МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра системного аналізу та управління

Т.А. Желдак, К.С. Хабарлак, Д.М. Гаранжа

САМОНАВЧАННЯ СКЛАДНИХ СИСТЕМ

Методичні рекомендації до виконання практичних робіт
для здобувачів ступеня магістра
освітньо-професійної програми «Системний аналіз»
зі спеціальності 124 Системний аналіз

Дніпро
НТУ «ДП»
2024

Желдак Т.А.

Самонавчання складних систем [Електронний ресурс] : методичні рекомендації до виконання практичних робіт для здобувачів ступеня магістра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / Т.А. Желдак, К.С. Хабарлак, Д.М. Гаранжа ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2024. – 66 с.

Автори:
Желдак Т.А., канд. техн. наук, доц.;
Хабарлак К.С., доктор філософії;
Гаранжа Д.М.

Затверджено науково-методичною комісією зі спеціальності 124 Системний аналіз (протокол № 3 від 10.05.2024) за поданням кафедри системного аналізу та управління (протокол № 5 від 10.05.2024).

Методичні рекомендації мають на меті допомогти студентам спеціальності 124 Системний аналіз у самостійному засвоєнні обов’язкової дисципліни «Самонавчання складних систем» під час виконання практичних робіт.

Розглянуто теоретичні засади різних класів алгоритмів самонавчання складних систем. Описано типову процедуру підготовки даних та навчання таких систем. Наведено приклади розв’язку практичних задач за наведеними алгоритмами.

Сформульовано вимоги до оформлення звіту до практичних робіт, питання для самоконтролю та критерії оцінювання практичних робіт.

Рекомендації орієнтовано на активізацію виконавчого етапу навчальної діяльності студентів.

Відповідальний за випуск завідувач кафедри системного аналізу та управління Т.А. Желдак, канд. техн. наук, доц.

Зміст

Вступ

Складні системи усюдисущі в різних областях, включаючи фізику, техніку, економіку та біологію. Ці системи часто демонструють нелінійну поведінку, що ускладнює їх моделювання та аналіз традиційними методами. З появою машинного навчання та нейронних мереж тепер стало можливим розробляти моделі самонавчання, які можуть розкривати приховані шаблони та взаємозв'язки в складних системах.

У даному посібнику подано методичні рекомендації щодо виконання практичних робіт з курсу «Самонавчання складних систем». Шість завдань, викладених у цьому посібнику, призначені для того, щоб надати студентам практичний досвід розробки та застосування різноманітних моделей машинного навчання та нейронних мереж до реальних проблем.

Завдяки цим завданням студенти отримають глибоке розуміння теоретичних основ кожної моделі, а також практичні навички, необхідні для їх реалізації за допомогою популярних мов програмування, таких як Python.

Мета дисципліни – сформувати у магістрів: 1) практичні навички обробки, аналізу, генерації даних провідними методами на основі нейронних мереж; 2) вміння будувати нейронні мережі, що відповідають задачі, та навчати їх; 3) здобути навички роботи із бібліотеками машинного навчання, зокрема TensorFlow та Keras, та мовою програмування Python для побудови нейронних мереж. Знання та навички, отримані в курсі, будуть корисними для подальшого працевлаштування здобувача.

Завдання курсу:

- навчитися проводити попередню обробку даних різної природи, здійснювати інтерполяцію невідомих значень;
- розуміти та навчитися застосовувати належні архітектури нейронних мереж, налаштовувати їх параметри, навчати їх та оцінювати якість роботи у відповідності до поставленої задачі;
- опанувати згорткові нейронні мережі, рекурентні нейронні мережі, підходи для відновлення та генерації сигналів, RBF-мережі та метод групового врахування аргументів;
- отримати практичні навички роботи з бібліотеками TensorFlow і Keras, мовою програмування Python для задач штучного інтелекту та побудови нейронних мереж різної архітектури.

Дисциплінарні результати навчання:

1. Знати сучасні наукові здобутки, вміти будувати та досліджу-вати архітектури глибоких нейронних мереж для обробки та розпізнавання даних різної природи. Вміти застосовувати різні підходи щодо оцінки якості роботи мережі на даних.

2. Знати сучасні наукові здобутки, вміти будувати та досліджувати нейронні мережі спрямовані на відновлення або генерацію сигналу.
3. Розуміти теоретичні основи та вміти застосовувати алгоритми машинного навчання для обробки неповних, частково визначених або пошкоджених сигналів.
4. Розуміти архітектуру повнозв’язної нейронної мережі, сферу її застосування, метод зворотного поширення помилки для навчання мережі.
5. Знати основні структурні блоки нейронних мереж спрямованих на обробку, розпізнавання та сегментацію зображень. Вміти будувати, навчати та застосовувати такі нейронні мережі.
6. Знати структурні блоки нейронних мереж спрямованих на обробку, класифікацію та генерацію текстів. Вміти оброблювати текст, будувати та навчати такі нейронні мережі.
7. Вміти проводити попередню обробку даних різної природи та інтерполювати невідомі значення.
8. Вміти будувати та навчати модель багатопараметричних даних, досліджувати вплив її параметрів на якість передбачення.

Практична робота №1 – Повнозв'язна мережа, метод зворотного поширення помилки

Мета роботи: закріплення теоретичних знань та розвинення навичок роботи із повнозв'язною мережею, методом її навчання. Використання нейронної мережі для передбачення на табличних даних.

Практична робота присвячена передбаченню невідомих даних за допомогою повнозв'язної нейронної мережі із використанням мови програмування Python та бібліотек TensorFlow, Keras. За узгодженням з викладачем студент може запропонувати свій набір табличних даних для розв'язання задачі практичної роботи. Допускається виконання роботи мовою програмування MATLAB.

1.1. Теоретичні відомості

Алгоритм зворотного поширення похибки (АЗПП) відноситься до класичних алгоритмів навчання повнозв'язаної нейронної мережі з учителем. В його основі лежить багаторазовий показ певного набору прикладів, для яких відомі наперед еталонні (визначені експериментально) відповіді, або ж бажані реакції нейронної мережі. Для кожного шару, починаючи з останнього, вихідного, у ході порівняння реакцій мережі y_j^p та d_j – еталонного значення ($j-$ номер нейрона у останньому шарі, загалом p шарів) похибка довільного нейрона довільного шару обчислюється за виразом

$$\delta_j^q = \sum_k \left(\delta_k^{q+1} w_{jk}^{q+1}\right) \frac{dy_j^q}{dS_j^q}, \tag{1.1}$$

де $q = \overline{1,p}$ - номер шару; $S_j^q = \sum_{i=1}^{k_{q-1}} w_{ij}^q \cdot x_{ij}^q$ - активація нейрона q-того шару; k_{q-1} – кількість нейронів у $q-1$ шарі.

Для вихідного шару розрахункова формула має вигляд

$$\delta_j^p = \left(d_j - y_j^p\right) \frac{dy_j^p}{dS_j^p}, \tag{1.2}$$

де $\frac{dy_j^p}{dS_j^p}$ - похідна від гладкої безперервної функції активації (частіше за все, простий сигмоїд або гіперболічний тангенс).

Величина корекції вагових коефіцієнтів q–того шару обчислюється за формулою

$$\Delta w_{ij}^q(t) = -\alpha\left(\mu \Delta w_{ij}^q(t-1) + (1+\mu)\delta_j^q y_j^{q-1}\right) \tag{1.3}$$

де α – крок навчання, μ – коефіцієнт інерційності, t – номер поточної ітерації.

Наведемо *алгоритм зворотного поширення помилки* для мережі у загальному вигляді, а потім розглянемо його на прикладі.

Крок 1. Ініціалізація вагових коефіцієнтів випадковими значеннями. Нормування вхідних значень у відповідності до обраної активаційної функції.

Крок 2. Подача на вхід навчального образу і розрахунок виходу мережі.

Крок 3. Розрахунок помилки вихідного шару.

Крок 4. Розрахунок помилки попереднього шару – повторюється для усіх шарів, аж до першого.

Крок 5. Корекція вагових коефіцієнтів останнього шару за формулою

$$w_{ij}^{q}(t) = w_{ij}^{q}(t-1) + \Delta w_{ij}^{q}(t), \tag{1.4}$$

де $\Delta w_{ij}^{q}(t)$ - обчислено за (1.3).

Крок 6. Корекція вагових коефіцієнтів усіх попередніх шарів аж до першого.

Крок 7. Якщо подані всі вхідні образи, то перехід на крок 8, інакше перехід на крок 2 (подається наступний образ).

Крок 8. Якщо помилка апроксимації $Er(w) = \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{m}\left(d_{ij} - y_{ij}^{p}\right)^2$ менша за наперед визначену величину ε, то перехід на крок 9, в іншому випадку – рандомізація образів й перехід на крок 2.

Крок 9. Подаємо на вхід навченої мережі контрольний вектор X і знаходимо *прогноз* Y.

Крок 10. Закінчення алгоритму.

Зауваження. Образи (приклади) бажано подавати у випадковому порядку, який змінюється при кожному поверненні на початок циклу навчання. Це необхідно аби сам порядок їх надходження не став ще одним віртуальним входом мережі і, відповідно, не вніс зміщення в результат.

Приклад Для мережі, що використовує у якості активаційної функції простий сигмоїд $f_2(z) = \frac{1}{1+e^{-x}}$ та складається з трьох нейронів вхідного шару, двох нейронів прихованого шару та двох виходів, розрахункові формули набувають вигляду:

- помилки вихідного шару (для кожного нейрону окремо для кожного образу):

$\delta_1^{(2)} = \left(d_{11} - y_1^{(2)}\right) y_1^{(2)} \left(1 - y_1^{(2)}\right)$ – перший нейрон,

$\delta_2^{(2)} = \left(d_{12} - y_2^{(2)}\right) y_2^{(2)} \left(1 - y_2^{(2)}\right)$ – другий нейрон.

- помилки прихованого шару (нейронів три, приклад той самий, що і для шару 2)

$\delta_1^{(1)} = \left(\delta_1^{(2)} w_{11}^{(2)} + \delta_2^{(2)} w_{12}^{(2)}\right) y_1^{(1)} \left(1 - y_1^{(1)}\right)$ – перший нейрон,

$\delta_2^{(1)} = \left(\delta_1^{(2)} w_{21}^{(2)} + \delta_2^{(2)} w_{22}^{(2)}\right) y_2^{(1)} \left(1 - y_2^{(1)}\right)$ – другий нейрон,

$\delta_3^{(1)} = \left(\delta_1^{(2)} w_{31}^{(2)} + \delta_2^{(2)} w_{32}^{(2)}\right) y_3^{(1)} \left(1 - y_3^{(1)}\right)$ – третій нейрон.

