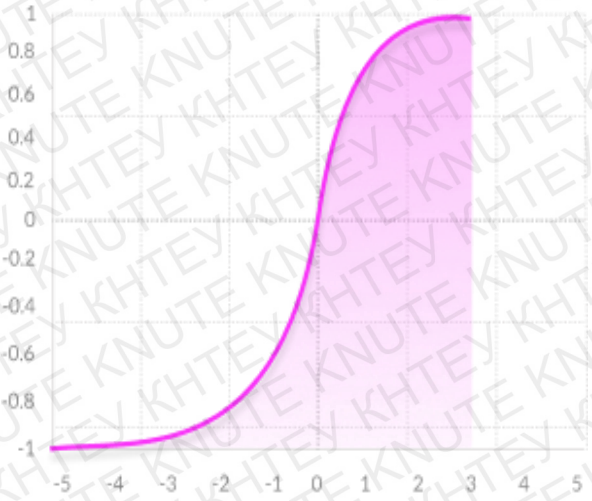
1. 

Рис. 2.5 Sigmoid

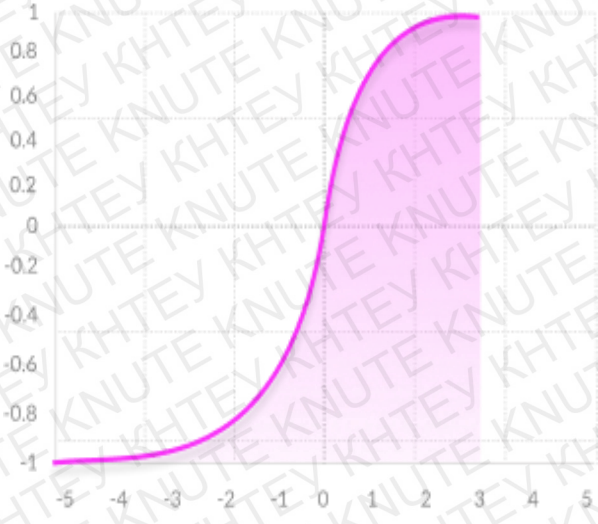
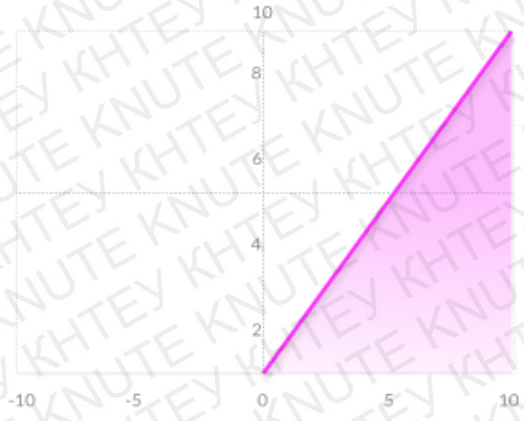
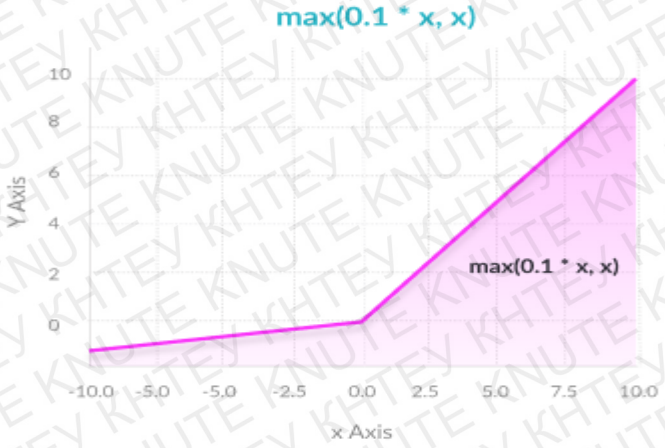
2. 

Рис. 2.6 Tanh



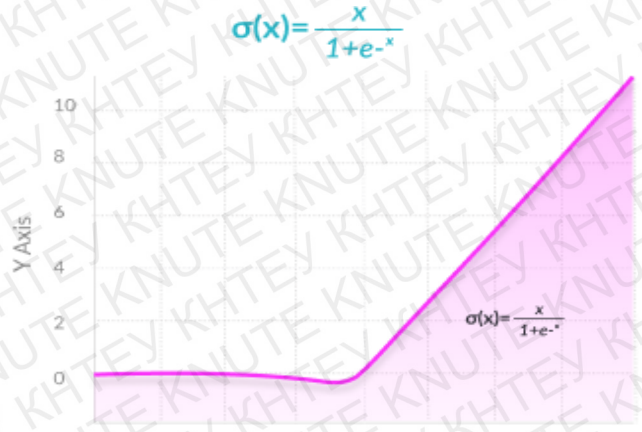
3.

Рис. 2.7 ReLU



4.

Рис. 2.8 Leaky ReLU



5.

Рис. 2.9 Swish

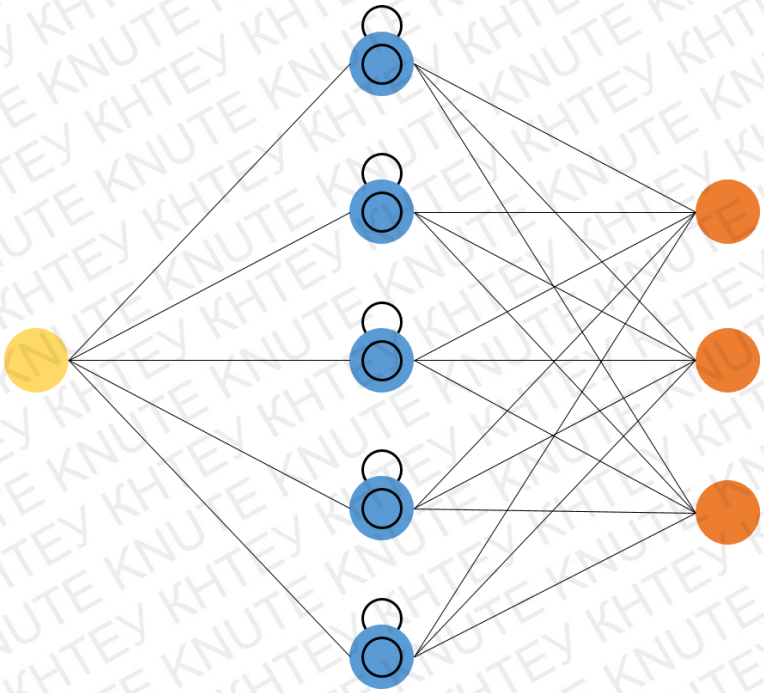


Рис. 2.10 Схематична структура RNN

2.4. Метод зворотнього поширення помилки

Перед написанням своєї бібліотеки для глибокого навчання потрібно дати відповідь на ще одне, останнє питання – як модель вчиться?

