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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка програмного забезпечення нейронної мережі прямого поширення»

Студента 4 курсу, 6 групи, спеціальності 121 «Інженерія програмного забезпечення»
____
підпис студента
Маслова Артема Івановича

Науковий керівник кандидат технічних наук професор
____
підпис керівника
Пашорін Валерій Іванович

Гарант освітньої програми кандидат технічних наук, доцент
____
підпис керівника
Цензура Микола Олександрович

КИЇВ – 2020

АНОТАЦІЯ

Дипломна робота на тему "Розробка програмного забезпечення нейронної мережі прямого поширення" містить 56 сторінок, 27 рисунків та 1 додаток. Список посилань включає 18 пунктів.

Мета – аналіз і розробка методу створення та навчання глибоких мереж прямого поширення.

Об’єкт – нейронні мережі прямого поширення.

Предмет – розробка методу навчання нейронної мережі прямого поширення.

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; розробити бібліотеку навчання нейронної мережі прямого поширення.

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 "Development of a feedforward neural network software" contains 56 pages, 27 drawings, and 1 appendix. The list of links includes 18 items.

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

The research subject is the feedforward neural network.

The subject of study is the development of a training method for a feedforward neural network.

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; develop a unique deep learning library.

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.

Київський національний торговельно-економічний університет

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

Затверджую
Зав. кафедри інженерії
програмного забезпечення
та кібербезпеки
Криворучко О. В.
"7" листопада 2019 р.

Завдання
на випускну кваліфікаційну роботу студентові

Маслову Артему Івановичу

1. Тема випускної кваліфікаційної роботи: *«Розробка програмного забезпечення нейронної мережі прямого поширення».*

Затверджена наказом ректора від *«02» грудня 2019 р. № 4112*

2. Строк здачі студентом закінченої роботи

3. Цільова установка та вихідні дані до роботи

Мета роботи: *аналіз і розробка методу створення та навчання глибоких мереж прямого поширення*

Об'єкт дослідження: *нейронні мережі прямого поширення*

Предмет дослідження: *розробка методу навчання нейронної мережі прямого поширення*

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 ФУНКЦІОНУВАННЯ НЕЙРОННОЇ МЕРЕЖІ

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 до мережі в цілому

1.3.2. Вимоги до структури

1.3.3. Вимоги пов'язані з технічними засобами та програмним забезпеченням:

1.4. Вимоги до функцій (задач), що виконуються системою:

1.4.1. Layer (шар)

1.4.2. Fully Connected Layer

1.4.3. Activation layer

1.4.4. Activations

1.4.5 Losses

1.4.6 Network

1.5. Огляд готових рішень

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*		
2.	*Вступ та перелік літературних джерел*		
3.	*Розділ 1. «Загальні відомості та технічне завдання»*		
4.	*Розділ 2. «Аналіз предмета дослідження та визначення основних компонентів системи»*		
5.	*Розділ 3. «Імплементація нейронної мережі»*		
6.	*Висновки*		
7.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*		
8.	*Підготовка автореферату та презентації доповіді*		
9.	*Попередній захист випускної кваліфікаційної роботи*		
10.	*Зовнішнє рецензування випускної кваліфікаційної роботи*		
11.	*Здача прошитої випускної кваліфікаційної роботи на кафедру*		
12.	*Публічний захист випускної кваліфікаційної роботи*		

7. Дата видачі завдання «___»_______________20__ р.

8. Науковий керівник випускної кваліфікаційної роботи Пашорін В. І.

9. Гарант освітньої програми Цензура М. О.

10. Завдання прийняв до виконання студент Маслов А. І.

11. Відгук керівника випускної кваліфікаційної роботи

____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____

Науковий керівник випускної кваліфікаційної роботи ____________________

(підпис, дата)

Відмітка про попередній захист Пашорін В. І. ________________

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну робота

Випускна кваліфікаційна робота студента Маслова А. І. може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Цензура М. О.

Завідувач кафедри Криворучко О. В.

«_____»_________________20 ___ р.

ЗМІСТ

ВСТУП

Що таке глибоке навчання і чому це важливо?

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

Машинне навчання, глибоке 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.

Глибоке навчання – це підмножина машинного навчання, де штучні нейронні мережі, алгоритми, схожі за своєю подобою до людського мозку, вчаться на великій кількості даних. Глибоке 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 глибоке навчання, що дозволяє автомобілям розпізнавати інші транспортні засоби, знаки зупинки та навіть пішоходів. Глибоке 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 глибокого 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 глибокого навчання робить 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.

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

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

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

В галузі медицини, дослідники використовують глибоке навчання для виявлення ракових клітин. Навіть промислові компанії використовують глибоке навчання, щоб покращити життя працівників, визначаючи, коли працівники ризикують поранити себе під час експлуатації важкої техніки.

Глибокі засоби навчання продовжуватимуть змінювати спосіб життя людей, роботи людей, створювати та навіть розробляти продукти.

Метою дослідження випускної кваліфікаційної роботи є аналіз і розробка методу створення та навчання глибоких мереж прямого поширення.

Об'єктом дослідження є нейронні мережі прямого поширення.

Предметом – розробка методу навчання нейронної мережі прямого поширення.

Завдання дослідження:

- Розглянути основні принципи побудови сучасних нейронних мереж.

- Розглянути та проаналізувати структуру готових рішень для глибокого навчання.
- Засвоїти основні поняття та визначення з лінійної алгебри та усвідомити їхнє геометричне представлення;
- На основі даних аналізу, розробити бібліотеку навчання нейронної мережі прямого поширення.

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 ФУНКЦІОНУВАННЯ НЕЙРОННОЇ МЕРЕЖІ

1.1. Загальні відомості

1.1.1. Найменування системи

Повне найменування: бібліотека для розробки нейронних мереж.

Скорочене найменування системи: бібліотека, API, фреймворк.

1.1.2. Планові терміни початку та закінчення робіт

Початок роботи: жовтень 2019 р.

Закінчення роботи: квітень 2020 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 зі створення нейронної мережі прямого поширення виконуються 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 машиного навчання, інженери, вчені різних галузей, бізнес-аналітики.

1.2. Мета та призначення створення системи

1.2.1. Призначення системи

Система призначена для швидкого проєктування нейронних мереж, для легкого експериментування та прототипування.