- корекції вагових коефіцієнтів нейронів другого шару (розмір матриці 3*2)

$$\Delta w_{11}^{(2)}(t) = -\eta\left(\mu \Delta w_{11}^{(2)}(t-1) + (1+\mu)\delta_1^{(2)} y_1^{(1)}\right);$$

$$\Delta w_{12}^{(2)}(t) = -\eta\left(\mu \Delta w_{12}^{(2)}(t-1) + (1+\mu)\delta_2^{(2)} y_1^{(1)}\right);$$

$$\Delta w_{21}^{(2)}(t) = -\eta\left(\mu \Delta w_{21}^{(2)}(t-1) + (1+\mu)\delta_1^{(2)} y_2^{(1)}\right);$$

$$\Delta w_{22}^{(2)}(t) = -\eta\left(\mu \Delta w_{22}^{(2)}(t-1) + (1+\mu)\delta_2^{(2)} y_2^{(1)}\right);$$

$$\Delta w_{31}^{(2)}(t) = -\eta\left(\mu \Delta w_{31}^{(2)}(t-1) + (1+\mu)\delta_1^{(2)} y_3^{(1)}\right);$$

$$\Delta w_{32}^{(2)}(t) = -\eta\left(\mu \Delta w_{32}^{(2)}(t-1) + (1+\mu)\delta_2^{(2)} y_3^{(1)}\right).$$

- корекції вагових коефіцієнтів першого шару (розмір матиці 3*3)

$$\Delta w_{11}^{(1)}(t) = -\eta\left(\mu \Delta w_{11}^{(1)}(t-1) + (1+\mu)\delta_1^{(1)} x_1\right);$$

$$\Delta w_{12}^{(1)}(t) = -\eta\left(\mu \Delta w_{12}^{(1)}(t-1) + (1+\mu)\delta_2^{(1)} x_1\right);$$

$$\Delta w_{13}^{(1)}(t) = -\eta\left(\mu \Delta w_{13}^{(1)}(t-1) + (1+\mu)\delta_3^{(1)} x_1\right);$$

$$\Delta w_{21}^{(1)}(t) = -\eta\left(\mu \Delta w_{21}^{(1)}(t-1) + (1+\mu)\delta_1^{(1)} x_2\right);$$

$$\Delta w_{22}^{(1)}(t) = -\eta\left(\mu \Delta w_{22}^{(1)}(t-1) + (1+\mu)\delta_2^{(1)} x_2\right);$$

$$\Delta w_{23}^{(1)}(t) = -\eta\left(\mu \Delta w_{23}^{(1)}(t-1) + (1+\mu)\delta_3^{(1)} x_2\right);$$

$$\Delta w_{31}^{(1)}(t) = -\eta\left(\mu \Delta w_{31}^{(1)}(t-1) + (1+\mu)\delta_1^{(1)} x_3\right);$$

$$\Delta w_{32}^{(1)}(t) = -\eta\left(\mu \Delta w_{32}^{(1)}(t-1) + (1+\mu)\delta_2^{(1)} x_3\right);$$

$$\Delta w_{33}^{(1)}(t) = -\eta\left(\mu \Delta w_{33}^{(1)}(t-1) + (1+\mu)\delta_3^{(1)} x_3\right).$$

Зауваження. Не існує універсальної поради щодо вибору значень α та η, які б забезпечували максимально точне й швидке навчання нейронної мережі. Ці значення обираються емпірично в діапазоні $(0; 1)$.

Зазвичай хороший результат забезпечує значення коефіцієнту навчання $\eta = 0{,}1$ та коефіцієнту інертності $\mu = 0{,}9$. Вони дозволяють зменшити різкі зміни вагових коефіцієнтів при надходженні кожного нового прикладу.

1.2. Приклад розв’язування задачі

Для реалізації повнозв’язної мережі використаємо бібліотеки TensorFlow та Keras. Далі наведено приклад як їх використовувати:

1. Встановлення необхідних бібліотек

Для встановлення бібліотек TensorFlow та Keras необхідно ввести в консоль команду:

```
pip install tensorflow
```

2. Імпорт необхідних модулів

Спочатку необхідно імпортувати необхідні модулі з scikit-learn:

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
```

3. Створення штучного набору даних

Давайте створимо штучний набір даних із двома вхідними ознаками та двома класами:

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])
```

4. Визначити архітектури мережі

Далі необхідно визначити архітектуру повнозв’язної мережі:

```
model = Sequential()
model.add(Dense(10, input_dim=2, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='sgd',
metrics=['accuracy'])
```

У цьому прикладі ми створюємо повнозв’язну мережу з одним прихованим шаром, що містить 10 нейронів.

5. Навчання моделі

Тепер можна навчити модель за допомогою раніше створеного набору даних:

```
model.fit(X, y, epochs=500, batch_size=32)
```

Максимальна кількість епох навчання встановлена на 500.

6. Прогнозування

Тепер можна використовувати навчену модель для прогнозування та оцінки моделі:

```
predictions = model.predict(X)
print(predictions)
print(model.evaluate(X, y))
```

Це виведе прогнозовані класи, помилку та точність для вхідних даних.

1.3. Індивідуальне завдання

1. Виконати програмну реалізацію нейронної мережі повнозв'язної нейронної мережі та алгоритму навчання зворотного поширення похибки для трьохшарової нейронної мережі із трьома нейронами першого шару, трьома нейронами – другого і двома нейронами – третього скориставшись прикладним пакетом.

2. Використовуючи дані таблиці початкових даних (табл. 1.1), написати програмний фрагмент для обчислення 27 значень функції по заданих значеннях аргументів та їх варіаціях з кроком ±1 і знайти їх середнє значення. Розділити вибірку на навчальну (20 прикладів) та тестову (7 прикладів). Сформувати істині виходи y як логічну функцію, що дорівнює "1", якщо d більше середнього значення, та "0", якщо менше.

3. Подати задані значення прикладів з навчальної вибірки на вхід нейронної мережі.

4. Використовуючи контрольні приклади з тестової вибірки оцінити точність алгоритму. Оцінити кількість епох навчання, необхідних для досягнення максимум точності.

5. Побудувати графік залежності точності на тестовій вибірці в залежності від кількості епох навчання.

6. Дослідити нейронну мережі прямого поширення, побудувавши графіки залежності:

- величини середньої помилки навченої мережі із заданою кількістю прикладів від кількості нейронів у прихованому шарі (в межах від 1 до 10);

- величини середньої помилки навченої мережі від кількості прикладів, що використовуються для навчання, тестові приклади при цьому фіксовані)

зменшувати від 20 до тих пір, поки помилка не сягне заданого рівня, наприклад 0,5;

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, розрахунки, необхідні описи та графіки (відповідно до завдання вище) та вихідний код програм.

Назва файлу: Прізвище_група

Роботи, виконані не за варіантом, не приймаються.

Початкові дані знаходяться в наступній таблиці. Номер варіанту відповідає порядковому номеру студента в списку групи.

Вхідні змінні приймають значення в діапазоні $[x_i^* - 1; x_i^* + 1]$, де x_i^* – базове значення i –тої змінної. Наприклад, для варіанту 7 змінна x_1 має базове значення 2. Це означає, що навчальна і тестова вибірки для навчання нейронної мережі за цим варіантом мають містити комбінації, в яких x_1 приймає значення 1, 2 або 3.

Таблиця 1.1 – Початкові дані

Варіант	Вхідні дані			Функція для моделювання
	x_1	x_2	x_3	
1	1	2	3	$d = x_1^2 - x_2^2 + x_3^2$
2	5	4	3	$d = \sin x_1 + \sin x_2 - \sin x_3$
3	9	8	7	$d = \tan x_1 + \sin x_2 - \sin x_3$
4	8	5	3	$d = \sin x_1 + \sin x_2 - \cos x_3$
5	5	6	7	$d = \tan x_1 + \sin x_2 - \tan x_3$
6	1	8	7	$d = \sin x_1 + \tan x_2 - \tan x_3$
7	2	5	4	$d = \cos x_1 + \cos x_2 - \sin x_3$
8	5	7	8	$d = \ln(\lvert\cos x_1\rvert) + \tan x_2 - \mathrm{ctg} x_3$
9	2	3	6	$d = 2^{x_1} + \cos x_2 - \sin x_3$
10	7	4	5	$d = \sin(x_2^2) + x_1^2 - \tan x_3$
11	2	2	3	$d = x_1 \sin x_2 + \cos x_3$
12	4	4	3	$d = 2x_1 + \frac{x_2}{x_3}$
13	9	10	7	$d = \frac{x_1}{x_2} \sin x_3$
14	8	6	3	$d = \cos(x_1 + x_2) + \sin x_3$
15	5	2	7	$d = x_2(\sin x_1 + \cos x_2)$
16	1	9	7	$d = x_1 + 2(x_2 - x_3^2)$
17	3	4	4	$d = x_1(\tan x_2 + \cos x_3)$

18	3	7	4	$d = \sin(x_1 - x_2 + 2x_3)$
19	3	4	6	$d = \cos(x_1 - 2x_2 + x_3)$
20	7	4	4	$d = \text{ctg}(x_1 + x_2 - x_3)$

1.4. Контрольні питання

1. Що таке вхідний, вихідний, прихований шари повнозв’язної нейронної мережі?
2. За якою формулою оновлюються ваги шарів повнозв’язної нейронної мережі?
3. В яких межах зазвичай обирається крок навчання для повнозв’язної мережі?
4. В якому прикладному пакеті реалізовано повнозв’язну нейронну мережу?
5. Які функції використовуються для навчання, передбачення та оцінки якості мережі?
6. Яка суть методу зворотного поширення помилки?
7. Чи можливе поширення помилки з початку мережі в методі поширення помилки? Чому?
8. Про що свідчить точність навчання 0,5 в задачі бінарної класифікації?

Практична робота №2 – Інтерполяція даних за допомогою радіально-базисних функцій

Мета роботи: закріплення теоретичних знань та розвинення навичок роботи із мережею з радіально-базисними функціями. Інтерполяція даних.

Практична робота присвячена вирішення задачі інтерполяції даних. Робота виконується із використанням мови програмування Python та бібліотек Keras та TensorFlow. За узгодженням з викладачем студент може запропонувати свій набір даних для розв'язання задачі практичної роботи. Допускається виконання роботи мовою програмування MATLAB.

2.1. Теоретичні відомості

Мережа RBF (radial-basis function), як і більшість інших нейромереж, призначена для апроксимації функцій, які задані в неявному вигляді набором шаблонів (навчальних образів). Теорема Ковера (Cover) говорить про те, що для набору образів у лінійному просторі роздільна здатність апроксиматора підвищується з підвищенням ступеню функції апроксиматора.

Доведено, що ймовірність P того, що певна випадково сформована множина з рівно ймовірним представництвом елементів двох класів кількістю N абсолютно розділяється гіперповерхнею m_1 −того порядку, дорівнює

$$P(N, m_1) = \left(\frac{1}{2}\right)^{N-1} \sum_{m=0}^{m_1-1} \binom{N-1}{m}, \quad (2.1)$$

де $\binom{N-1}{m} = \frac{(N-1)!}{m!(N-1-m)!}$ - біноміальні коефіцієнти.

З (2.1) витікають дві основні особливості мереж RBF у порівнянні з багатошаровим персептроном:

1) прихований шар має бути значно потужнішим за вхідний та вихідний;
2) активаційна функція прихованого шару має бути нелінійною по всіх вимірах з однаковим радіусом σ.

Остання властивість й обумовила назву таких мереж.

Нейромережа характеризується такими особливостями: має єдиний прихований шар, нейрони прихованого шару мають нелінійну активаційну функцію, синаптичні ваги всіх нейронів прихованого шару дорівнюють одиниці. Розглянемо наступні позначення:

$c = (c_1, c_2, \ldots, c_H)$ – вектор координат центрів активаційних функцій нейронів прихованого шару;

σ_j – ширина вікна активаційної функції j -го нейрона прихованого шару;

$f(X,c) = e^{\frac{-\sum_{j=1}^{n}(x_j - c_j)^2}{\sigma_j^2}}$ – радіально-симетрична активаційна функція нейрона прихованого шару (див. рис. 2.1);

w_{ij} – вага зв'язку між i −тим нейроном прихованого шару та j-тим нейроном вихідного шару.