2.4.1. Функція витрат

Ціль процесу навчання – мінімізувати функцію витрат. Саме функція витрат показує наскільки добре мережа працює. Фактично вона показує наскільки близько до цілі модель в будь-який момент часу. Функціонал якості є сумою функцій втрат на навчальної вибірки об’єктів.

$$Q(w) = \frac{1}{n}\sum_{i=1}^{l} L(a(x_i), y_i) \to \min \qquad (2.7)$$

де $L(a(x_i), y_i)$ – це задана функція втрат, що характеризує величину помилки відповіді a при правильній відповіді y. Найбільш універсальною є квадратична функція втрат.

$$L\left(a\left(x_i\right), y_i\right) = \left(a(x_i) - y_i\right)^2 \quad (2.8)$$

Залежно від завдання функція втрат може приймати необхідний вид. Наприклад, функція втрат може сильніше штрафувати позитивне відхилення від мети і менш сильно негативне.

Backpropagation

Яким же чином ваги нейронної мережі оновлюються? Цей процес називається – метод зворотнього поширення помилки або **backpropagation (backprop)**. Сам алгоритм, який визначає як оновлюються ваги варіюється від моделі до моделі з мінімальними змінами тому що в основі всіх лежить метод градієнтного спуску або **gradient descent**.

2.4.2. Gradient descent

Припустимо, що ми знаходимося на вершині гори вночі і ми хочемо якнайшвидше спуститися вниз. Оскільки ми не бачимо далі ніж декілька метрів, ми інтуїтивно припускаємо, що оскільки вершина гори є «найвищою» точкою гори, то найбільш крутий шлях веде нас до самої нижньої точки найбільш ефективно. Ми підходимо до цього виклику ітеративно, на кожному кроці роблячи крок у бік самого стрімкого спуску.

По суті, це була аналогія методу градієнтного спуску, де гора це область значень, а ми – це змінна, вага в мережі.

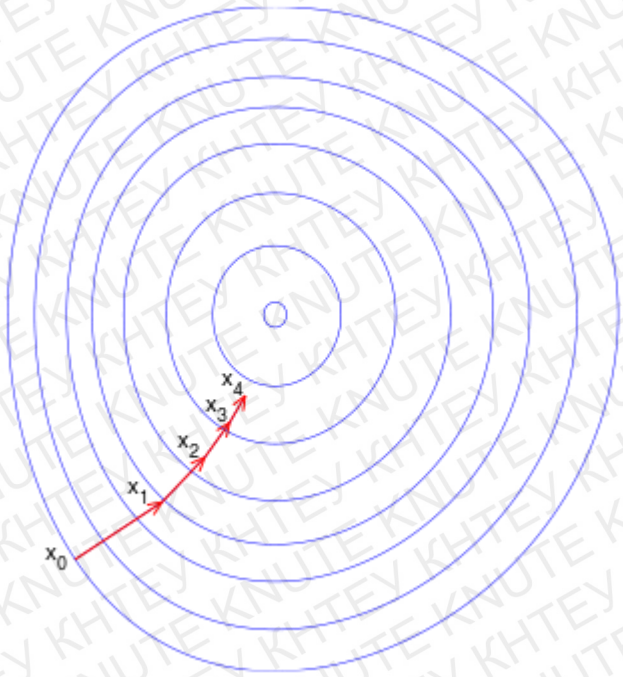


Рис. 2.11 Ілюстрація методу найшвидшого спуску на ряді множин рівнів.

На кожній ітерації алгоритму нейронна мережа змінює свої ваги таким чином, щоб мінімізувати свою помилку, яка задається функцією витрат.

По мірі того, як нейронна мережа вчиться, вона повільно коригує багато значень, щоб вони могли правильно відображати сигнал і значення. Взаємозв'язок між помилкою мережі та кожною з цих ваг є похідною, dE / dw, яка вимірює ступінь, до якої незначна зміна ваги викликає незначну зміну похибки. Оскільки похідна показує приріст функції, а нас цікавить спадання, то береться похідна і множиться на -1. Таким чином отримується «найшвидший спуск» функції, звідси і назва **метод найшвидшого спуску.**

Кожна вага є лише одним фактором у глибокій мережі, яка передбачає багато перетворень; сигнал ваги проходить через функції активації і додається протягом кількох шарів, тому використовується ланцюгове правило (**правило диференціювання складної функції)** або **Chain rule** [10], щоб повернутись

через активації та виходи мереж і, нарешті, дійти до ваги, про яку йде мова, та її відношення до загальної помилки.

Ланцюгове правило стверджує, що

$$\frac{dz}{dx} = \frac{dz}{dy} * \frac{dy}{dx} \tag{2.9}$$

У мережі прямого поширення зв’язок між помилкою мережі та однією вагою буде виглядати приблизно так:

$$\frac{d\text{Помилка}}{d\text{вага}} = \frac{d\text{Помилка}}{d\text{активація}} * \frac{d\text{активація}}{d\text{вага}} \tag{2.10}$$

Тобто, маючи дві змінні, помилку та вагу, третю змінну, активацію, через яку передається вага, можна обчислити, як зміна ваги впливає на зміну помилки, спочатку обчисливши, як впливає зміна активації на зміну помилки та те, як зміна ваги впливає на зміну активації.

Суть глибинного навчання – це не що інше, як коригування ваги моделі у відповідь на помилку, яку вона створює, допоки помилку більше зменшити не вдається.

2.5 Висновки до розділу 2

Отже, в цьому розділі було описано основні поняття і принципи на яких побудовані нейронні мережі, зокрема, в контексті даної роботи, рекурентні нейронні мережі.

Основні блоки нейронної мережі: нейрон (вузол), функція активації. функція витрат, оптимізатор. Вона разом складають більш високорівневі структури такі як: вхідний шар, прихований шар, вихідний шар.

Також, було ретельно досліджено метод зворотнього поширення помилки (backpropagation), спосіб диференціювання складної функції (chain rule), градієнтний спуск (gradient descent).

Розуміння математичного змісту функцій, знання необхідних елементів будь-якої мережі та формул якими ці елементи керуються та процесу навчання має вирішальне значення для написання коду.