1.2.2. Мета створення системи

Метою даної роботи є розробка адаптивного методу навчання 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 до мережі в цілому

Бібліотека має представляти з себе високорівневий API нейронних мереж. Вона має бути розроблена з акцентом на експериментування та прототипування; спроєктована для уможливлення швидких експериментів з мережами глибинного навчання; зосереджено на тому, щоби вона була зручною в користуванні, модульною та розширюваною.

Знаючи вимоги, можна виокремити 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 мереж прямого поширення.

1.3.3. Вимоги пов’язані з технічними засобами та програмним забезпеченням:

- Мати повністю відкритий код
- мінімізувати зовнішні залежності

- не використовувати програмне забезпечення сторонніх розробників

Як наслідок дану бібліотеку можна розповсюджувати з Ліцензією MIT.

Ліцензія MIT – це дозвільна ліцензія на вільне програмне забезпечення, що походить з Массачусетського технологічного інституту (MIT) наприкінці 1980-х років. Як дозвільна ліцензія, вона ставить мінімальні обмеження щодо повторного використання і, отже, має високу сумісність ліцензій [13].

Цей перелік якостей є необхідним для будь-якого проєкта в open-source розробці.

Таким чином його можна викласти з MIT ліцензією у відкритий доступ в систему контролю версій, де інші розробники зможуть вносити свій вклад в розробку.

Використовувати цю бібліотеку необхідно, якщо потрібна бібліотека глибокого навчання, яка:

- Дозволяє легко та швидко прототипувати (завдяки зручності користування, модульності та розширюваності).
- Підтримує як згорткові мережі, так і рекурентні мережі, а також їх комбінації.

1.4. Вимоги до функцій (задач), що виконуються системою:

Основна задача запропонованої системи – це можливість створювати і навчати мережі прямого поширення. Звісно для того щоб система була розширювана і така в якій легко розібратися і доповнити сторонньому розробнику, потрібно створити її модульною, використовуючи основні принципи ООП.

1.4.1. Layer (шар)

Клас Layer буде реалізувати основний компонент системи – шар. 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 батьківским класом - інтерфейсом для всіх реалізацій будь-яких шарів.

1.4.2. Fully Connected Layer

Основний шар в даному проекті. 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.

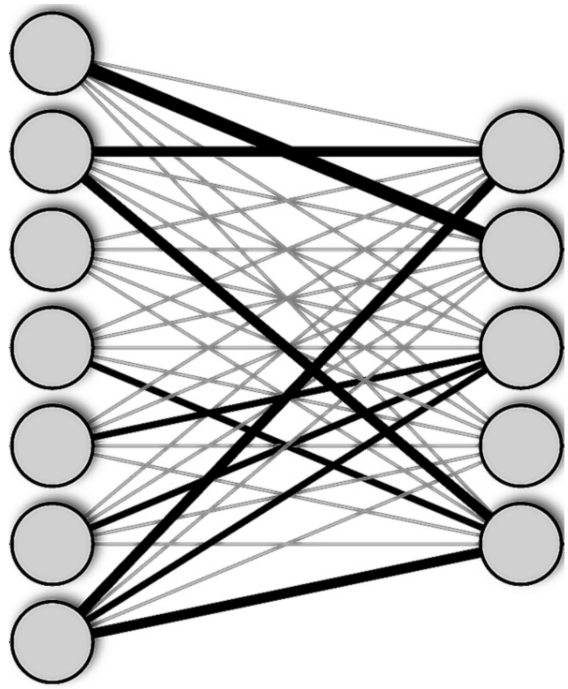

Рис. 1.1 Fully connected layer

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

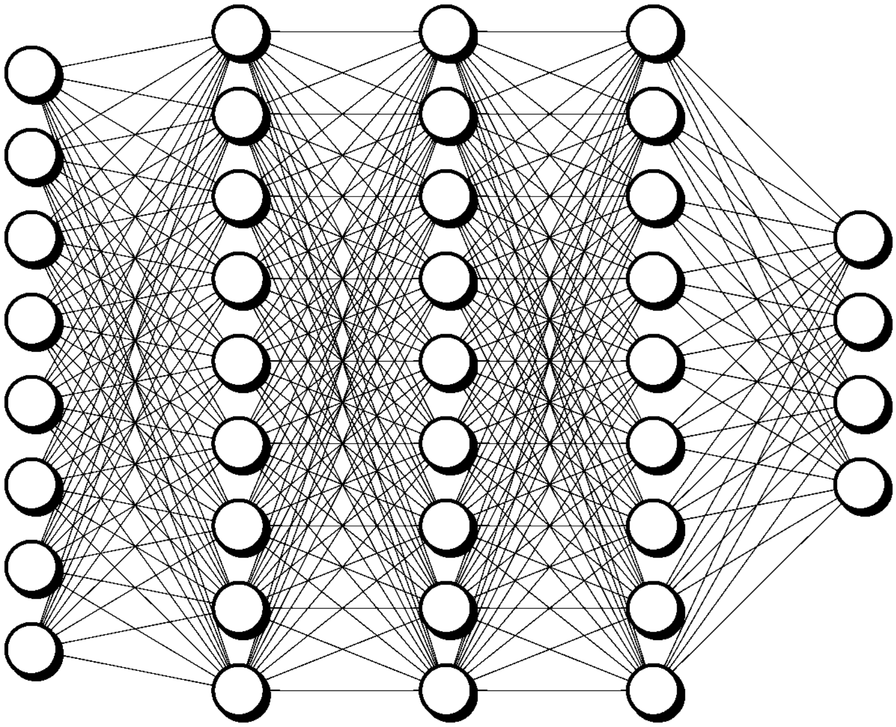

Рис. 1.2. Приклад кінцевої мережі

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.

Вимоги аналогічні що й у повністю поєднаному шарі: реалізація алгоритму прямого поширення і зворотнього.

1.4.4. Activations

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

1.4.5 Losses

В цьому модулі будуть описані функції витрат. Прикладами можуть служити: середнє квадратичне відхилення (mean square error) або абсолютне середнє відхилення (mean absolute error).

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

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

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.