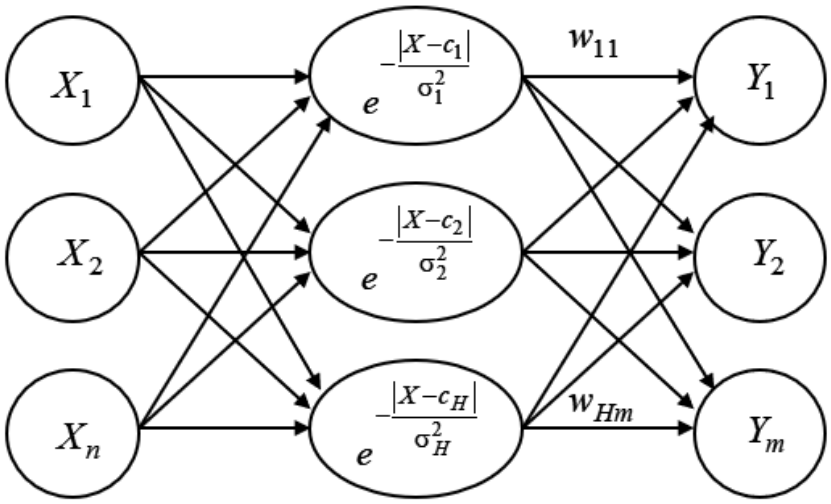


Рис. 2.1. Радіально-базисна нейронна мережа.

Відмінність радіально-базисних мереж від розглянутого раніше багатошарового персептрона полягає в наявності лише одного прихованого шару, з яким всі нейрони вхідного шару пов'язані рівними одиничними зв'язками. Якщо в персептроні як прихований шар (яких може бути кілька), так і вихідний є обчислюючими, то в радіально-базисній мережі обчислює лише один прихований шар, вихідний натомість забезпечує лінійну згортку нелінійних функцій. Найголовніша ж відмінність – це аргумент функції активації: в перцептроні він є скалярним добутком вхідного вектора та вектора вагів, у RBF – відстань від вхідним вектором та центром даного нейрона.

Існує кілька методів навчання радіально-базисної мережі – як ітераційних з рекурсією, так і прямих. Вони можуть використовувати самоорганізацію або вчителя. Далі розглянемо один з методів навчання RBF-мережі, що не містить рекурсії. Алгоритм цього методу, що має назву випадкового вибору фіксованих центрів, є наступним:

Крок 1. Обрати розмір прихованого шару H рівним кількості тренувальних шаблонів Q . Синаптичні ваги нейронів прихованого шару прийняти рівними 1.

Крок 2. Розмістити центри активаційних функцій нейронів прихованого шару в точках простору вхідних сигналів мережі, які входять до набору навчальних образів Ξ: $c_j = \overline{X_j}$, для всіх $j = \overline{1,H}$.

Крок 3. Обрати ширини вікон активаційних функцій нейронів прихованого шару σ_j для всіх $j = \overline{1,H}$ достатньо великими, але так, щоб перетин поверхонь, що визначаються активаційним функціями, був мінімальним в просторі вхідних образів.

Крок 4. Визначити ваги нейронів вихідного шару нейронної мережі w_{ij} для усіх $i = \overline{1,H}$, $j = \overline{1,m}$. Для цього пред'явити мережі весь набір навчальних образів. Вихід i −того нейрона прихованого шару для p −того образу буде таким:

$$Y_i = w_{i1}f(X_p, c_1) + w_{i2}f(X_p, c_2) + \ldots + w_{iH}f(X_p, c_H) = \\ = w_{i1}f(X_p, X_1) + w_{i2}f(X_p, X_2) + \ldots + w_{iH}f(X_p, X_H). \quad (2.2)$$

Для відповідності виходу мережі еталонному зразку потрібно, аби $Y_i = D_p$.

Прирівнявши внутрішній вираз з його ідеальним значенням, можемо отримати систему лінійних алгебраїчних рівнянь, яку зручно записати в матричній формі:

$$F \cdot W = D, \quad (2.3)$$

у якому $F = \begin{pmatrix} f_{11} & f_{12} & \ldots & f_{1H} \\ f_{21} & f_{22} & \ldots & f_{2H} \\ \ldots & \ldots & \ldots & \ldots \\ f_{H1} & f_{H2} & \ldots & f_{HH} \end{pmatrix}$ – інтерполяційна матриця відстаней i −того зразку від j −того центру;

$W = \begin{pmatrix} w_{11} & w_{12} & \ldots & w_{1m} \\ w_{21} & w_{22} & \ldots & w_{2m} \\ \ldots & \ldots & \ldots & \ldots \\ w_{H1} & w_{H2} & \ldots & w_{Hm} \end{pmatrix}$ – матриця початкових синаптичних вагових коефіцієнтів;

$D = \begin{pmatrix} d_{11} & d_{12} & \ldots & d_{1m} \\ d_{21} & d_{22} & \ldots & d_{2m} \\ \ldots & \ldots & \ldots & \ldots \\ d_{H1} & d_{H2} & \ldots & d_{Hm} \end{pmatrix}$ – матриця заданих образів.

Рішення системи (2.3) досить просто знаходиться на вихідному просторі матричним методом:

$$W = F^{-1}D \quad (2.4)$$

Таким чином, ми одержимо шукані синаптичні вагові коефіцієнти, що забезпечують найкраще проходження інтерполяційної поверхні через навчальні образи в просторі вхідних образів.

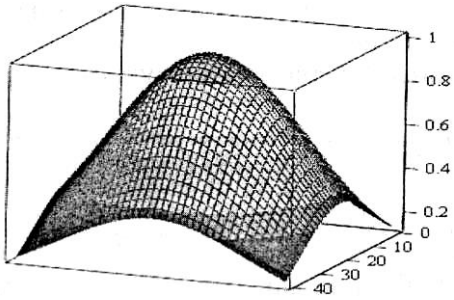


Рис 2.2. Активаційна функція

За відносною простотою побудови й навчання радіально-базисної нейронної мережі ховається цілий ряд її недоліків, які обумовлюють не надто часте застосування цього інструменту в задачах аналізу.

Насамперед, мережа RBF є вкрай чутливою до величини ширини вікон активаційної функції σ. Двомірний аналог активаційної функції зображено на рис. 2.2. Мало того, що для кожного нейрона σ має бути своя, вона ще й має забезпечувати не конфліктність нейронів між собою та, водночас, рівномірність охоплення всього діапазону визначення вхідного масиву даних. Для того аби уникнути «паралічу» мережі через неповну визначеність певних областей вхідних даних і не було великих помилок при апроксимації, вимагаємо виконання наступної умови:

$$-2 < \frac{\sqrt{\sum_{i=1}^{n}\left(x_{ij} - c_i\right)^2}}{\sigma} < 2 \tag{2.5}$$

(вибір границь інтервалу достатньо довільний, але його раціональність підтверджена експериментами) або, обмежившись вибором додатного σ:

$$0 < \frac{\sqrt{\sum_{i=1}^{n}\left(x_{ij} - c_i\right)^2}}{\sigma} < 2. \tag{2.6}$$

Враховуючи, що після нормування $x_{ij}, c_i \in [0; 1]$, одержимо $0 \leq \sum_{i=1}^{n}\left(x_{ij} - c_i\right)^2 \leq n$ для всіх навчальних образів. Тоді $\sigma > \sqrt{n}/2$.

Можна задатися великими значеннями радіусів, але при їх збільшенні зменшується ексцес і ростуть «хвости» графіків активаційної функції, що знову-таки призводить до «паралічу» мережі, але вже в інший бік – неможливості розрізнити образи між собою. Прийнятні результати були отримані в реальних задачах при

$$\frac{\sqrt{n}}{2} < \sigma < \frac{3\sqrt{n}}{2} \tag{2.7}$$

Ще одним недоліком застосування мереж RBF є необхідність виконання певних підготовчих операцій для адекватного навчання і використання навченої мережі RBF. По-перше, треба максимізувати спільну ентропію початкових образів, наприклад, за допомогою відомих методів головних компонент або «вибілювання» входів. Перетворені дані забезпечать якісне та швидке навчання на множині даних мінімальної потужності. Оскільки вказані методи достатньо трудомісткі, необхідно із всієї множини вхідних образів обирати ті, які мають максимальну сумарну попарну евклідову відстань. Наступним кроком має стати нормування.

Неітераційний метод навчання RBF-мережі не завжди є оптимальним методом навчання. Зокрема, коли кількість вхідних образів є великою, застосування градієнтних методів навчання дозволяє зменшити кількість нейронів прихованого шару.

Нарешті, сама мережа RBF має істотне обмеження: якщо багатошарові персептрони забезпечують глобальну апроксимацію нелінійного відображення, радіально-базисні мережі – лише локальну.

Процес функціонування мережі RBF має ще багато особливостей. Але одну варто згадати: за допомогою такої мережі можна розв'язувати лише задачу інтерполяції.

В той же час мережі RBF мають кілька переваг, зокрема:

- Універсальна апроксимація: мережі RBF можуть апроксимувати будь-яку безперервну функцію з довільною точністю, враховуючи достатню кількість нейронів.
- Швидке навчання: процес навчання, як правило, відбувається швидше порівняно з іншими архітектурами нейронних мереж.
- Проста архітектура: проста трирівнева архітектура робить мережі RBF легшими для реалізації та розуміння.

Застосування мереж RBF:

- Класифікація: Мережі RBF використовуються в задачах розпізнавання образів і класифікації, таких як розпізнавання мови та класифікація зображень.
- Регресія: ці мережі можуть моделювати складні зв’язки в даних для завдань прогнозування.
- Апроксимація функцій: мережі RBF ефективні в апроксимації нелінійних функцій.