РОЗДІЛ 3

ІМПЛЕМЕНТАЦІЯ НЕЙРОННОЇ МЕРЕЖІ

3.1. Теорія необхідна для реалізації

Спочатку варто нагадати дуже узагальнений процес навчання нейронної мережі:

1. Спочатку на вхід нейронної мережі подаються вхідні значення – оброблений набір даних.
2. Інформація перетікає з одного шару в інший допоки не буде отримане вихідне значення.
3. Коли вихідне значення отримане, можна порахувати похибку за допомогою функції витрат, це значення є скаляром.
4. На кінець, значення мережі оновлюються шляхом віднімання похідної помилки в залежності від параметра який зараз оновлюється. Цей крок є окремим ітеративним процесом.

Найважливіший це четвертий крок. Вимога до мережі – це мати так багато шарів як забажає інженер, що буде використовувати бібліотеку. Крім того шари мають бути будь-яких типів. Але якщо хтось захоче модифікувати/додати/прибрати один шар з мережі, вихідне значення мережі також зміниться, що вплине на помилку, яка в свою чергу змінить похідну помилки відносно параметрів. Потрібно вміти вираховувати похідні незалежно від архітектури нейромережі, незалежно від функцій активації, незалежно від функції витрат, яка використовується в моделі.

Для того щоб це досягти, кожен шар має бути розроблений окремо, в окремому файлі, окремим класом.

Що має реалізовувати кожний шар

Кожний шар, який може бути створений (fully connected, convolutional, maxpooling, dropout, etc.) повинен мати хоча б два однакові параметри: **вхідна та вихідна інформація**.

Рис. 3.1. Схематичний інтерфейс будь-якого якого шару при прямому поширенні

Пряме поширення

Уже можна виокремити один важливий факт: вихідні значення одного шару являються вхідними значеннями наступного шару.

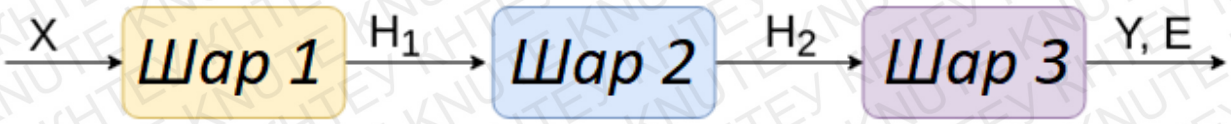


Рис. 3.2. Залежність між шарами

Це є пряме розповсюдження. По суті, вхідна інформація подається на вхід першому шару, потім вихідна інформація кожного шару стає вхідними даними всіх наступних шарів поки не досягне кінця мережі.

Gradient Descent

Тут задача полягає в тому, щоб змінити довільний параметр в мережі (нехай це буде w) таким чином щоб загальна помилка E зменшилася. Робиться це наступним чином:

$$w \leftarrow w - \alpha \frac{\partial E}{\partial w} \tag{3.1}$$

де α це параметр [0,1], що задається вручну і називається **learning rate**. Важливий тут вираз $^{\partial E}/_{\partial w}$ (похідна E по w). Потрібно знайти значення цього виразу для будь-якого параметру мережі не залежно від архітектури самої мережі.

Backward propagation

Нехай шар отримує похідну похибки відносно свого вихідного значення ($^{\partial \mathrm{E}}/_{\partial Y}$), тоді логічно, що він має змогу порахувати похибку відносно свого вхідного значення ($^{\partial \mathrm{E}}/_{\partial X}$).

Рис. 3.3. Схематичний інтерфейс будь-якого якого шару при зворотньому поширенні

Варто пам’ятати, що E є скаляр (число**), а** X і Y матриці.

Вся ідея в тому що, маючи $^{\partial \mathrm{E}}/_{\partial Y}$ порахувати $^{\partial \mathrm{E}}/_{\partial w}$ стає тривіальною задачою. Тут вступає в силу диференціювання складної функції:

$$\frac{\partial E}{\partial x_i} = \sum_j \frac{\partial E}{\partial y_i} \frac{\partial y_j}{\partial w} \tag{3.2}$$

Невідомий параметр $\partial y_i / \partial w$ повністю залежить від того як шар вираховує своє вихідне значення. Звідси якщо кожний шар маж доступ до $\partial E / \partial Y$, де Y це своє власне вихідне значення.

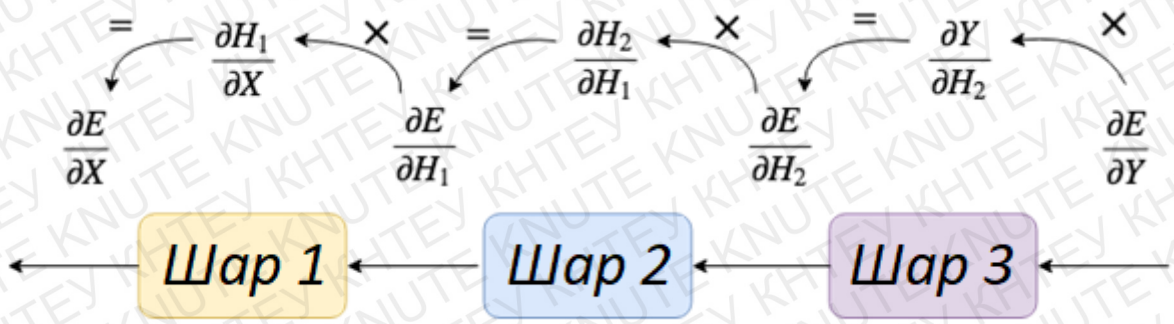


Рис. 3.4. Передача між шарами кількісного значення функції витрат

Шар №3 змінить свої параметри використавши $\partial E / \partial Y$, потім передасть $\partial E / \partial H_2$ попередньому шару, для якого це уже буде своїм власним $\partial E / \partial Y$. Шар №2 потім зробить це саме і так далі аж до самого першого шару.

Цієї інформації достатньо для написання першого, базового шару нейронної мережі.