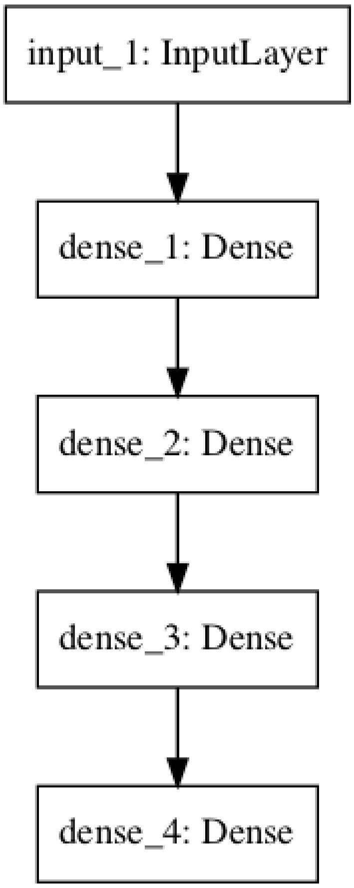

Рис. 1.3. Схематичний приклад об'єднання шарів у мережу

1.5. Огляд готових 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 [14].

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 [15].

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 [16].

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 [17].

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 [18].

Переваги:

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

Недоліки:

- Це лише надстройка над іншими бібліотеками
- Недостатня документація

1.6. Вимоги до розробки сучасних нейронних мереж

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

- Зручність у користуванні. Проект пропонує послідовні та прості високорівневі API, що мінімізує 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 нейронних мереж прямого поширення для задач які вирішує навчання з вчителем, а саме: класифікації та регресії.

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 кластеризувати та класифікувати. 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 основні поняття нейронних мереж, як вони працюють, яка у них структура, як вони навчаються. Всі ці теоретичні знання є необхідними для написання своєї нейронної мережі.

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 $f\colon \mathrm{A} \to \mathrm{B}$ (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 — 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 коефцієнт «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 y, 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, пішоходні 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 Елементи нейронної мережі

2.2.1. Нейрон

Структурною одиницею нейронної мережі є нейрон або вузол (node).

Вузол – це просто місце, де відбувається обчислення, що віддалено нагадує нейрон в людському мозку, який спрацьовує, стикаючись із достатньою кількістю подразників. Вузол поєднує вхідні дані з набором коефіцієнтів або ваг, які або підсилюють, або зменшують цей вхід, надаючи таким чином важливості вхідним даних стосовно завдання, яке алгоритм намагається вивчити; наприклад. який вхід є найбільш корисним для класифікації даних без помилок? Ці скалярні добутки підсумовуються, потім сума передається через функцію активації вузла, щоб визначити, чи повинен і в якій мірі цей сигнал просуватися далі по мережі, щоб вплинути на кінцевий результат, скажімо, акт класифікації. Якщо сигнали проходять, то нейрон був «активований», приклад нейрона показано на рис. 2.1:

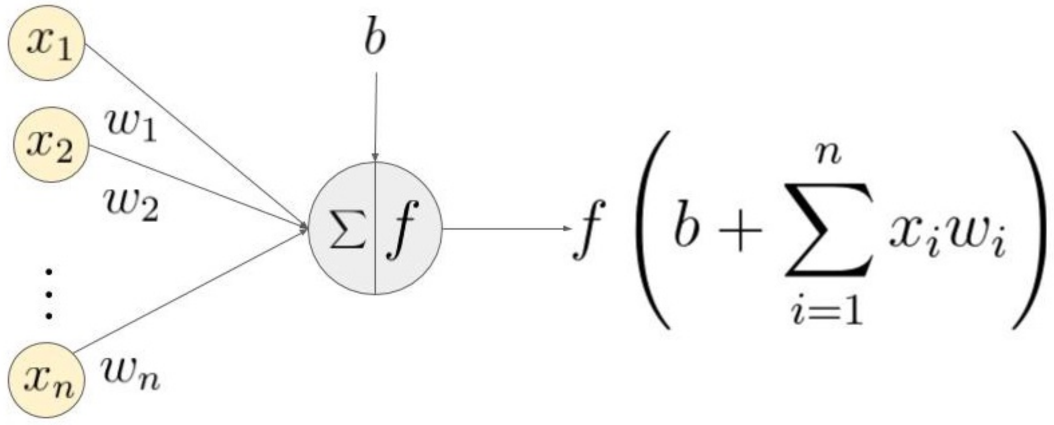

Рис. 2.1. Нейрон

На вхід нейрона подається вектор розміру n області визначення $x = (x^1, \ldots, x^n)$. Кожній одиниці вхідних даних відповідає ваговий коефіцієнт – $w_1, \ldots, w_n$, який присвоюється на основі його важливості відносно інших вхідних даних. Вихідним значенням нейрона є значення функції активації від зваженої суми входів $f(w_1x^1 + w_2x^2 + \ w_nx^n + b)$.

Даний вид нейрона є найпростішим лінійним класифікатором і може самостійно навчатися. Якщо нейрон знаходиться в прихованому або вихідному шарі нейронної мережі, то на вхід йому подаються вихідні значення інших нейронів.

2.2.2. Функції активації

Функції активації – це математичні рівняння, що визначають вихідні значення нейронної мережі. Одна із таких функцій приєднана до кожного нейрона в мережі та визначає, чи слід його активувати ("запустити") чи ні, виходячи з того, чи є вхідне значення кожного нейрона необхідним для прогнозування моделі. Функції активації також допомагають нормалізувати вихід кожного нейрона в діапазон між 1 і 0 або між -1 і 1.

Додатковим аспектом функцій активації є те, що вони повинні бути обчислювально ефективними, оскільки вони обчислюються тисячами або навіть мільйонами нейронів для кожного одиниці даних. Сучасні нейронні мережі використовують тренінг моделі, що називається методом зворотного поширення помилки, яка покладає на функцію активації та її похідну функцію підвищену обчислювальну напругу.

Нелінійні функції активації

Сучасні моделі нейронної мережі використовують нелінійні функції активації. Вони дозволяють моделі створювати складні відображення між входами та виходами мережі, які мають важливе значення для вивчення та моделювання складних даних, таких як зображення, відео, аудіо та набори даних, які нелінійні або мають високу розмірність.

Практично будь-який процес, який можна уявити, може бути представлений як функціональне обчислення в нейронній мережі за умови, що функція активації нелінійна.

Нелінійні функції вирішують проблеми лінійної функції активації:

- Вони дозволяють зворотне розповсюдження, оскільки вони мають похідну функцію, яка пов'язана з входами.
- Вони дозволяють «укладати» декілька шарів нейронів для створення глибокої нейронної мережі. Для вивчення складних наборів даних з високим рівнем точності потрібно кілька прихованих шарів нейронів.