2.2. Приклад розв'язування задачі

Мережу з радіально-базисними функціями достатньо легко реалізувати за допомогою бібліотек Keras та TensorFlow. Далі наведено приклад використання даних бібліотек на штучному наборі даних:

1. Імпорт необхідних модулів

Спочатку необхідно імпортувати необхідні модулі:

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Input
from keras.initializers import RandomUniform, Constant
import tensorflow as tf
import numpy as np
```

2. Створення штучного набору даних

Створимо штучний набір даних із двома ознаками та безперервною цільовою змінною:

```
X = np.array([[0.1, 0.2], [0.1, 1.3], [1.1, 0.5], [1.1, 1.2]])
y = np.array([0.0, 1.5, 1.2, 0.3])
```

3. Визначення шару RBF

```
class RBFLayer(keras.Layer):
    def __init__(self, output_dim):
        self.output_dim = output_dim
        super(RBFLayer, self).__init__()

    def build(self, input_shape):
        self.centers = self.add_weight(name='centers',
                                       shape=(self.output_dim,
input_shape[1]),
                                       initializer=RandomUniform(0.0,
1.0),
                                       trainable=True)
        self.betas = self.add_weight(name='betas',
                                     shape=(self.output_dim,),
                                     initializer=Constant(1.0),
```

```
                                    trainable=True)

        super(RBFLayer, self).build(input_shape)

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.output_dim)

    def call(self, x):
        C = tf.expand_dims(self.centers, -1)  # inserts a dimension of 1
        H = tf.transpose(C-tf.transpose(x))  # matrix of differences
        return tf.exp(-self.betas * tf.math.reduce_sum(H**2, axis=1))
```

4. Ініціалізація

Далі необхідно ініціалізувати модель Keras з радіально-базисним ядром (RBF):

```
model = Sequential()
model.add(Input(shape=(2,)))
model.add(RBFLayer(10))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='sgd')
```

5. Навчання

```
model.fit(X, y, epochs=500, batch_size=32)
```

6. Прогнозування та виведення середньоквадратичної помилки