3.2. Реалізація

3.2.1. Абстрактний базовий клас: Layer

Абстрактний клас *Layer*, який будуть наслідувати всі інші шари, обробляє прості ознаки такі як вхідні та вихідні значення, а також методи як прямого так і зворотнього поширення.

```python
class Layer:
    def __init__(self):
        self.input = None
        self.output = None

    # computes the output Y of a layer for a given input X
    def forward_propagation(self, input):
        raise NotImplementedError

    # computes dE/dX for a given dE/dY (and update parameters if any)
    def backward_propagation(self, output_error, learning_rate):
        raise NotImplementedError
```

Рис. 3.5. Клас *Layer*

Також тут з'являється уже вищесказаний гіперпараметр *learning rate* або швидкість навчання. Це той самий скаляр на який домножається градієнт при оновленні значень. В різних інших бібліотеках він називається по різному, але суть залишається однакова: він задає спосіб оновлення параметрів в мережі, а саме те з якою швидкістю це відбувається.

3.2.2. Fully connected layer

Тепер час перейти до першого типу шару: повнозв’язного шару або fully connected layer або FC шар. FC шар э найбільш базовим, де кожний вихідний нейрон пов’язаний з кожним вхідним.

Forward Propagation

За допомогою матриць цю формулу можна легко обчислити використовуючи скалярний добуток. Векторизований варіант формул:

$$Y = XW + B \tag{3.3}$$

де $X = [x_1 \ \dots \ x_i]$, $W = \begin{bmatrix} w_{11} & w_{1j} \\ w_{i1} & w_{ij} \end{bmatrix}$, $B = [b_1 \ \dots \ b_i]$

Backward Propagation

Нехай шар отримує на вхід загальну похибку відносно вихідних даних цього шару ($^{\partial \mathrm{E}}/_{\partial Y}$), тоді потрібно порахувати:

1. Похідну відносно кожного свого параметру ($^{\partial \mathrm{E}}/_{\partial w}$, $^{\partial \mathrm{E}}/_{\partial B}$)
2. Похідну відносно вхідного значення ($^{\partial \mathrm{E}}/_{\partial X}$).

Формула для одного окремого значення $^{\partial \mathrm{E}}/_{\partial w}$:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_1}{\partial w_{ij}} + \cdots + \frac{\partial E}{\partial y_i}\frac{\partial y_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}x_i \tag{3.4}$$

Векторизована формула:

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial y_1}x_1 & \cdots & \frac{\partial E}{\partial y_i}x_1 \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial y_1}x_i & \cdots & \frac{\partial E}{\partial y_i}x_i \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_i \end{bmatrix}\begin{bmatrix} \frac{\partial E}{\partial y_1} & \cdots & \frac{\partial E}{\partial y_j} \end{bmatrix} = X^t\frac{\partial E}{\partial Y} \tag{3.5}$$

Отже, три формули необхідні для написання FC шару:

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y}W^T \tag{3.6}$$

$$\frac{\partial E}{\partial X} = X^t\frac{\partial E}{\partial Y} \tag{3.7}$$

$$\frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y} \tag{3.8}$$

Реалізація:

```
import numpy as np

from layer import Layer


# inherit from base class Layer
class FCLayer(Layer):
    # input_size = number of input neurons
    # output_size = number of output neurons
    def __init__(self, input_size, output_size):
        self.weights = np.random.rand(input_size, output_size) - 0.5
        self.bias = np.random.rand(1, output_size) - 0.5

    # returns output for a given input
    def forward_propagation(self, input_data):
        self.input = input_data
        self.output = np.dot(self.input, self.weights) + self.bias
        return self.output
```

Рис. 3.6. Python код для повнозв'язного шару (1)

```
    # computes dE/dW, dE/dB for a given output_error=dE/dY. Returns input_error=dE/dX.
    def backward_propagation(self, output_error, learning_rate):
        input_error = np.dot(output_error, self.weights.T)
        weights_error = np.dot(self.input.T, output_error)
        # dBias = output_error

        # update parameters
        self.weights -= learning_rate * weights_error
        self.bias -= learning_rate * output_error
        return input_error
```

Рис. 3.7. Python код для повнозв'язного шару (2)

3.2.3. Шар активації

Всі перетворення до цього моменту були лінійними. Для того щоб мережа могла вивчити складні відображення потрібно додати нелінійності. Зробимо це шляхом додавання нелінійності до вихідних значень деяких нейронів.

Тепер задача є простіша тому, що у функціях активації немає параметрів які можуть навчатися. Все що треба зробити – це обчислити $\partial E / \partial X$.

Нехай f та f' це функція активації та її похідна.

Forward Propagation

Для прямої передачі все просто. Для деякого вхідного значення X, вихідне значення це функція активація застосована до кожного елемента окремо. Це означає що розмірність вихідного значення дорівнює розмірності вхідного.

$$\mathrm{Y} = [f(x_1) \quad \dots \quad f(x_i)] = f(X) \tag{3.9}$$

Backward Propagation

Для $\partial \mathrm{E}/\partial Y$ потрібно обрахувати $\partial \mathrm{E}/\partial X$.

$$\begin{aligned}
\frac{\partial E}{\partial X} &= \left[\frac{\partial E}{\partial x_1} \quad \dots \quad \frac{\partial E}{\partial x_i}\right] \\
&= \left[\frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_1} \quad \dots \quad \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i}\right] \\
&= \left[\frac{\partial E}{\partial y_1} f'(x_1) \quad \dots \quad \frac{\partial E}{\partial y_1} f'(x_i)\right] \\
&= \left[\frac{\partial E}{\partial y_1} \quad \dots \quad \frac{\partial E}{\partial y_1}\right] \odot [f'(x_1) \quad \dots \quad f'(x_i)] \\
&= \frac{\partial E}{\partial Y} \odot f'(X)
\end{aligned} \tag{3.10}$$

Важливо відмітити, що тут поелементне множення двох матриць (у формулах вище використовується скалярний добуток).

Реалізація

```python
from layer import Layer

# inherit from base class Layer
class ActivationLayer(Layer):
    def __init__(self, activation, activation_prime):
        self.activation = activation
        self.activation_prime = activation_prime

    # returns the activated input
    def forward_propagation(self, input_data):
        self.input = input_data
        self.output = self.activation(self.input)
        return self.output

    # Returns input_error=dE/dX for a given output_error=dE/dY.
    # learning_rate is not used because there is no "learnable" parameters.
    def backward_propagation(self, output_error, learning_rate):
        return self.activation_prime(self.input) * output_error
```

Рис. 3.8. Python код для шару активації

3.2.4. Функція активації

Шар активації в конструкторі приймає конкретну функцію активацію яку потрібно використати. Створимо окремий файл з усіма функціями.

```python
import numpy as np


# activation function and its derivative
def tanh(x):
    return np.tanh(x)


def tanh_prime(x):
    return 1 - np.tanh(x) ** 2


def softmax(x):
    e = np.exp(x)
    return e / np.sum(e, axis=1, keepdims=True)


def softmax_prime(x):
    return np.exp(x) / np.sum(np.exp(x), axis=1, keepdims=True)
```

Рис. 3.9. Реалізація функцій активації

3.2.5. Функція витрат

До цього моменту кожний шар отримував $\partial E/\partial Y$ від наступного по черзі, але звідки взяти витрати для останнього шару? Для цього і використовується функція витрат.