6 Поширених нелінійних функцій активації

Сигмоїда

Переваги

- Плавний градієнт, що запобігає «стрибкам» у вихідних значеннях.
- Вихідні значення лежать в проміжку $[0, 1]$, нормалізуючи вихід кожного нейрона.

Недоліки

- Зміна градієнта - для дуже високих або дуже низьких значень X майже не змінюється прогнозування, що спричиняє проблему зникання градієнту. Це може призвести до того, що мережа перестане навчатися чи буде робити це занадто повільно, щоб досягти точного прогнозу.
- Обчислювально дорога

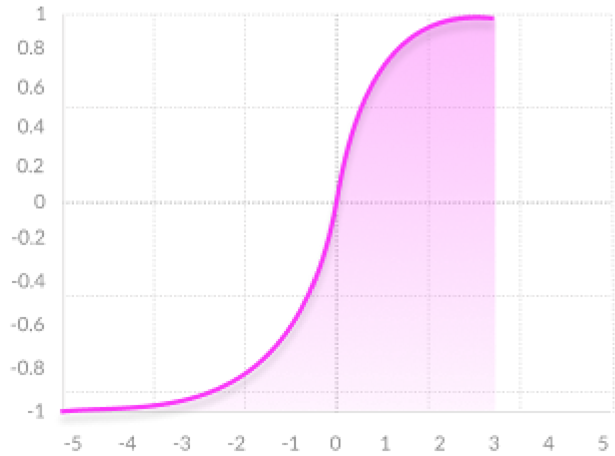

Рис. 2.2. Сигмоїда

TanH / гіперболічний тангенс

Переваги

- Зосереджена біля 0 - полегшує моделювання входів із сильно негативними, нейтральними та сильно позитивними значеннями.
- Все інше аналогічне сигмоїдальній функції.

Недоліки

- Такі самі як у сигмоїдальній функції.

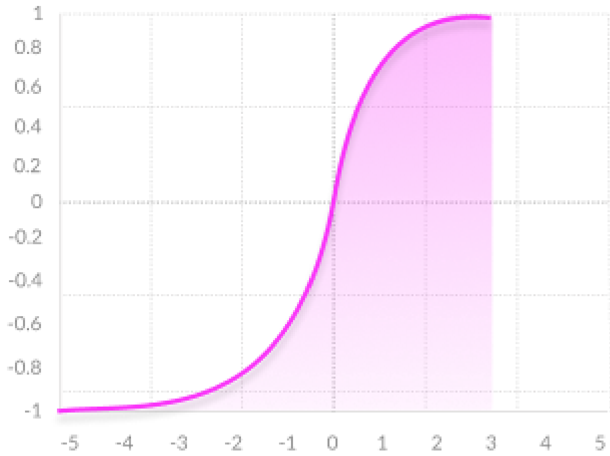

Рис. 2.3. Гіперболічний тангенс

ReLU (Rectified Linear Unit)

Переваги

- Обчислювально ефективний – дозволяє мережі швидше зійтися.
- Нелінійна – хоча це виглядає як лінійна функція, ReLU має похідну функцію і дозволяє здійснювати зворотне поширення помилки.

Недоліки

- Проблема Dying ReLU – коли входи наближаються до нуля або є від’ємними, градієнт функції стає нульовим, мережа не може виконувати зворотне поширення помилки і не може вчитися.

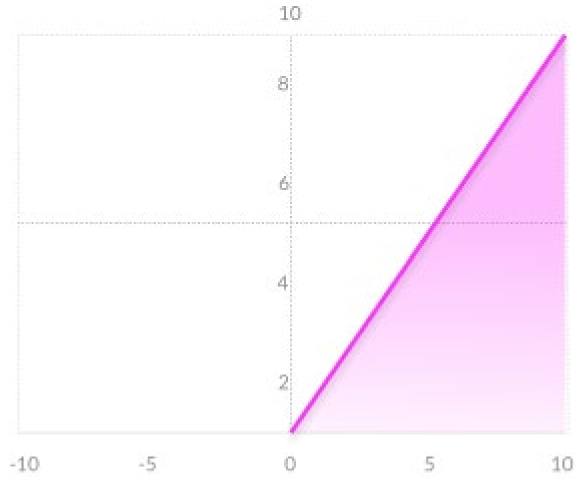

Рис. 2.4. ReLU

Нещільна ReLU

Переваги

- Запобігає виникненню проблеми з ReLU – цей варіант ReLU має невеликий позитивний нахил у негативній області, тому це дозволяє включити зворотне поширення навіть для негативних вхідних значень
- Все інакше як ReLU

Недоліки

- Результати не постійні – нещільний ReLU не дає послідовних прогнозів для негативних вхідних значень.

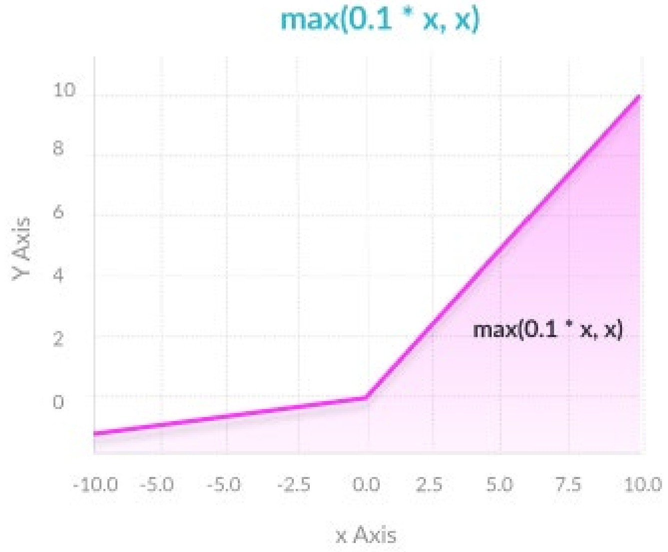

Рис. 2.5. ReLU

Softmax (нормована експоненційна функція)

Переваги

- Здатний обробляти кілька класів - нормалізує результати для кожного класу від 0 до 1 і ділиться їх на їхню суму, надаючи ймовірність того, що вхідне значення знаходиться в конкретному класі.
- Корисні для вихідних нейронів - зазвичай Softmax використовується лише для вихідного шару, для нейронних мереж, яким потрібно класифікувати входи на кілька категорій.

$$\sigma(z)j = \frac{ezj}{\sum_{k=1ezk}^{k}} \quad for\ j = 1....k.$$

Рис. 2.6. Softmax

Swish

Swish – це нова функція активації, відкрита дослідниками Google. Згідно з їх роботою, вона працює краще, ніж ReLU з аналогічним рівнем обчислювальної ефективності. В експериментах на ImageNet з однаковими моделями, але одна з ReLU, а інша з Swish, нова функція досягла точності класифікації на 0,6-0,9 % більше.