```
predictions = model.predict(X)
print(predictions)
print(model.evaluate(X, y))
```

Порівняйте середньоквадратичні оцінки регресора, отримані з різними значеннями кількості нейронів шару RBF та епох навчання. Чим менше MSE, тим краще модель відповідає даним.

Зауважимо, що вибір значень кількості нейронів та епох навчання залежить від конкретної задачі та набору даних. На практиці необхідно провести дослідження різних значень, щоб знайти найкращу комбінацію для вашої конкретної проблеми.

2.3. Індивідуальне завдання

1. **Початкові дані** знаходяться в файлі *lab_ccc_2.xlsx* в робочому каталозі дисципліни на сервері Moodle. Вхідні змінні *X1 – X3* – спільні для всіх. Вихідна змінна *Yi* обирається за наступним принципом: номер варіанту відповідає останній цифрі порядкового номеру студента в списку групи.
2. Розбити початкові дані в пропорції 80/20 на навчальні та тестові.
3. Здійснити програмну реалізацію радіально-базисної нейронної мережі. При реалізації забезпечити можливість динамічної зміни кількості нейронів та епох навчання.
4. Побудувати графіки залежності якості мережі від кількості нейронів та кількості епох. Зробити висновок щодо оптимальних значень зазначених величин.
5. Дослідити: залежність якості мережі на тестовій вибірці від кількості навчальних прикладів. Побудувати відповідний графік залежності. Зробити висновок, чи є необхідним використання всіх даних для навчання? Чи є надана кількість даних достатньою для навчання радіально-базисних функцій?

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, розрахунки, необхідні описи та графіки (відповідно до завдання вище) та вихідний код програм.

Назва файлу: Прізвище_група

Роботи, виконані не за варіантом, не приймаються.

2.4. Контрольні питання

1. Якими є переваги мережі RBF?
2. Які є параметри мережі RBF? Яке їх значення?
3. За домогою яких прикладних пакетів можна реалізувати мережу RBF?
4. На що спрямовано використання мережі RBF, на інтерполяцію даних або екстраполяцію? Відповідь обґрунтуйте.
5. Що таке ширина вікон активаційних функцій в мережі RBF?
6. Зазначте в яких випадках відбувається «параліч» мережі RBF?
7. Для мережі RBF, реалізованої за допомогою Keras, як саме можна змінити кількість нейронів шару RBF?
8. Що таке середньоквадратична помилка? Яка її формула?

Практична робота №3 – Метод групового урахування аргументів

Мета роботи: закріплення теоретичних знань та розвинення навичок роботи із методом групового урахування аргументів, побудова полінома. Дослідження впливу параметрів моделі МГУА на якість передбачення.

Практична робота присвячена побудові та навчанні модель багатопараметричних даних, досліджувати впливу її параметрів на якість передбачення. Робота виконується із використанням мови програмування Python та бібліотеки GmdhPy. За узгодженням з викладачем студент може запропонувати свій набір даних для розв'язання задачі практичної роботи. Допускається виконання роботи за допомогою інших пакетів прикладних програм.

3.1. Теоретичні відомості

Нехай початкові дані зосереджені в матриці $A = (X_1, X_2, \ldots, X_n, Y)$, де X_i, для всіх $i = \overline{1,n}$ та Y – вектори-стовпчики розмірністю N, серед яких X_i – вхідні фактори, а Y – вихідна характеристика. Задача полягає в ідентифікації залежності

$$Y = F(X_1, X_2, ..., X_n) \tag{3.1}$$

поліномом Колмогорова-Габора у загальному вигляді

$$Y = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{i,j=1}^{n} a_{ij} x_i x_j + \sum_{i,j,k=1}^{n} a_{ijk} x_i x_j x_k + \ldots \tag{3.2}$$

Відомо, що при збільшенні порядку полінома точність наближення ним функції *F(x)* зростає, а потім спадає. Інакше кажучи, нарощуючи максимальний ступінь полінома S, маємо наступний вигляд від цього ступеню середньоквадратичної похибки апроксимації (рис. 3.1). Якщо точність є максимальною, то цей процес закінчується.

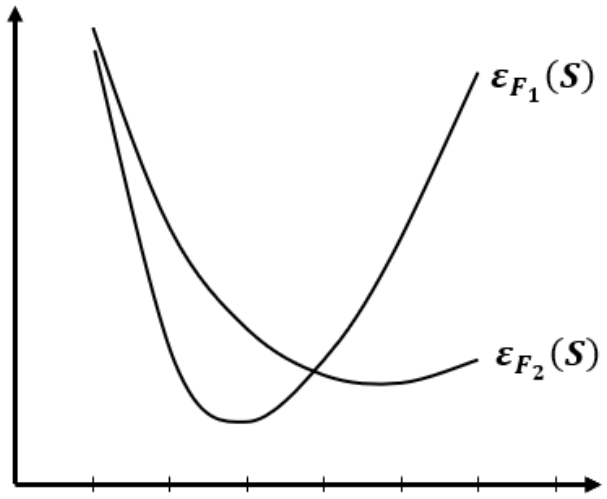


Рис. 3.1 – Зміна точності різних моделей МГУА при збільшенні складності

Особливістю МГУА є те, що він може бути застосований у випадку малої кількості точок експериментів, навіть значно меншої, ніж кількість членів полінома. Це пояснюється тим, що на кожному етапі моделювання апроксимація

виконується не за допомогою повного поліному поточної складності, а за допомогою елементарної опорної функції.

Опорна функція обирається на першому етапі реалізації МГУА. Найчастіше в якості опорних використовуються наступні залежності:

1) мультиплікативна

$$y = a_0 + a_1 x_i x_j; \tag{3.3}$$

2) адитивна

$$y = a_0 + a_1 x_i + a_2 x_j; \tag{3.4}$$

3) повна першого порядку

$$y = a_0 + a_1 x_i + a_2 x_j + a_3 x_i x_j; \tag{3.5}$$

4) повна другого порядку

$$y = a_0 + a_1 x_i + a_2 x_j + a_3 x_i^2 + a_4 x_j^2 + a_5 x_i x_j. \tag{3.6}$$

Для першої функції необхідні початкові дані хоча б трьох експериментів, для другої – 4, для третьої – 5, для четвертої – 7. Це пояснюється тим, що для визначення коефіцієнтів буде використано метод найменших квадратів, в якому необхідно мати хоча б одну ступінь свободи, аби отримана залежність мала ступінь довіри, відмінний від 0.

Позначимо $y_k = f(x_i, x_j)$, де f – одна із залежностей (3.3)-(3.6) або, можливо, подібна. На наступному кроці за допомогою МНК визначають коефіцієнти p рівнянь, де (без урахування повторів та діагональних елементів) $p = \frac{n(n-1)}{2}$.

Після того, як усі залежності $y_k = f(x_i, x_j), k = \overline{1, p}$ ідентифіковані за МНК, до справи вступає зовнішній критерій, за яким із усієї множини відбирають найкращі моделі.

Для забезпечення роботи зовнішнього критерію початкову вибірку слід поділити на навчальну та перевірочну. В залежності від обраного критерію якості, навчальна вибірка має містити 50-70% рядків початкової таблиці даних, але мінімум на один рядок більше, ніж порядок опорної функції.

Аби одразу відсіяти «неблагонадійні» моделі, у вигляді обмеження застосовується допоміжний критерій точності, відомий з дисперсійного аналізу

$$\delta_k^2 = \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i^k)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}, \tag{3.7}$$

де y_i – табличне значення шуканої залежності в i-тому спостереженні; $\hat{y}_i^k$ – прогнозоване значення в тому ж рядку таблиці за k-тою моделлю; $\bar{y}$ – середнє значення шуканої величини.

Якщо за (3.7) отримано значення $\delta_k^2 < 0{,}5$, кажуть, що модель перспективна й може застосовуватися далі, якщо $0{,}5 \le \delta_k^2 < 0{,}8$ – модель може використовуватись, але обережно, при $0{,}8 \le \delta_k^2 < 1$ використання k-тої моделі

небажано, а при $\delta_k^2 \geq 1$ модель не може застосовуватися, оскільки вносить дезінформацію.

Серед моделей, що задовольняють обмеженню за (3.7), визначають Q кращих за критеріями регулярності та незміщеності (про них – далі). Q може бути константою (деякі дослідники встановлюють наперед $Q = n$, аби не мати проблем зі зміною розмірності задачі), зростати чи зменшуватися у функції етапу S.

Ті моделі залежності, які залишилися, перенумеровують і одержують нову таблицю зі значень $y_1, y_2, \ldots, y_Q$. На цьому перший крок селекції закінчено, якщо розрахувати та запам'ятати ефективність найкращої з моделей. На наступному кроці все викладене повторюється: за допомогою МНК визначають коефіцієнти таких самих опорних функцій, але вже від нових змінних

$$z_1 = f(y_1, y_1); z_2 = f(y_1, y_2); \ldots$$
$$z_Q = f(y_1, y_Q); z_{Q+1} = f(y_2, y_2); \ldots \quad (3.8)$$
$$z_p = f(y_n, y_n)$$

Зрозуміло, що процес має циклічний характер і повторюється, допоки значення зовнішнього критерію покращуються. Селекція припиняється, коли її подальше ускладнення неможливе за (4.7) або погіршує значення зовнішнього критерію. Тоді говорять, що оптимальна за складністю модель знайдена й здійснюють зворотній процес розкриття отриманого виразу.

Умови закінчення ітерацій не канонізовані і можуть бути, наприклад, такими:

- середнє значення помилки для наступного ряду селекції є більшим ніж найбільше (середнє) значення помилки для попереднього ряду;
- мінімальне значення помилки наступного ряду більше мінімального значення помилки попереднього ряду;
- максимальне значення помилки наступного ряду більше максимального значення помилки попереднього раду;
- модуль відхилення помилок наступного і попереднього ряду менше деякого числа.

Критерії регулярності

Опишемо зовнішні критерії, використання яких базується на принципі зовнішнього доповнення. В залежності від типу задачі О.Г. Івахненко запропонував розглядати такі критерії: регулярності, незміщеності та балансу змінних. Відомі два критерії регулярності:

- мінімум середньоквадратичної помилки на нових точках окремої контрольної послідовності;

$$\varepsilon = \frac{\sum_{i=1}^{k}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{k} y_i^2} \to \min; \qquad (3.9)$$

- максимум коефіцієнта кореляції на тих же точках

$$RR = \frac{\sum_{i=1}^{k}(y_i \cdot \hat{y}_i)}{\sum_{i=1}^{k} y_i^2} \to \max. \qquad (3.10)$$

Розглянемо процедуру їх застосування. Початкові дані знаходяться у таблиці. Розділимо її рядки на дві частини (приблизно 60% на 40%), тоді $N = l + k$, де l - кількість точок експерименту в першій (навчальній) вибірці, k - у другій (контрольній). Значення l, нагадаємо, повинне бути більшим від числа доданків в опорній функції $f(x_i, x_j)$.

Використовуючи елементи навчальної вибірки, визначаємо коефіцієнти тієї з залежностей (3.3)-(3.6), яка прийнята за опорну функцію для усіх p сполучень стовпців, перевіряючи кожну модель за критерієм точності (3.7).

Далі розраховуємо значення критерію регулярності (3.9) чи (3.10) на точках контрольної вибірки, після чого впорядковуємо моделі від кращого значення критерію до гіршого і залишаємо з них певну кількість Q кращих. Після перенумерації вони складуть множину функцій наступного ряду селекції.

Перевагою критеріїв регулярності є плавність зміни їх значення при збільшенні складності моделі. Недоліком їх використання є низька точність при розв'язанні екстраполяційних задач. Тому критерії регулярності раціонально застосовувати для ідентифікації, інтерполяції та короткострокового прогнозу.

Критерії незміщеності

Відомі три види критерію незміщеності, що може обчислюватись на базі аналізу розв'язків, аналізу коефіцієнтів та як «критерій відносної незміщеності».

Критерій незміщеності, що базується на аналізі розв'язків (КН1). Для розрахунку КНl необхідно ранжувати всі точки експериментів за збільшенням або зменшенням значення дисперсії. Процедура ранжування описана нижче. Після ранжування точки експериментів нумерують і ділять на дві послідовності:

- до першої відносять точки з парними номерами, їх кількість N_1,

- до другої – точки з непарними, їх кількість N_2. Логічно, що $N_1 + N_2 = N$.

На першому ряді селекції перша послідовність є навчальною, друга – контрольною. Отримані на навчальній послідовності N_1 рівняння регресії позначимо $y_k^{'} = f(x_i, x_j)$.

Далі першу послідовність вважають контрольною, другу – навчальною. На навчальній послідовності знаходять рівняння регресії $y_k'' = f(x_i, x_j)$. Кількість рівнянь y_k' та y_k'' повинна співпадати, випадок невиконання цієї умови не розглядаємо. Для кожного k-того рівняння розраховують середньоквадратичне відхилення

$$KH1_k = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_{k_i}^{'} - y_{k_i}^{''}\right)^2} \to min, \tag{3.11}$$

де k_i – номер рядка у таблиці значень k-тої моделі. Моделі з найменшим значенням критерію вважаються найточнішими.

Як і за критерієм регулярності, за критерієм КН1 обирають Q найкращих рівнянь, що відповідають меншій оцінці критерію.

Для визначення фактора зупинки алгоритму визначають середнє зважене значення критерію незміщеності моделей даного покоління

$$\overline{KH1}_s = \frac{1}{Q}\sum_{k=1}^{Q} KH1_k \tag{3.12}$$

На другому і подальших рядах селекції s процедура залишається тією самою. Селекція продовжується доти, доки середнє значення критерію незміщеності зменшується ($\overline{KH1}_s < \overline{KH1}_{s-1}$).

Критерій незміщеності, що базується на аналізі коефіцієнтів (КН2). Точки експериментів ранжуються за величиною дисперсії і поділяються навпіл на навчальну та контрольну послідовності. Точки з більшим значенням дисперсії потрапляють до навчальної послідовності, з меншим – до контрольної. Особливість критерію полягає в тому, що на кожному ряді селекції ранжування і розділення точок експерименту виконується наново. Крім того, зменшується свобода вибору згідно з формулою

$$Q(S) = \alpha n - \beta S, \tag{3.13}$$

де Q – число моделей, що переходять у нове покоління на даному ряді селекції; S – номер ряду; n - кількість вхідних змінних; $\alpha = 1 \ldots 5$ (чим більше незалежних змінних, тим менше); $\beta = 0{,}1 \ldots 0{,}2$. Формула (3.13) та пов'язана з нею процедура дають можливість швидкого розв'язання задач великої розмірності.

Значення критерію незміщеності оцінок коефіцієнтів моделі розраховується за формулою:

$$KH2_k = \frac{\sum_{i=1}^{p}\chi(a_i = b_i)\cdot a_i^2}{\sum_{i=1}^{p} a_i^2 + \sum_{i=1}^{p} b_i^2} \to \max \tag{3.14}$$

де p – загальне число моделей, a_i – коефіцієнти i-тої моделі, отримані до зміни послідовностей; b_i – коефіцієнти тієї ж моделі, отримані після зміни; $\chi(a_i = b_i)$ – відносна доля співпадінь.

До наступного ряду селекції (в наступне покоління) пропускають Q за (3.13) моделей, що мають більше значення $KH2$.

Як і для $KH1$, розрахунки середнього значення залишених моделей для формування умови загальної зупинки алгоритму справедливі. Селекція продовжується доти, доки середнє значення $\overline{KH2}_s$ у ряді селекції збільшується.

Критерій незміщеності $KH2$ необхідно застосовувати або разом з критерієм регулярності, або в алгоритмах з повним перебором моделей. При відтинанні частини моделей він швидко сходиться до локального оптимуму – найближчої задовільної моделі.

Критерій відносної незміщеності. У цьому випадку використовують лише лінійні часткові описи (наприклад, $y = a_0 + a_1 x_i + a_2 x_j$). Але щоб уникнути втрати точності, простір початкових аргументів включає також коваріації (наприклад, $x_1^2,\ x_2^2,\ x_1 x_2$). Присвоюючи значення коваріацій новим змінним, отримаємо узагальнені аргументи x_i, кількість яких може значно переважати кількість стовпців у початковій таблиці.

Оскільки частковий опис на другому ряді має вигляд $z = a_0 + a_1 y_i + a_2 y_j$, то ортогоналізований частковий опис матиме вигляд

$$z = y_i + A\hat{y}_j \tag{3.15}$$

де $\hat{y}_j$ – вектор, ортогоналізований по відношенню до x_i.

Якщо у часткових описах значення змінних центровані та нормовані за середнім значенням, то в ортогоналізованих часткових описах вільний член $a_0 = 0$, а інші коефіцієнти мають такі значення:

$$\begin{cases} a_1 = 1; \\ a_2 = A = \frac{\sum_{i=1}^{n}(y_i y_j)}{\sum_{i=1}^{N} y_i^2} \end{cases} \tag{3.16}$$

Ортогоналізація – це перетворення $\hat{y}_j$ по відношенню до базового y_i, в результаті якого одержуємо $\sum_{i=1}^{n}\left(y_i \hat{y}_j\right) = 0$. Для цього достатньо знайти такі модельні значення $\hat{y}_j = y_j - Ay_i$, де A визначається за (3.16).

Таким чином, алгоритм МГУА з використанням критерію відносної незміщеності має такі базові кроки:

Крок 1. Початкові дані ділимо на дві частини (описано раніше).

Крок 2. На першій послідовності визначаємо значення коефіцієнтів A' в рівнянні регресії, на іншій - коефіцієнти A''. Кращими вважаємо ті описи, у яких

$$KH3_k = \frac{A_k' - A_k''}{A_k} \to \min_k \tag{3.17}$$

Ряди селекції при цьому матимуть вигляд: $y = a_1 x_i$; $z = y_i + A\hat{y}_j$; $t = z_i + A\hat{z}_j$ і так далі до зупинки за глобальним критерієм.

Критерій балансу змінних

Критерій балансу змінних є найефективнішим при екстраполяції, тобто застосуванні МГУА у середньостроковому та довгостроковому прогнозуванні. Його визначення може виконуватись як емпірично, так і штучно.

При емпіричному визначенні критерію балансу змінних із подальшого розгляду на даному ряді селекції виключаються ті моделі, які дають заздалегідь невірні результати, що лежать за межами можливих значень прогнозованої залежності. Крім того, на значення прогностичних моделей можуть накладатися додаткові умови, наприклад, $\hat{y}_i = f(x_i, x_j) \le y_i$ (модельне значення не має перевищувати відомого з таблиці) або $\hat{y}_i = f(x_i, x_j) \in [y_{\min}; y_{\max}]$ (модельне значення не має виходити за певний діапазон).

Штучні умови балансу не є наслідком принципу фізичної реалізації моделі і визначаються дослідником. Найчастіше – це функції, що є комбінацією сум чи різниць вхідних факторів. При цьому, до переліку вхідних факторів X додаються незалежні змінні X', які зазвичай не мають фізичного сенсу.

Наприклад, для задачі побудови екстраполяційної моделі за трьома факторами x_1, x_2 та x_3 можуть бути введені наступні змінні:

$$s_1 = x_1 + x_2 + x_3;\ s_2 = x_1 + x_2 - x_3;\ \ldots\ s_6 = -x_1 - x_2 - x_3 \qquad (3.18)$$

Ці змінні разом з початковими змінними x_1, x_2 та x_3 утворюють вектор вхідних факторів. Розраховані s_i заносяться у таблицю початкових даних та надалі використовуються разом з іншими змінними.

За описаною вище методикою із застосуванням МНК намагаються отримати модель оптимальної складності $Y = F(x_1, x_2, x_3)$. При цьому, на кожному кроці моделі оцінюють не за звичними критеріями незміщенності чи регулярності, а наступним чином.

Припустимо, що на певному ряду отримані сімейство моделей вигляду

$$\hat{y}^k = f(X) = a_0^k + \sum_{i=1}^{m} a_i^k \cdot \prod_{j=1}^{3} x_j^{b_k},\ k = \overline{1, p}. \qquad (3.19)$$

Найкращою з усіх p моделей буде вважатися така, яка матиме мінімальне розсіювання на інтервалі спостереження

$$\Phi = \sum_{i=1}^{6} E_i^2 = \sum_{i=1}^{6}\sum_{t=1}^{N} \left(S_{it} - \hat{S}_{it}\right) = \sum_{i=1}^{6}\sum_{t=1}^{N} \left(S_{it} - \hat{S}^k(x_1, x_2, x_3, A, B, m)\right) \qquad (3.20)$$

Таким чином, за всіма табличними значеннями відомих X ми обчислюємо найкращу модель для прогнозування Y, обчислюючи на тому ж інтервалі за допоміжними змінними якість моделі.

Вигляд критерію (3.20) та необхідність створення й обчислення додаткових векторів даних вказують на очевидну трудомісткість застосування даного критерію на практиці. Тому при його використанні необхідно застосовувати

процедури зменшення кількості комбінацій перебору, у тому числі й враховуючи принципи фізичної реалізації та здорового глузду. Кількість додаткових змінних обирається дослідником і має знаходитися в межах здорового глузду та можливостей обчислювальної машини.

Слід пам’ятати, що критерій балансу змінних за математичною сутністю дуже близький (а в певних задачах ідентичний) критерію мінімуму незміщеності.

Алгоритм поділу початкової вибірки даних

Реалізація МГУА, як ми показали раніше, пов'язана із необхідністю поділу генеральної сукупності даних на дві вибірки – навчальну та контрольну. Найбільш поширеним, проте не єдиним, є підхід, за яким до навчальної послідовності обирають точки експериментів з більшим значенням дисперсії, а до контрольної – з меншим. Це пояснюється тим, що область навчання повинна бути якнайширшою, а контрольні точки, в більшості своїй, знаходитися всередині неї.

Для практичної реалізації пропонується наступний варіант алгоритму:

Крок 1. Визначити відсоткове співвідношення між кількістю елементів у навчальній і контрольній послідовності (від 70% / 30% до 50% / 50%, пов’язано з критерієм ефективності, що буде застосовуватись).

Крок 2. Для кожного стовпчика X_i, $i = \overline{1,n}$ розрахувати середнє значення його елементів

$$\bar{x}_i = \frac{1}{N}\sum_{j=1}^{N} x_{ji}\ , \tag{3.21}$$

та отримати середнє значення множини образів ($\bar{x}_1, \bar{x}_2, ..., \bar{x}_n$).

Крок 3. Знайти вибіркові дисперсії для кожного рядка таблиці за формулою

$$D_i = \frac{1}{n-1}\sum_{j=1}^{n}\left(x_{ij} - \bar{x}_j\right)^2\ ,\ i = \overline{1,N} \tag{3.22}$$

Крок 4. Для впорядковування таблиці переставити рядки таким чином, щоб першим був рядок з найбільшим значенням дисперсії, а останнім - з найменшим.

Крок 5. У відповідності до результату кроку 4 розділити дані в таблиці на навчальну та контрольну послідовності.

Якщо розв'язується задача короткострокового прогнозу або інтерполяція всередині інтервалу наявних даних, тоді додатково шукають оптимальне співвідношення кількості образів у навчальній та контрольній послідовностях з метою отримання найпростішої та достовірної моделі.

3.2. Приклад розв’язування задачі

Модель GMDH – це тип штучної нейронної мережі, який можна використовувати для завдань регресії.

1. Завантаження та встановлення пакету GmdhPy

Однією з реалізацій методу групового урахування аргументів є GmdhPy, доступний за посиланням. Для встановлення необхідно скачати пакет на комп'ютер та в консолі в директорії із завантаженим пакетом запустити команду:

pip install ./GmdhPy-2.0-py2.py3-none-any.whl

2. Імпорт необхідних модулів

Першим кроком є імпорт необхідних бібліотек.