Помилка мережі показує наскільки добре або погано відпрацювала мережа і вона визначається вручну. Є багато різних функції але дві найпопулярніші: середнє квадратичне відхилення та середнє абослютне відхилення.

Формула середнього квадратичного відхилення або **Mean Squared Error (MSE)**:

$$E = \frac{1}{n}\sum_{i}^{n}(y_i^* - y_i)^2 \tag{3.11}$$

Реалізація:

```
import numpy as np


# loss function and its derivative
def mse(y_true, y_pred):
    return np.mean(np.power(y_true - y_pred, 2))


def mse_prime(y_true, y_pred):
    return 2 * (y_pred - y_true) / y_true.size
```

Рис. 3.10. Середнє квадратичне відхилення

Формула середнього абсолютного відхилення або **Mean Absolute Error (MAE)**:

$$E = \frac{1}{n}\sum_{i}^{n}(y_i^* - y_i)^2 \tag{3.12}$$

Реалізація:

```
def mae(y_true, y_pred):
    return np.mean(np.abs(y_true - y_pred))


def mae_prime(y_true, y_pred):
    gradient = y_true - y_pred
    pos = (gradient > 0) * -1
    neg = (gradient < 0)
    return pos + neg
```

Рис. 3.11. Середнє абсолютне відхилення

3.2.6. Network клас

Залишилося реалізувати останній клас, задача якого об'єднати всі минулі класи в єдину мережу. В цьому класі не буде нічого нового, всі теоретичні знання вже були подані і роз'яснені. На рис. 3.12 представлена лише частина коду, повна реалізація наведена в Додатку А.

```
class Network:
    def __init__(self):
        self.layers = []
        self.loss = None
        self.loss_prime = None

    # add layer to network
    def add(self, layer):
        self.layers.append(layer)

    # set loss to use
    def use(self, loss, loss_prime):
        self.loss = loss
        self.loss_prime = loss_prime
```

Рис. 3.12. Реалізація класу *Network*

3.3. Побудова моделі

3.3.1. Підготовка датасету

Для навчання потрібні дані. Більшість великих, якісних датасетів на будь-яку тематику є платними або зібрані силами вчених та дослідників великих корпорацій безпосередньо під час використання продукту користувачами,

наприклад Google, Facebook тощо. Для кращого розуміння, широко відомий WSJ0 [11] коштує $1500, тому в данній розробці будуть використовуватися тільки open source датасети.

OpenSLR – це сайт, присвячений розміщенню мовних та розмовних ресурсів, таких як навчальні корпуси та програмне забезпечення, пов'язане з розпізнаванням мовлення. Це зручне місце, де будь-хто може розмістити створені ним ресурси, щоб їх можна було завантажити у відкритому доступі [12].

LibriSpeech ASR corpus – це корпус приблизно з 1000 годин читання англійської мови на частоті 16 кГц, підготовлений Василем Панайотовим за сприяння Деніела Поуві. Дані отримані з прочитаних аудіокниг проекту LibriVox і були ретельно сегментовані та вирівняні [13].

Для навчання буде використуватися саме цей датасет, а саме *train-clear-100.tar.gz* (6.3G) і *train-clear-360.tar.gz* (23G). Як правило чим більше даних на початку тим краще результат буде в кінці.

Для створення фонового шуму буде використаний ESC-50 [14]. Датасет містить 50 класів з 40 прикладами для кожного довжиною у 5 секунд. Загалом всі класи поділено на 5 великих груп: тварини, звуки природи та води, людські звуки, побутові звуки, шум міста.

Для коректної роботи з даними їх потрібно нормалізувати. Необхідність нормалізації вибірок даних обумовлена природою алгоритмів, що використовуються. Вхідні значення ознак можуть змінюватися в дуже великому діапазоні і відрізнятись один від одного на кілька порядків. Наприклад, датасет містить відомості про концентрацію діючої речовини, що вимірюється в десятих або сотих частках відсотків, та показники тиску в сотнях тисяч атмосфер. Або, наприклад, в одному вхідному векторі є інформація про вік і прибуток клієнта. Будучи різними за фізичним змістом, дані сильно різняться між собою за абсолютними величинами. Робота аналітичних моделей машинного навчання з

такими показниками виявиться некоректною: дисбаланс між значеннями ознак може спричинити нестійкість роботи моделі, погіршити результати навчання та сповільнити процес моделювання. Після нормалізації всі числові значення вхідних ознак будуть приведені до однакової сфери їх зміни – це деякого вузького діапазону. Це дозволить звести їх разом в одній моделі машинного навчання та забезпечить коректну роботу обчислювальних алгоритмів.

Для нормалізації даних буде використаний bash скрипт *normalize-resample.sh*.

```
open_sem(){
    mkfifo pipe-$$
    exec 3<>pipe-$$
    rm pipe-$$
    local i=$1
    for((;i>0;i--)); do
        printf %s 000 >&3
    done
}
run_with_lock(){
    local x
    read -u 3 -n 3 x && ((0==x)) || exit $x
    (
     ( "$@"; )
    printf '%.3d' $? >&3
    )&
}

N=32 # set "N" as your CPU core number.
open_sem $N
for f in $(find . -name "*.flac"); do
    run_with_lock ffmpeg-normalize "$f" -ar 16000 -o "${f%.*}-norm.wav"
done
```

Рис. 3.13 Код нормалізації

Архітектура моделі буде використана Open-Unmix-pytorch [15]. В основі моделі лежить трьохшарова двонаправлена LSTM. Модель вчиться передбачати спектрограму цільового джерела, наприклад, вокала, з спектрограми вхідної суміші. Передбачення отримується шляхом накладння маски на вхідний сигнал.

Модель оптимізована в області частоти з використання середньої квадратичної помилки.