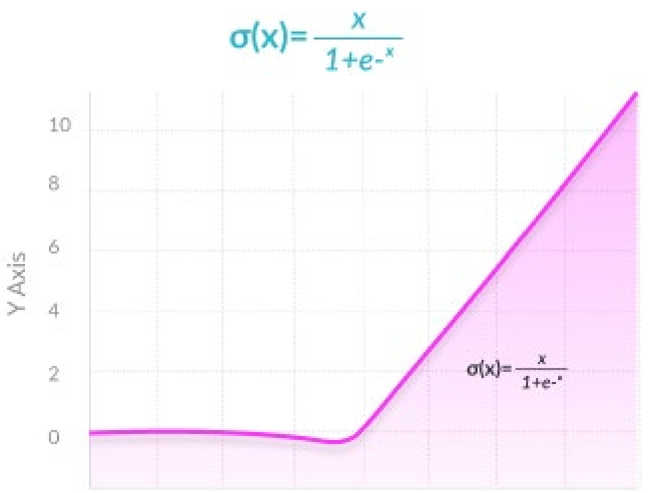

Рис. 2.7. Swish

2.3. Нейронна мережа

З'єднувати нейрони в мережу можна багатьма способами. Найбільш поширений і досліджений спосіб композиції нейронів – багатошаровий персептрон. Нейрони розташовуються шарами, при цьому кожен нейрон з наступного шару пов'язаний з усіма нейронами попереднього. Виділяють три типи шару – вхідний шар нейронів, прихований шар і вихідний шар. Найчастіше використовується тришаровий персептрон, тобто з одним прихованим шаром.

Вихідні значення алгоритму є виходами останнього шару нейронної мережі. Кількість нейронів у вихідному шарі відповідає розмірності вектору відповідей. Кількість нейронів у вхідному шарі відповідає розмірності простору ознак опису об'єкта. Кількість нейронів у прихованих шарах вибирається експертом чи за допомогою евристик щодо оптимізації структури мережі.

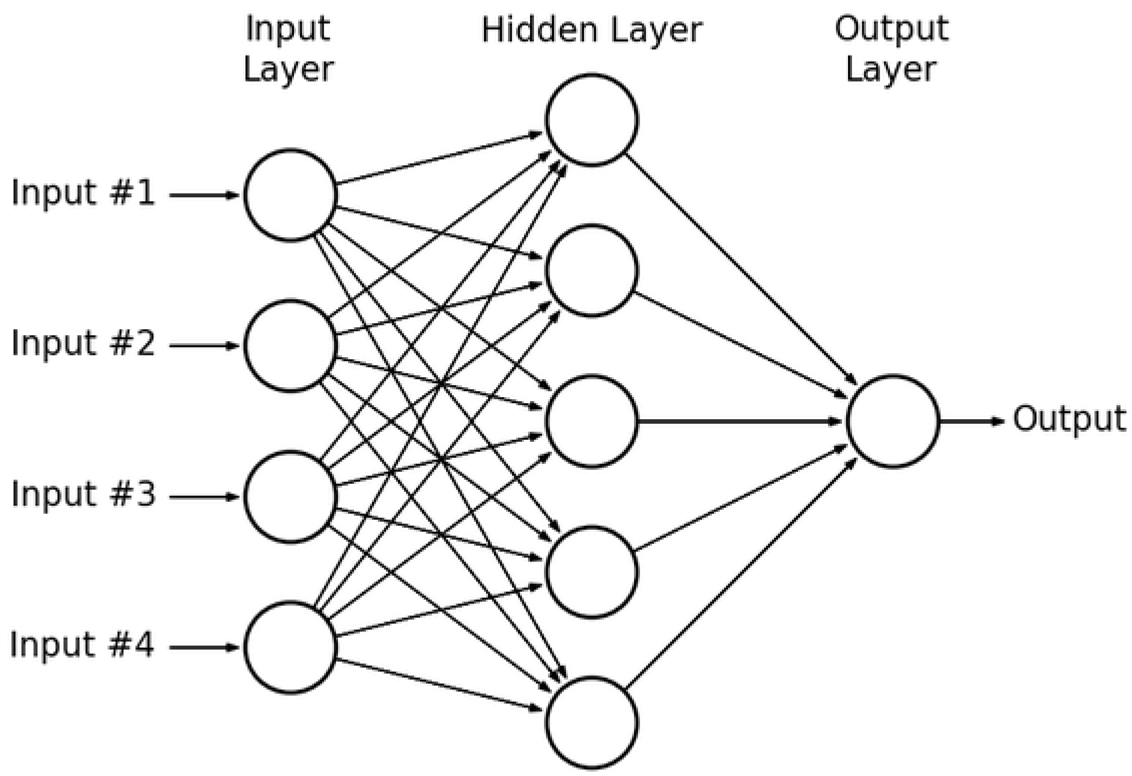

Рис. 2.8. Структура нейронної мережі

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.

$$Q(w) = \frac{1}{n}\sum_{i=1}^{l} L(a(x_i), y_i) \rightarrow \min \qquad (2.1)$$

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.

$$L\,(a\,(x_i), y_i)\; = \;(a(x_i) - y_i)^2 \qquad (2.2)$$

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

Back propagation

Яким же чином ваги нейронної мережі оновлюються? Цей процес називається – метод зворотнього поширення помилки або **back propagation (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.