```
import numpy as np
from gmdhpy.gmdh import Regressor
from sklearn.metrics import mean_squared_error
```

3. Створення штучного набору даних

Далі нам потрібно визначити наші вхідні масиви. X це двовимірний масив, де кожен рядок представляє вхідний зразок, а кожен стовпець представляє функцію. y це 1D масив, де кожен елемент представляє відповідне вихідне значення.

```
X = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
y = np.array([10, 20, 30])
```

4. Створення регресорної модель GMDH

Тепер ми створюємо екземпляр класу Regressor, передаючи список довідкових функцій (ref_functions) як аргумент. У цьому прикладі ми використовуємо ['linear', 'cubic'] як довідкові функції.

```
model = Regressor(ref_functions=['linear', 'cubic'])
```

Всі види опорних функцій:

linear	Лінійна
linear_cov	Повна 1-го порядку
quadratic	Повна 2-го порядку
cubic	Повна 3-го порядку

5. Підгонка моделі до вхідних даних

Далі ми підганяємо модель до вхідних даних за допомогою методу fit().

```
model.fit(X, y)
```

6. Прогнози за допомогою навченої моделі

Тепер, коли модель навчена, ми можемо робити прогнози за допомогою методу predict().

```
predict_y = model.predict(X)
```

7. Обчисліть середньо квадратичної помилки

Нарешті, ми обчислюємо середню квадратичну помилку між прогнозованими значеннями (predict_y) і фактичними вихідними значеннями (y) за допомогою функції mean_squared_error(). Потім ми друкуємо розраховане значення помилки.