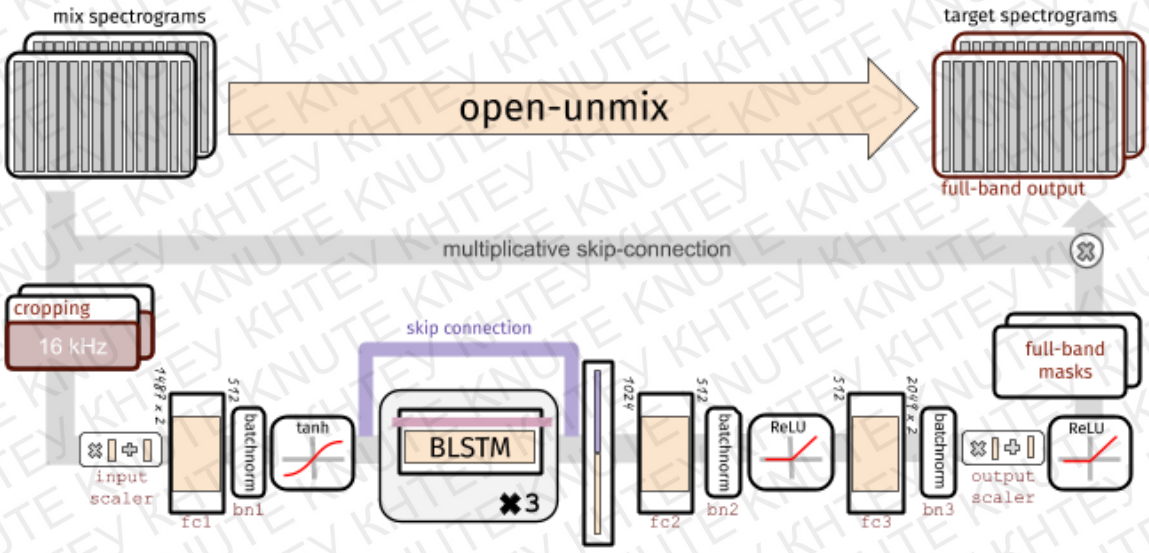


Рис. 3.14 Архітектура Open-Unmix

Модель навчалася 100 епох з використанням NVIDIA GeForce 920MX приблизно 10 годин. Граф функції витрат можна побачити на рис. 3.15. Кінцеве значення навчальної вибірки ~0.17, кінцеве значення вибірки для валідації ~0.22.

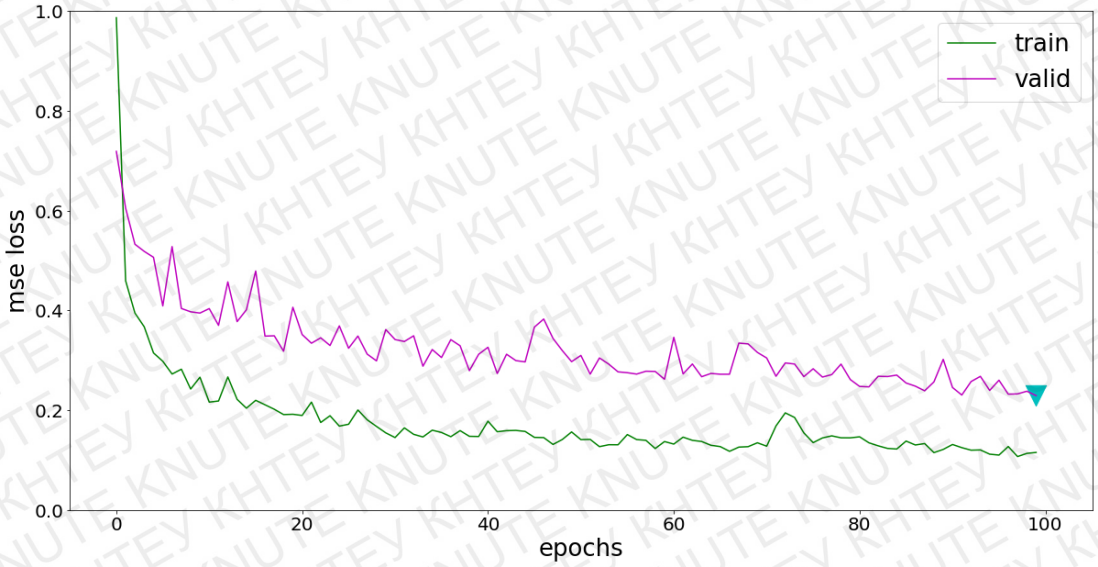


Рис. 3.15 Прогрес під час навчання

Також в результаті отримано діаграму розмаху всіх фонових шумів (рис. 3.16). З графіку видно, що звуки які відбуваються на окремих від навчальної вибірки частотах легше розрізнити. Найпростіші звуки відповідно: цвіркун, клацання мишкою, півень, звуки води та тікання годинника.