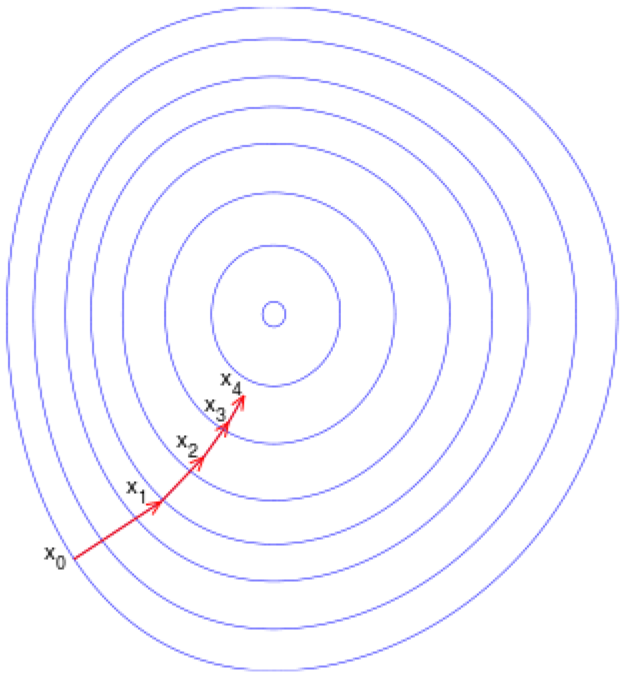

Рис. 2.9. Ілюстрація методу найшвидшого спуску на ряді множин рівнів.

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 ваг, 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**, 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

$$\frac{dz}{dx} = \frac{dz}{dy} * \frac{dy}{dx} \tag{2.3}$$

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

$$\frac{d\text{Помилка}}{d\text{вага}} = \frac{d\text{Помилка}}{d\text{активація}} * \frac{d\text{активація}}{d\text{вага}} \tag{2.4}$$

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 навчання в глибокому навчанні - це не що інше, як коригування ваги моделі у відповідь на помилку, яку вона створює, допоки помилку більше зменшити не можна.

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, мережа прямого поширення.

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 (back propagation), 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.

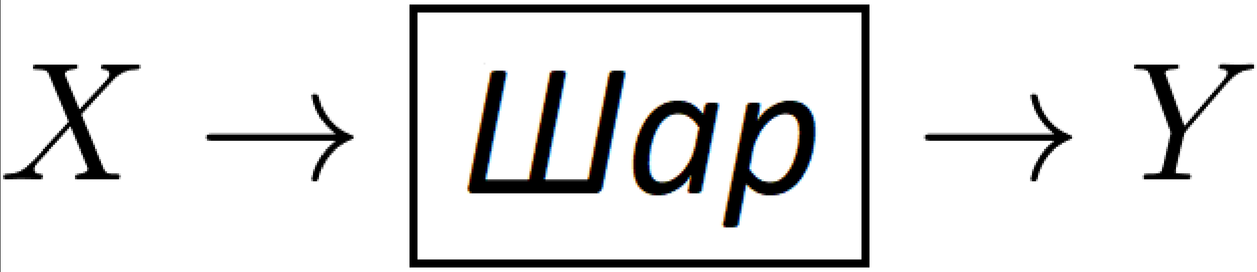

Рис. 3.1. Схематичний інтерфейс будь-якого якого шару при прямому поширенні

Пряме поширення

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

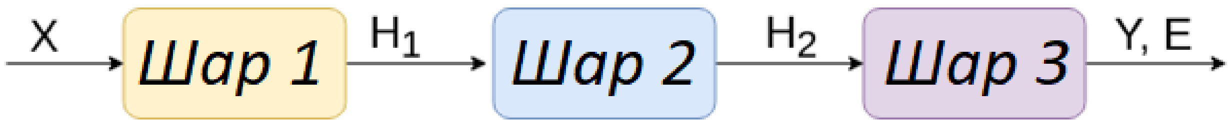

Рис. 3.2. Залежність між шарами

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 ($\partial \mathrm{E}/_{\partial X}$).

$$\frac{\partial E}{\partial X} \leftarrow \boxed{Шар} \leftarrow \frac{\partial E}{\partial Y}$$

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:

$$\frac{\partial E}{\partial x_i} = \sum_j \frac{\partial E}{\partial y_i} \frac{\partial y_j}{\partial w} \qquad (3.2)$$

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.

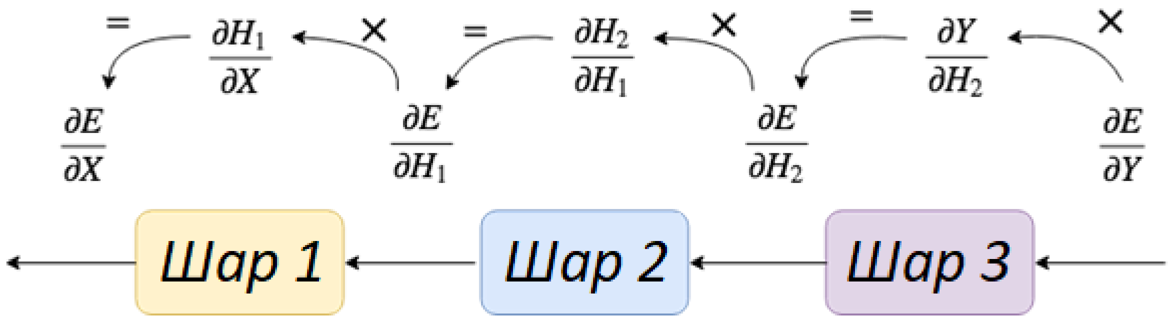

Рис. 3.4. Передача між шарами кількісного значення функції витрат

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 окремих класів

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
# Base class
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 вагів. 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. Повнозв’язний шар

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.