```
mse = mean_squared_error(y, predict_y)
print('Error:')
print(mse)
```

3.3. Індивідуальне завдання

1. Завантажити пакет, що містить реалізацію методу групового врахування аргументів.
2. Завантажити дані *lab3_vars.xlsx*. Номер цільової змінної співпадає з останньою цифрою студента в списку групи.
3. Випадково перемішати вибірку та розбити її на тренувальну та тестову у співвідношенні 80/20.
4. Навчити регресійну модель МГУА з різними типами опорних функцій (див. табл. 3.1 із завданням варіантів). Порівняти середньоквадратичну помилку на тренувальній та тестовій вибірках, обрати кращу модель. В стовпчику «модель 2» вказано одразу декілька типів опорних функцій. Їх необхідно подати разом в одну модель.
5. Побудувати діаграми розсіювання, де зобразити початкові дані та передбачені. Окремі діаграми необхідно побудувати для тренувальних та тестових даних.
6. У висновках зазначити кращу модель, оцінити чи є вона адекватною.

Початкові дані знаходяться у файлі *lab3_vars.xls* в робочому каталогу навчальної дисципліни на сервері Moodle. Набір даних складається з трьох вхідних змінних і однієї цільової, що обирається за номером варіанту. Останній співпадає з останньою цифрою студента в списку групи.

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, розрахунки, необхідні описи та графіки (відповідно до завдання вище) та вихідний код програм.

Назва файлу: Прізвище_група

Роботи, виконані не за варіантом, не приймаються.

Табл. 3.1. – Індивідуальні варіанти завдання.

№	Модель 1	Модель 2
1	Лінійна	Повні 1-го та 2-го порядків
2	Повна 1-го порядку	Лінійна та повна 3-го порядку
3	Повна 2-го порядку	Повні 1-го та 3-го порядків
4	Повна 3-го порядку	Повні 1-го та 2-го порядків
5	Лінійна	Лінійна та повна 2-го порядку
6	Повна 1-го порядку	Повні 1-го та 2-го порядків
7	Повна 2-го порядку	Лінійна та повна 3-го порядку
8	Повна 3-го порядку	Лінійна та повна 1-го порядку
9	Лінійна	Повні 1-го та 3-го порядків
10	Повна 1-го порядку	Лінійна та повна 2-го порядку
11	Повна 2-го порядку	Повні 1-го та 2-го порядків
12	Повна 3-го порядку	Лінійна та повна 2-го порядку

3.4. Контрольні питання

1. До якого класу методів належить метод групового врахування аргументів?
2. Які принципи покладені в основу реалізації МГУА?
3. Як у МГУА використовується принцип неповноти Геделя?
4. В чому полягає принцип свободи вибору при визначенні множини моделей?
5. Дайте характеристику принципу повноти у МГУА.
6. Які варіації має критерій регулярності?
7. Якими можуть бути критерії закінчення ітераційного процесу у МГУА?
8. Опишіть сутність критерію незміщеності, що базується на аналізі розв'язків,
9. Яка роль навчальної та контрольної послідовностей в алгоритмах МГУА?
10. Опишіть критерій незміщеності, що базується на аналізі коефіцієнтів.
11. Які особливості застосування критерію відносної незміщеності?
12. Наведіть приклади емпіричного та штучного формування критерію балансу змінних.
13. До розв'язання яких задач доцільно застосовувати критерій балансу змінних?
14. Наведіть алгоритм поділу генеральної сукупності на навчальну та контрольну послідовності за величиною дисперсії.
15. Яким чином в алгоритмах МГУА реалізовано елементи самоорганізації?
16. Опишіть особливості вибору та застосування опорних функцій.
17. У чому полягає реалізація принципу зовнішнього доповнення?

Практична робота №4 – Згорткова нейронна мережа для розпізнавання зображень

Мета роботи: закріплення теоретичних знань та розвинення навичок роботи із типовими архітектурами згорткових нейронної мереж.

Практична робота присвячена знайомству із задачами комп'ютерного зору та розробці зготкової нейронної мережі, що може вирішити задачу класифікації зображень. За узгодженням з викладачем студент може запропонувати свій набір даних для розв'язання задачі практичної роботи. Допускається виконання роботи мовою програмування MATLAB.

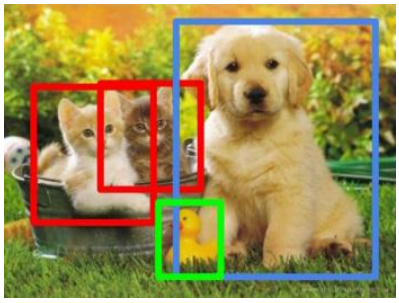
4.1. Теоретичні відомості

Згорткові нейронні мережі найчастіше використовують для задач комп'ютерного зору. Основними задачами є:

1. Класифікація – для кожного вхідного зображення необхідно зазначити клас, до якого дане зображення належить. Класи зображення є деякою фіксованою наперед заданою множиною. Приклад задачі класифікації зображено на рис. 4.1а.
2. Пошук об'єктів (детекція) – для кожного вхідного зображення необхідно: 1) виділити об'єкти наявні на зображенні прямокутною рамкою; 2) кожному знайденому об'єкту зазначити клас до якого він належить. Див. рис. 4.1б.
3. Сегментація – для кожного вхідного зображення необхідно знайти об'єкти наявні на зображенні, виділити дані об'єкти якомога точніше маскою довільної форми та зазначити клас кожного об'єкту. Зображено на рис. 4.1в.

кіт

(а)



кіт, собака, качка

(б)

кіт, собака, качка

(в)

Рис. 4.1. Види задач комп'ютерного зору: (а) – задача класифікації, (б) – задача пошуку об'єктів, (в) – задача сегментації.

Типова нейронна мережа складається з наступних операцій:

1. Шар згортки – вводить вхідні зображення через набір згорткових фільтрів, кожен з яких активує певні функції із зображень.

2. Активаційна функція, найчастіше використовується ReLU – дозволяє прискорити і зробити ефективнішим навчання.
3. Пулінг або операція субдискретизації – спрощує виведення, виконуючи нелінійне зменшення простору, зменшуючи кількість параметрів, які потрібно ідентифікувати мережі.
4. Повнозв'язний шар – «вирівнює» двовимірні просторові функції мережі в вектор ознак для цілей класифікації.
5. Шар Softmax розподіляє ймовірності для кожної категорії в наборі даних.

Порядок застосування зазначених вище функцій має значення. Деякі шари можуть повторюватись. Найтиповішою є наступна структура нейронної мережі:

1. Шар згортки, активаційна функція, пулінг – даний набір операцій і саме в такому порядку може бути повторюваний один чи декілька разів. Чим більше таких шарів, тим більш глибокою стає нейронна мережа.
2. Далі йде повнозв'язний шар – цей шар перетворює ознаки, отримані на виході з попереднього кроку до вигляду, що є зручним для задачі класифікації.
3. Останнім шаром йде softmax – задача цього шару перетворити вихід повнозв'язного шару на розподілення ймовірностей класів.

Далі розглядається визначення кожного з цих компонентів на прикладі класифікації рукописної цифри (рис. 4.2.).

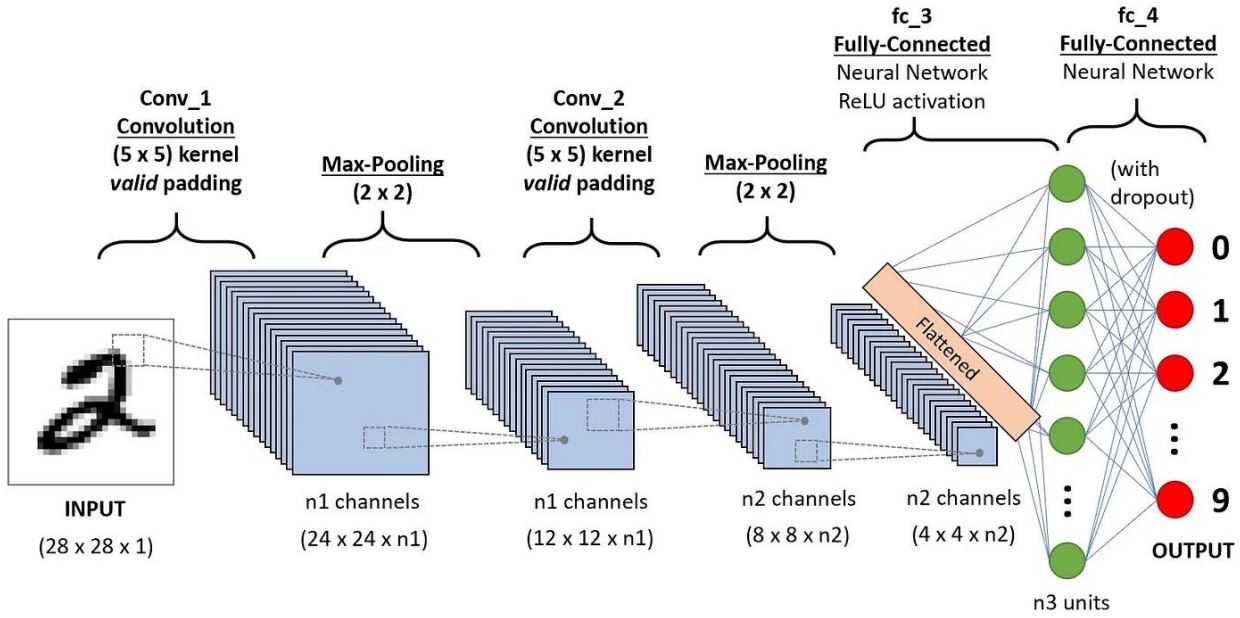


Рисунок 4.2 – Архітектура ЗНМ, застосована до розпізнавання цифр.

Шари згортки

Це перший будівельний блок ЗНМ. Як випливає з назви, основна математична задача, яка виконується, називається згорткою, яка є застосуванням функції ковзного вікна до матриці пікселів, що представляє зображення. Ковзна

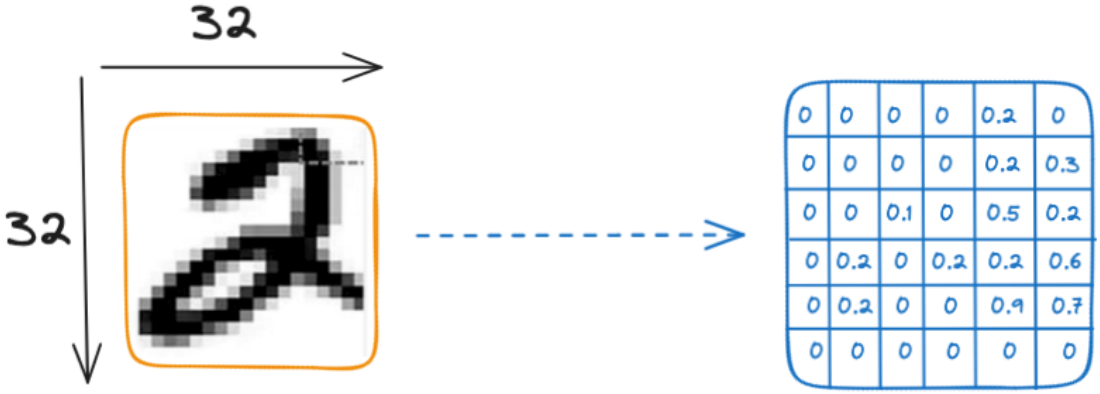
функція, застосована до матриці, називається ядром або фільтром – назви, що можуть використовуватися як взаємозамінні.

У шарі згортки застосовуються кілька фільтрів однакового розміру, і кожен фільтр використовується для розпізнавання певного шаблону із зображення, наприклад викривлення цифр, країв, усієї форми цифр тощо.

Простіше кажучи, у шарі згортки ми використовуємо невеликі сітки (так звані фільтри або ядра), які переміщаються по зображенню. Кожна маленька сітка схожа на міні-збільшувальне скло, яке шукає на фотографії певні візерунки, як-от лінії, криві чи форми. Переміщаючись по фотографії, згортка створює нову сітку, яка виділяє, де знайшла шукані візерунки.

Наприклад, один фільтр може добре знаходити прямі лінії, інший – криві тощо. Використовуючи кілька різних фільтрів, ЗНМ може отримати гарне уявлення про всі різні шаблони, які складають зображення.

Давайте розглянемо зображення рукописної цифри (рис. 4.3) в градаціях сірого розміром 32×32. Значення в матриці наведені для ілюстрації.



(а) (б)

Рисунок 4.3 – Ілюстрація вхідного зображення та його піксельне представлення. (а) – вхідне зображення, (б) – чисельне представлення пікселів.

Двовимірна згортка (2D convolution) – це досить проста операція: починаємо з ядра, що представляє собою матрицю ваг. Ядро «ковзає» над двовимірним зображенням, поелементно виконуючи операцію множення з тією частиною вхідних даних, над якою воно зараз знаходиться, а потім підсумовує всі отримані значення в один вихідний піксель.