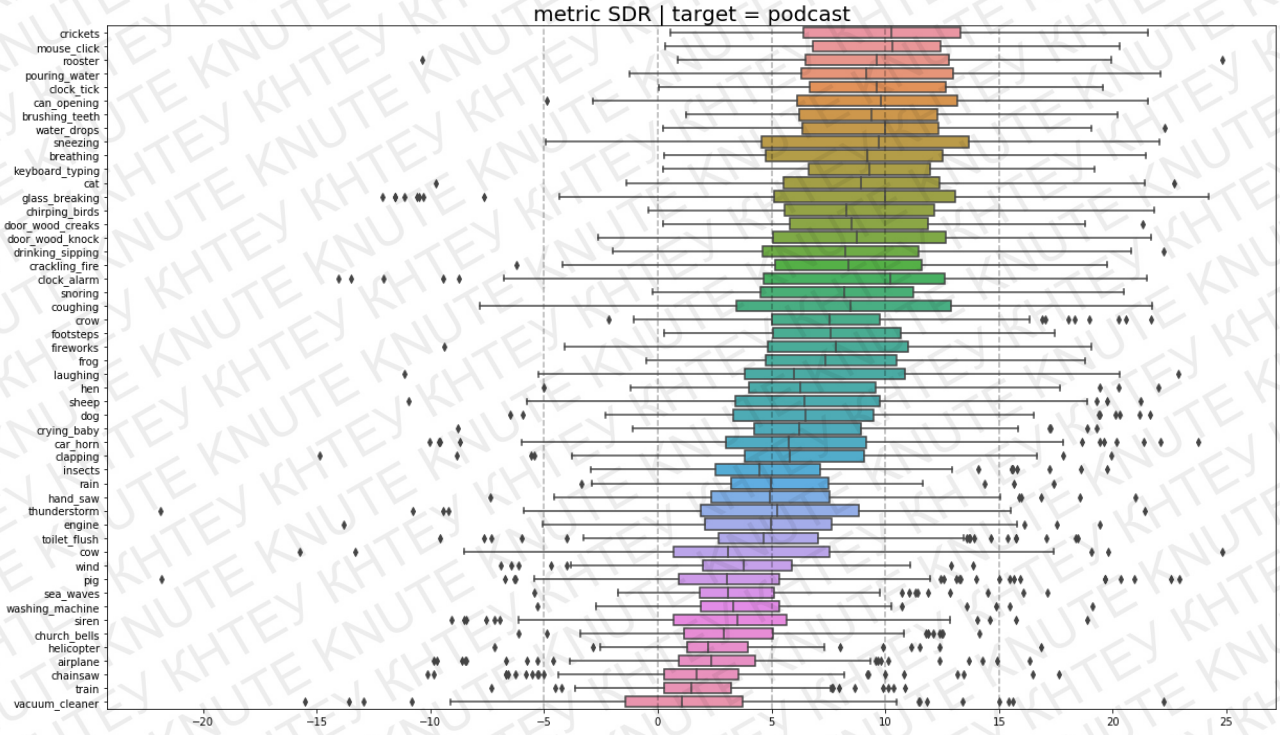


Рис. 3.16 Діаграма розмаху фонових шумів

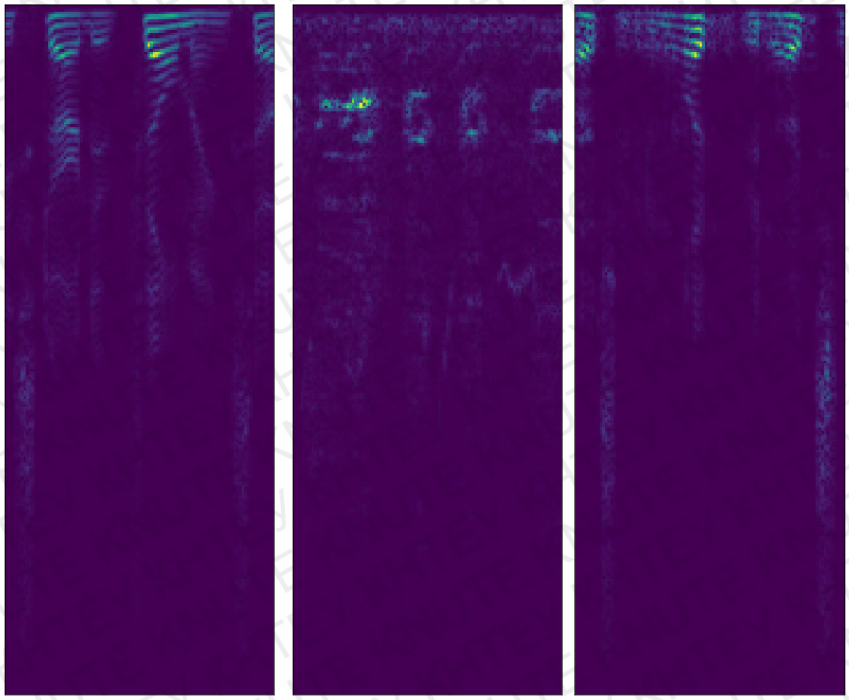


Рис. 3.17 Аудіо приклад. Зліва направо: вхідна аудіодоріжка, сторонній шум, результат на виході

3.4. Висновки до розділу 3

Отже, була розроблена високорівнева бібліотека для побудови та тренування нейронних мереж для глибинного навчання, яка при цьому спроектована так, щоб бути компактною, модульної та розширюваною.

Дана розробка має три ключові переваги:

- Зручність у користуванні.
- Простий, стійкий інтерфейс, оптимізований для випадків загального використання.
- Модульність та композиційність. Моделі створюються шляхом з'єднання настроюваних будівельних блоків (шарів) разом, з невеликими обмеженнями.
- Легко розширюється. Користувач може створювати свої унікальні будівельні блоки: шари, функції, метрики, функції витрат. Зручний інтерфейс накладає мінімум обмежень на функціонал нових систем, класів, структур.

Також було створено модель для фільтрування аудіо від фонових шумів, наприклад гавкання собаки або шум міста. Модель справилися із завдання, що доводить правильність реалізації кожної окремої частини коду.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Основна ціль даної роботи була розробка програмного забезпечення для побудови та навчання нейронних мереж.

Першим кроком при вирішенні завдання було проведено дослідження роботи сучасних нейронних мереж, також проведено ретельний аналіз способів досягнення таких видатних результатів сучасних моделей глибокого навчання в областях комп'ютерного бачення та обробки природної мови. Проаналізовані роботи відомих фреймворків, якими користуються передові компанії такі як Google, Amazon, Facebook, Apple та на основі яких і були створені state-of-the-art моделі; дослідження переваг та недоліків кожного з них для виявлення загальних рис притаманних їм.

Другий крок – це розгляд основних понять лінійної алгебри та принципів на яких побудовані нейронні мережі. Були ретельно розглянуті окремі елементи нейронних мереж: нейрон (вузол), функція активації. функція витрат, оптимізатор; Також, досліджені алгоритми, на яких базуються ці елементи: метод зворотнього поширення помилки (back propagation), спосіб диференціювання складної функції (chain rule), градієнтний спуск (gradient descent).

Третій крок – безпосередня реалізація. Набуті знання були застосовані на практиці для переведення математичних формул в Python код. В процесі роботи кожна окрема формула була приведена до свого матричного варіанту для зручності користування та для оптимації обчислень процесором під час навчання.

Поставлені у вступі цілі й завдання є виконані повністю.

Результати роботи можна використовувати при глибокому навчанні моделей в самих різних областях, наприклад: охорона здоров'я, фінансова, автомобільна індустрія, роздрібна торгівля, урядові установи, транспортна галузь, нафтогазова промисловість.

Щодо можливих напрямів розвитку проекту, тут декілька варіантів. Перший варіант – це вдосконалення уже існуючих модулів в плані додавання якісних змін для прискорення процесу навчання та покращення зв'язку між інженером на цим процесом. Другий варіант – це додавання нових шарів і модулів, наприклад, популярні нині компактні згорткова нейронні мережі (Depthwise separable convolution).