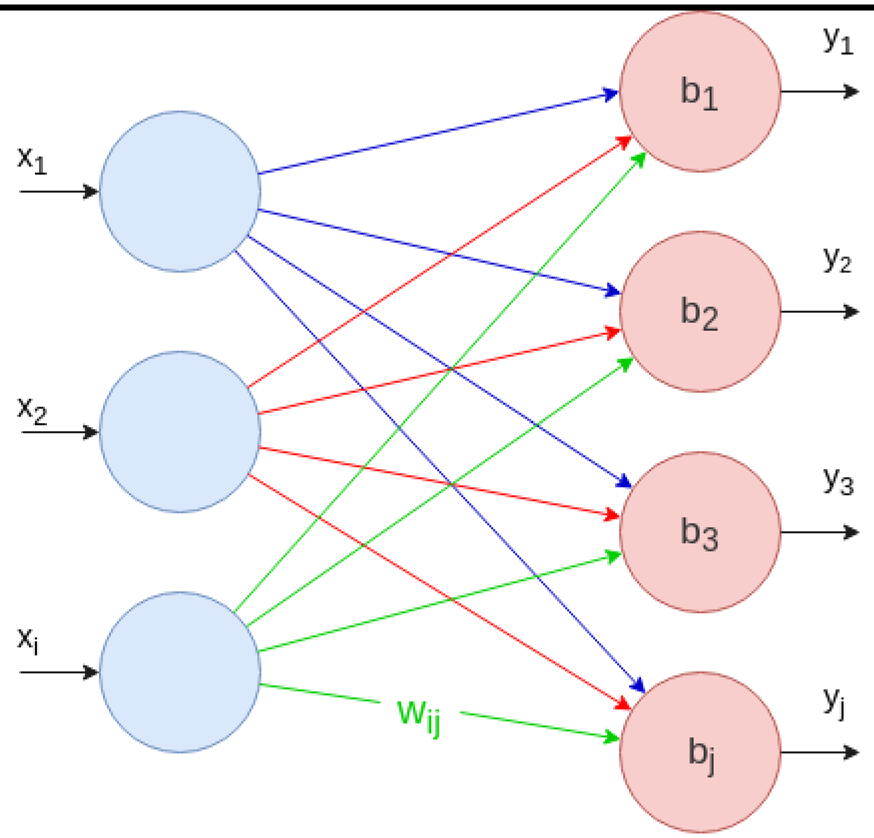

Рис. 3.6. FC layer

Forward Propagation

Значення кожного нейрону обраховується за формулою:

$$y_j = b_j + \sum_i x_i w_{ij} \tag{3.3}$$

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

$$\mathrm{Y} = XW + B \tag{3.4}$$

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 $^{\partial \mathrm{E}}/_{\partial w}$:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_1}{\partial w_{ij}} + \cdots + \frac{\partial E}{\partial y_i}\frac{\partial y_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}x_i \tag{3.5}$$

Векторизована формула:

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial y_1}x_1 & \dots & \frac{\partial E}{\partial y_i}x_1 \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial y_1}x_i & \dots & \frac{\partial E}{\partial y_i}x_i \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_i \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial y_1} & \dots & \frac{\partial E}{\partial y_j} \end{bmatrix} = X^t \frac{\partial E}{\partial Y} \tag{3.6}$$

Вирахуємо тепер формулу для єдиного значення $^{\partial \mathrm{E}}/_{\partial B}$.

$$\frac{\partial E}{\partial B} = \begin{bmatrix} \frac{\partial E}{\partial b_1} & \dots & \frac{\partial E}{\partial b_j} \end{bmatrix} \tag{3.7}$$

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial b_j} + \cdots + \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial b_j} = \frac{\partial E}{\partial y_j} \tag{3.8}$$

Векторизований варіант для $^{\partial \mathrm{E}}/_{\partial B}$:

$$\frac{\partial E}{\partial B} = \begin{bmatrix} \frac{\partial E}{\partial y_1} & \dots & \frac{\partial E}{\partial y_j} \end{bmatrix} = \frac{\partial E}{\partial Y} \quad (3.9)$$

Нарешті, останній пункт це порахувати $\partial \mathrm{E} / \partial X$, який буде служити як $\partial \mathrm{E} / \partial Y$ шару перед поточним.

$$\frac{\partial E}{\partial X} = \begin{bmatrix} \frac{\partial E}{\partial x_1} & \frac{\partial E}{\partial x_2} \dots & \frac{\partial E}{\partial x_i} \end{bmatrix} \quad (3.10)$$

Використовуючи ланцюгове правило:

$$\frac{\partial E}{\partial x_i} = \begin{bmatrix} \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_i} & + \dots + & \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i} \end{bmatrix} = \frac{\partial E}{\partial y_i} w_{i1} + \quad \dots \quad + \frac{\partial E}{\partial y_i} w_{ij} \quad (3.11)$$

Формула для матриць:

$$\frac{\partial E}{\partial X} = \begin{bmatrix} (\frac{\partial E}{\partial y_1} w_{11} + \quad \dots \quad + \frac{\partial E}{\partial y_{ij}} w_{1j}) & \dots & (\frac{\partial E}{\partial y_1} w_{ij} + \quad \dots \quad + \frac{\partial E}{\partial y_{ij}} w_{ij}) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial E}{\partial y_1} & \dots & \frac{\partial E}{\partial y_j} \end{bmatrix} \begin{bmatrix} w_{11} & \dots & w_{i1} \\ \vdots & \ddots & \vdots \\ w_{1j} & \dots & w_{ij} \end{bmatrix} \quad (3.12)$$

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

Отже, три формули необхідні для написання FC шару:

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} W^T \quad (3.13)$$

$$\frac{\partial E}{\partial X} = X^t \frac{\partial E}{\partial Y} \quad (3.14)$$

$$\frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y} \qquad (3.15)$$

Реалізація:

```python
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. 3.7. Python код для повнозв'язного шару (1)

```python

    # 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

    # 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. 3.8. 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.

$$Y = [f(x_1) \quad \dots \quad f(x_i)] = f(X) \tag{3.15}$$

Backward Propagation

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

$$\begin{aligned}
\frac{\partial E}{\partial X} &= \begin{bmatrix} \frac{\partial E}{\partial x_1} & \dots & \frac{\partial E}{\partial x_i} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial E}{\partial y_1} f'(x_1) & \dots & \frac{\partial E}{\partial y_1} f'(x_i) \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial E}{\partial y_1} & \dots & \frac{\partial E}{\partial y_1} \end{bmatrix} \odot [f'(x_1) \quad \dots \quad f'(x_i)] \\
&= \frac{\partial E}{\partial Y} \odot f'(X)
\end{aligned} \tag{3.16}$$

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. 3.9. 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 (Рис. 3.6). Повний лістинг файлу *activations.py* наведено в додатку А.

```python
import numpy as np


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


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

Рис. 3.10. Реалізації 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)**:

$$E = \frac{1}{n}\sum_{i}^{n}(y_i^* - y_i)^2 \tag{3.15}$$

Реалізація:

```python
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_back(y_true, y_pred):
    return 2 * (y_pred - y_true) / y_true.size
```

Рис. 3.11. Середнє квадратичне відхилення

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

$$E = \frac{1}{n}\sum_{i}^{n}(y_i^* - y_i)^2 \tag{3.15}$$

Реалізація:

```python

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


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

Рис. 3.12. 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. 3.8 представлена лише частина коду, повна реалізація наведена в Додатку А.

```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):
```

Рис. 3.13. Реалізація класу *Network*

3.3. Побудова моделі

Мережа готова до використання. Для перевірки роботи було створено два приклади:

1. "Hello world!" машинного навчання – звичайний XOR.

2. MNIST датасет, суть якого полягає в розпізнаванні написаної цифри.

Рішення XOR

XOR або виключна диз'юнкція – це простий спосіб зрозуміти чи нейронна мережа взагалі навчається.

Бінарний XOR *Таблиця 3.1*

$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a \oplus b}$
0	0	0
0	1	1
1	0	1
1	1	0

Результат

```
Epoch 996/1000  Error = 1e-05
Epoch 997/1000  Error = 1e-05
Epoch 998/1000  Error = 1e-05
Epoch 999/1000  Error = 1e-05
Epoch 1000/1000 Error = 1e-05
Predicted: [[[4.51320723e-04]]

 [[1.00541469e+00]]

 [[1.00045276e+00]]

 [[5.86043261e-04]]]
True: [[[0]]

 [[1]]

 [[1]]

 [[0]]]
```

Рис. 3.14. Робота *example_xor.py*

Результат роботи правильний, мережа працює так як очікувалося.

Рішення MNIST

MNIST датасет складається з цифр від 0 до 9 написаних від руки. Розмір кожного зображення 28x28, кількість каналів – 1. Задача – передбачити яка цифра написана на зображенні.

Хоча датасет складається із зображень, варто пам’ятати що кожне зображення це матриця, де кожне значення це частота пікселя в тій точці. Задача легко вирішується, якщо перетворити кожне зображення на матрицю з розмірами $[1, 784]$.

Результат

```
Epoch 87/100    Error = 0.00051
Epoch 88/100    Error = 0.00048
Epoch 89/100    Error = 0.00049
Epoch 90/100    Error = 0.0005
Epoch 91/100    Error = 0.00053
Epoch 92/100    Error = 0.0005
Epoch 93/100    Error = 0.00049
Epoch 94/100    Error = 0.00048
Epoch 95/100    Error = 0.00047
Epoch 96/100    Error = 0.00046
Epoch 97/100    Error = 0.00043
Epoch 98/100    Error = 0.00037
Epoch 99/100    Error = 0.00037
Epoch 100/100   Error = 0.00037
Test accuracy: 77.47%
```

Рис. 3.15. Робота *example_mnist_fc.py*

3.4. Висновки до розділу 3

Отже, була розроблена високорівнева бібліотека для побудови та тренування мереж прямого поширення для глибокого 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 дві невеликі моделі для вирішення прикладних тренувальних задач машинного навчання: XOR та MNIST dataset. Моделі 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 нейронної мережі прямого поширення.

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 поняття 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, вживані в обробці натуральних мов, рекурентні нейронні мережі та їх види: LSTM, GRU.

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. 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.
2. Клепко В.Ю Вища математика в прикладах і задачах: Навчальний посібник. 2-ге видання. / В.Ю. Клепко, В.Л. Голець - К.: Центр учбової літератури, 2009. - 594 с.
3. Дубовик В. П.; Юрик І. І. (2006 р.). *Вища математика*. / За ред. В.П. Дубовика. - К. : А.С.К., 2001. - 480 с.
4. van Rossum, G., Drake, F. L. (2011). *The Python Language Reference Manual*. Network Theory Ltd.
5. "Python : the holy grail of programming". *CERN Bulletin*. CERN Publications (31/2006). 31 July 2006. Retrieved 11 February 2012.
6. Deily, Ned (25 March 2019). Python 3.7.3 is now available. *Python Insider*. The Python Core Developers.
7. Chollet, F. (2017). *Deep Learning with Python*. Manning, 2017. — 384 p.
8. Deng, L. & Yu, D. (2014). *Deep Learning: Methods and Applications*., 2014
9. Distilling the knowledge in a neural network (2015), G. Hinton et al.
10. Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015), S. Loffe and C. Szegedy
11. G. Cybenko Approximation by Superpositions of a Sigmoidal Function, Math. Control Signals Systems (1989) 2:303-314

Електронні ресурси

12. Run and automate deep learning experiments faster [Електронний ресурс]. – Режим доступу: https://missinglink.ai/
13. The MIT License [Електронний ресурс]. – Режим доступу: https://opensource.org/licenses/mit-license.php
14. An end-to-end open source machine learning platform [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/
15. Theano [Електронний ресурс]. – Режим доступу: http://deeplearning.net/software/theano/
16. From research to production [Електронний ресурс]. – Режим доступу: https://pytorch.org/
17. scikit-learn Machine Learning in Python [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/
18. Keras. Simple. Flexible. Powerful. [Електронний ресурс]. – Режим доступу: https://keras.io/

ДОДАТКИ

Додаток А

```python
# activation functions and its derivative
def tanh(x):
    return np.tanh(x)


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


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def sigmoid_back(x):
    return sigmoid(x) * (1 - sigmoid(x))


def relu(x):
    return np.maximum(0, x)


def relu_back(x):
    return np.heaviside(x, 1)


def leaky_relu(x, alpha=0.01):
    y1 = ((x > 0) * x)
    y2 = ((x <= 0) * x * alpha)
    return y1 + y2


def leaky_relu_back(x, alpha=0.01):
    dx = np.ones_like(x)
    dx[x < 0] = alpha
    return dx


def swish(x, beta=1):
    return (x * sigmoid(beta * x))


def swish_back(x, beta=1):
    sig = sigmoid(x)
    return (sig * (1 + x * (1 - sig)))
```

Рис. А.1. *activations.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 np.asarray(result)

    # train the network
    def fit(self, x_train, y_train, epochs, learning_rate, verbose=False, x_val=None, y_val=None):
        # 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 {}/{}\tError = {}'.format(i + 1, epochs, round(err, 5)))

            if verbose:
                out = self.predict(x_val)
                out = np.argmax(out.reshape(out.shape[0], out.shape[2]), axis=1)
                true = np.argmax(y_val, axis=1)

                print('Validation accuracy: {}%'.format((out == true).mean() * 100))
```

Рис. А.2. *network.py*