Оскільки система побудована таким чином, що нові будівельні блоки для навчання додавати легко, то для спільної, «безшовної» роботи на них накладається мінімум умов та обмежень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Основний

1. J. Capon, "High-resolution frequency-wavenumber spectrum analysis," in Proceedings of the IEEE, vol. 57, no. 8, pp. 1408-1418, Aug. 1969, doi: 10.1109/PROC.1969.7278.
2. O. L. Frost, "An algorithm for linearly constrained adaptive array processing," in Proceedings of the IEEE, vol. 60, no. 8, pp. 926-935, Aug. 1972, doi: 10.1109/PROC.1972.8817.
3. A. Hyvärinen, E. Oja, Independent component analysis: algorithms and applications, Neural Networks, Volume 13, Issues 4–5, 2000, Pages 411-430, ISSN 0893-6080
4. An end-to-end open source machine learning platform [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/
5. Theano [Електронний ресурс]. – Режим доступу: http://deeplearning.net/software/theano/
6. From research to production [Електронний ресурс]. – Режим доступу: https://pytorch.org/
7. scikit-learn Machine Learning in Python [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/
8. Keras. Simple. Flexible. Powerful. [Електронний ресурс]. – Режим доступу: https://keras.io/
9. Hornik, Kurt; Tinchcombe, Maxwell; White, Halbert (1989). Multilayer Feedforward Networks are Universal Approximators (PDF). Neural Networks.

10. Chain rule [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Chain_rule
11. WSJ0 [Електронний ресурс]. – Режим доступу: https://catalog.ldc.upenn.edu/LDC93S6A
12. Open Speech and Language Resources [Електронний ресурс]. – Режим доступу: http://www.openslr.org/index.html
13. LibriSpeech ASR corpus [Електронний ресурс]. – Режим доступу: http://www.openslr.org/12/
14. ESC-50: Dataset for Environmental Sound Classification [Електронний ресурс]. – Режим доступу: https://github.com/karolpiczak/ESC-50
15. Open-Unmix for PyTorch [Електронний ресурс]. – Режим доступу: https://github.com/sigsep/open-unmix-pytorch
16. Андрійчук В. І. Лінійна алгебра : навч. посіб. / В. І. Андрійчук, Б. В. Забавський; Львів. нац. ун-т ім. І.Франка. - Л., 2008. - 238 с. - Бібліогр.: с. 228-229. - укр.
17. van Rossum, G., Drake, F. L. (2011). *The Python Language Reference Manual*. Network Theory Ltd.
18. "Python : the holy grail of programming". *CERN Bulletin*. CERN Publications (31/2006). 31 July 2006. Retrieved 11 February 2012.
19. Deily, Ned (25 March 2019). Python 3.7.3 is now available. *Python Insider*. The Python Core Developers.
20. Chollet, F. (2017). *Deep Learning with Python*. Manning, 2017. — 384 p.
21. Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015), S. Loffe and C. Szegedy

22. G. Cybenko Approximation by Superpositions of a Sigmoidal Function, Math. Control Signals Systems (1989) 2:303-314

23. Hochreiter, Sepp[en]; and Schmidhuber, Jürgen[en]; Long Short-Term Memory, Neural Computation, 9(8):1735–1780, 1997

24. Run and automate deep learning experiments faster [Електронний ресурс]. – Режим доступу: https://missinglink.ai/

ДОДАТКИ

Додаток А

```python
import numpy as np
from scipy import signal

from layer import Layer


class ConvLayer(Layer):
    # input_shape = (i,j,d)
    # kernel_shape = (m,n)
    # layer_depth = output_depth
    def __init__(self, input_shape, kernel_shape, layer_depth):
        self.input_shape = input_shape
        self.input_depth = input_shape[2]
        self.kernel_shape = kernel_shape
        self.layer_depth = layer_depth
        self.output_shape = (input_shape[0] - kernel_shape[0] + 1, input_shape[1] - kernel_shape[1] + 1, layer_depth)
        self.weights = np.random.rand(kernel_shape[0], kernel_shape[1], self.input_depth, layer_depth) - 0.5
        self.bias = np.random.rand(layer_depth) - 0.5

    # returns output for a given input
    def forward_propagation(self, input):
        self.input = input
        self.output = np.zeros(self.output_shape)

        for k in range(self.layer_depth):
            for d in range(self.input_depth):
                self.output[:, :, k] += signal.correlate2d(self.input[:, :, d], self.weights[:, :, d, k], 'valid') + \
                                        self.bias[k]

        return self.output

    # computes dE/dW, dE/dB for a given output_error=dE/dY. Returns input_error=dE/dX.
    def backward_propagation(self, output_error, learning_rate):
        in_error = np.zeros(self.input_shape)
        dWeights = np.zeros((self.kernel_shape[0], self.kernel_shape[1], self.input_depth, self.layer_depth))
        dBias = np.zeros(self.layer_depth)

        for k in range(self.layer_depth):
            for d in range(self.input_depth):
                in_error[:, :, d] += signal.convolve2d(output_error[:, :, k], self.weights[:, :, d, k], 'full')
                dWeights[:, :, d, k] = signal.correlate2d(self.input[:, :, d], output_error[:, :, k], 'valid')
            dBias[k] = self.layer_depth * np.sum(output_error[:, :, k])

        self.weights -= learning_rate * dWeights
        self.bias -= learning_rate * dBias
        return in_error
```

Рис. А.1. *conv_layer.py*

```python
class Network:
    def __init__(self):
        self.layers = []
        self.loss = None
        self.loss_prime = None

    # add layer to network
    def add(self, layer):
        self.layers.append(layer)

    # set loss to use
    def use(self, loss, loss_prime):
        self.loss = loss
        self.loss_prime = loss_prime

    # predict output for given input
    def predict(self, input_data):
        # sample dimension first
        samples = len(input_data)
        result = []

        # run network over all samples
        for i in range(samples):
            # forward propagation
            output = input_data[i]
            for layer in self.layers:
                output = layer.forward_propagation(output)
            result.append(output)

        return result

    # train the network
    def fit(self, x_train, y_train, epochs, learning_rate):
        # sample dimension first
        samples = len(x_train)

        # training loop
        for i in range(epochs):
            err = 0
            for j in range(samples):
                # forward propagation
                output = x_train[j]
                for layer in self.layers:
                    output = layer.forward_propagation(output)

                # compute loss (for display purpose only)
                err += self.loss(y_train[j], output)

                # backward propagation
                error = self.loss_prime(y_train[j], output)
                for layer in reversed(self.layers):
                    error = layer.backward_propagation(error, learning_rate)

            # calculate average error on all samples
            err /= samples
            print('epoch %d/%d   error=%f' % (i + 1, epochs, err))
```

Рис. А.2. *network.py*