Ядро повторює цю процедуру з кожною локацією (рис. 4.5), над якою воно «ковзає», перетворюючи двовимірну матрицю на іншу двовимірну матрицю ознак. Ознаки на виході є зваженими сумами (де ваги є значеннями самого ядра) ознак на вході, розташованих приблизно в тому ж місці, що вихідний піксель на вхідному шарі.

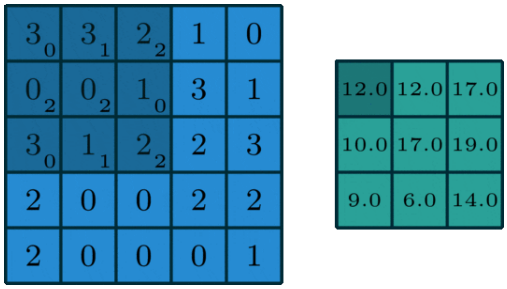


Рисунок 4.5 – Операція згортки.

Розмір ядра згорткової нейронної мережі визначає кількість ознак, які будуть об'єднані для отримання нової ознаки на виході.

У наведеному вище прикладі ми маємо $5 \times 5 = 25$ ознак на вході і $3 \times 3 = 9$ ознак на виході. Для повнозв’язного шару ми мали б вагову матрицю $25 \cdot 9 = 225$ параметрів, а кожна вихідна ознака була би зваженою сумою всіх ознак на вході. Згортка дозволяє зробити таку операцію з усього 9-ма параметрами, адже кожна ознака на виході є результатом аналізу не кожної ознаки на вході, а тільки однієї вхідної, що знаходиться в «приблизно тому ж місці». Зверніть на це увагу, оскільки це буде важливим для подальшого обговорення.

Згортка, в цілому, все ще є лінійним перетворенням, але в той же час вона також є зовсім іншим видом перетворення. Для матриці з 64 елементами існує лише 9 параметрів, які повторно використовуються кілька разів. Кожен вихідний вузол отримує лише певну кількість входів (ті, що знаходяться всередині ядра). Немає жодної взаємодії з іншими входами, тому що вага для них дорівнює 0.

Локальні особливості. Отже:

- Ядра поєднують пікселі лише з невеликої локальної області для формування виходу. Тобто вихідні ознаки бачать лише вхідні ознаки із невеликої локальної області;
- Ядро застосовується глобально по всьому зображення для створення матриці вихідних значень.

Таким чином, з backpropagation (метод зворотного розповсюдження помилки), що йде у всіх напрямках від вузлів класифікації мережі, ядра мають цікаву задачу вивчення ваг для створення ознак тільки з локального набору входів. Крім того, оскільки саме ядро застосовується по всьому зображенню, ознаки, які вивчає ядро, повинні бути достатньо спільними, щоб надходити з будь-якої частини зображення.

Якщо це були якісь інші дані, наприклад, дані про установки додатків за категоріями, то це стало б катастрофою, тому що наявність стовпців кількості установок додатків та типів додатків поруч не означає, що вони мають «локальні загальні ознаки», загальні з датами встановлення програм та часом використання. Звичайно, у кількох можуть бути основні ознаки вищого рівня, які можуть бути знайдені, але це не дає нам жодних підстав вважати, що параметри для перших двох такі самі, як параметри для останніх двох.

Пікселі, однак, завжди відображаються в послідовному порядку, а сусідні пікселі впливають на піксель поруч. Наприклад, якщо всі сусідні пікселі червоні, досить ймовірно, що піксель поряд також червоний. Якщо є відхилення, це цікава аномалія, яка може бути перетворена на ознаку і все це можна виявити при порівнянні пікселя зі своїми сусідами, з іншими пікселями у своїй місцевості.

Ця ідея – те, на чому були засновані більш ранні методи отримання ознак комп'ютерним зором. Наприклад, для виявлення граней можна використовувати фільтр виявлення граней Собеля – ядро з фіксованими параметрами, що діє так само, як стандартна одноканальна згортка (рис. 4.6).

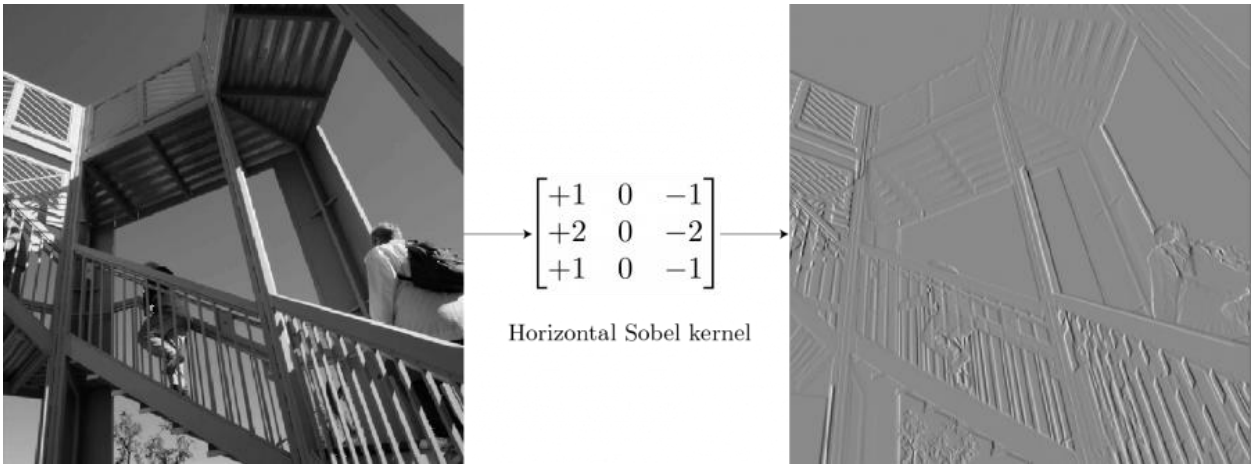


Рис. 4.6 – Застосування ядра, що визначає межі.

Для сітки, що не містить граней (наприклад, неба на задньому фоні), більшість пікселів мають однакове значення, тому загальний висновок ядра в цій точці дорівнює 0. Для сітки з вертикальними гранями існує різниця між пікселями зліва і праворуч від грані, і ядро обчислює цю ненульову різницю, знаходячи ребра. Ядро за раз працює лише з сітками 3×3, виявляючи аномалії в певних місцях, але застосування до всього зображення достатньо для виявлення певної ознаки в будь-якому місці зображення!

Використовуючи вхідну матрицю та матрицю ядра, ми можемо виконати операцію згортки, застосовуючи скалярний добуток і працювати наступним чином (рис. 4.7):

1. Застосувати матрицю ядра від верхнього лівого кута до правого.
2. Виконати поелементно множення.
3. Просумувати значення добутків.
4. Отримане значення відповідає першому значенню (верхній лівий кут) у згорнутій матриці.
5. Перемістити ядро вниз відносно розміру ковзного вікна.
6. Повторювати кроки з 1 по 5, доки матриця зображення не буде повністю покрита.

Розмір згорнутої матриці залежить від розміру ковзного вікна. Чим вище в ковзне вікно, тим менше кінцева розмірність.

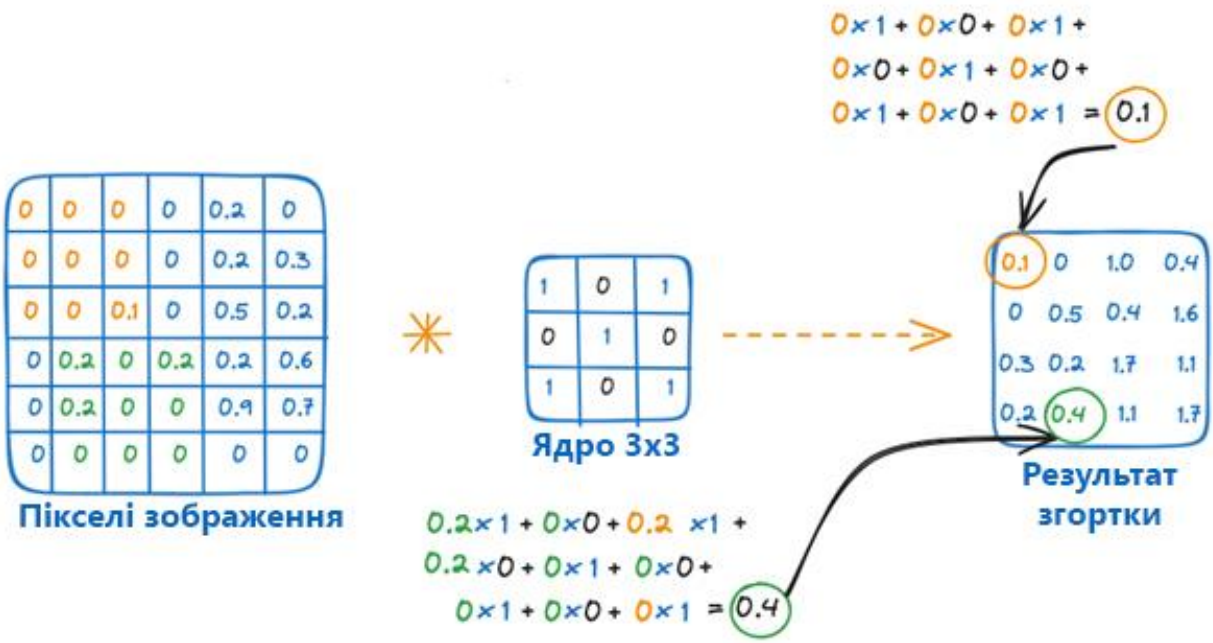


Рисунок 4.7 – Застосування завдання згортки з кроком 1 із ядром 3 × 3.

Функція активації

Функція активації ReLU (рис. 4.8) застосовується після кожної операції згортки. Ця функція допомагає мережі вивчати нелінійні зв'язки між об'єктами на зображенні, таким чином роблячи мережу більш надійною для визначення різних шаблонів. Це також допомагає пом'якшити проблеми зникнення градієнта. $\mathrm{ReLU}(x) = \max(x, 0)$.

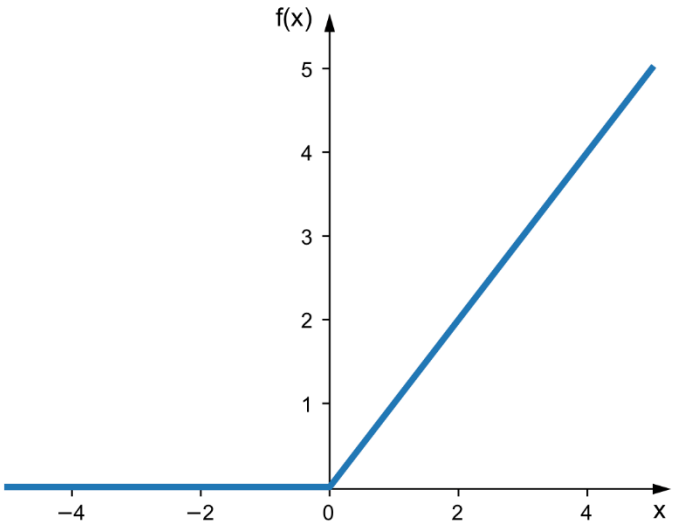


Рисунок 4.8 – Функція активації ReLU.

Шар субдискретизації (Pooling)

Метою шару субдискретизації є вилучення найбільш значущих характеристик зі складної матриці. Це робиться шляхом застосування деяких операцій агрегації, які зменшують розмірність карти ознак (згорнутої матриці),

отже, зменшуючи пам’ять, що використовується під час навчання мережі. Об’єднання також актуальне для пом’якшення перенавчання.

Найпоширеніші функції агрегації, які можна застосувати:

- Субдискретизація за максимумом, що є максимальним значенням карти ознак
- Субдискретизація за середнім – це середнє значення в ковзному вікні.

На рис. 4.9 наведено ілюстрацію субдискретизації за максимумом.

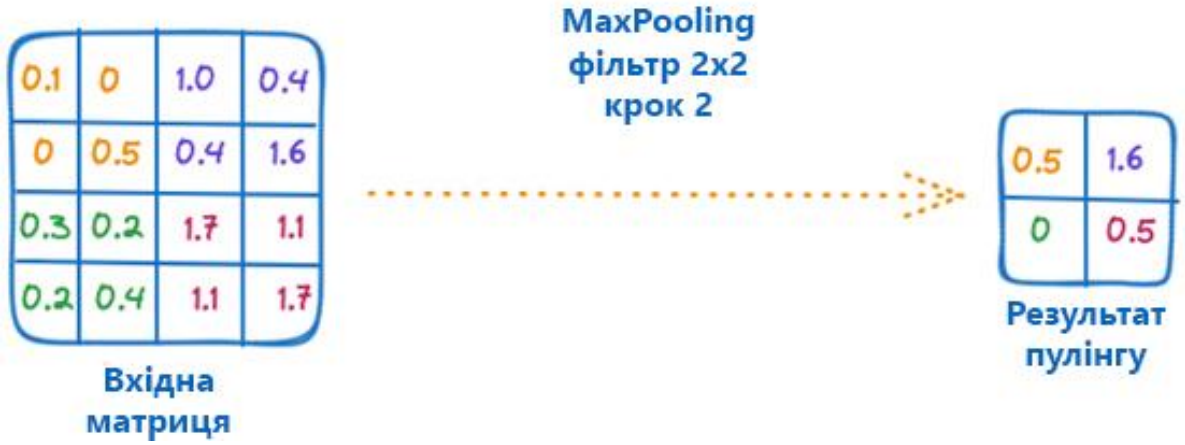


Рисунок 4.9 – Застосування субдискретизації за максимумом з кроком 2 за допомогою фільтра 2×2.

Існує 2 підходи до навчання глибоких нейронних мереж:

1. Навчання мережі з нуля – найдовший, але найкраще пристосований для певної задачі.
2. Використання трансферного навчання для навчання існуючої мережі – вже навчену мережу, що розпізнає певні класи, навчають додатково на прикладах нових класів.

Дану практичну роботу буде присвячено зображенням, як найбільш типовому об’єкту згорткових нейронних мереж. Але глибоке навчання стає все більш популярними і для інших задач: обробки звуків, зокрема мови, генерації текстів чи зображень і навіть відео. Принципи глибокого навчання також можуть бути застосовані до будь-яких сигналів даних у задачах регресії, прогнозування, пошуку асоціацій та інших.

4.2. Приклад розв’язування задачі

Для роботи із зображеннями використаємо пакет для роботи з нейронними мережами Keras, що вже було встановлено в практичній роботі №1. Приклад цифр з набору даних показано на рис. 4.10.

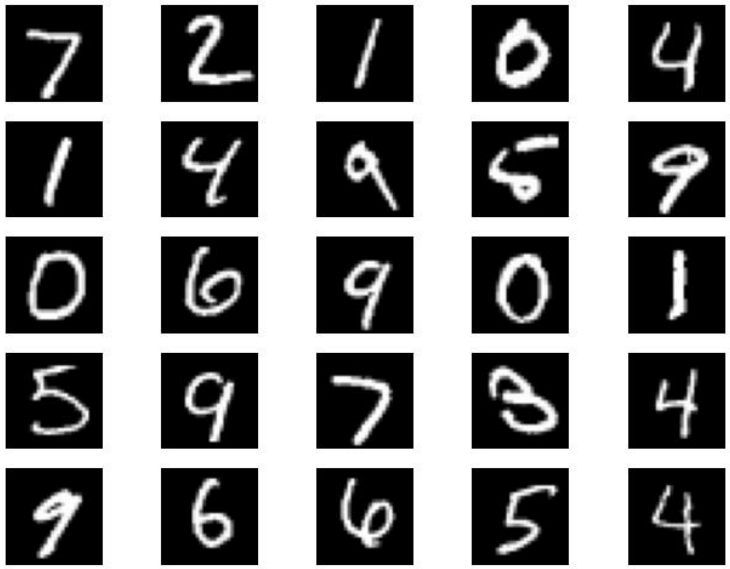


Рис. 4.10. – Приклади рукописних цифр з набору даних MNIST.

1. Імпорт необхідних модулів

Першим кроком є імпорт необхідних бібліотек.

```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
import matplotlib.pyplot as plt
```

2. Завантаження набору даних

В якості початкових даних в використаємо набір даних рукописних цифр MNIST, який містить 60 000 зображень рукописних цифр 0-9 та 10 000 тестових зображень.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

3. Первинна обробка даних

Для роботи із зображеннями в пакеті Keras їх необхідно попередньо обробити, зокрема значення пікселів звести в діапазон [0; 1], та вказати кількість категорій зображень.

```
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)

X_train = X_train.astype('float32') / 255
```

```
X_test = X_test.astype('float32') / 255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

4. **Покажемо декілька цифр з набору даних MNIST**

```
plt.figure()
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(X_test[i], cmap='gray')
plt.show()
```

Зауважимо, що вікно із зображення цифр необхідно закрити для продовження навчання.

5. **Створимо архітектуру нейронної мережі**

```
model = Sequential()
model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(28,
28, 1)))
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

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

6. **Проведемо навчання нейронної мережі**

```
history = model.fit(X_train, y_train, batch_size=128, epochs=1,
                    validation_data=(X_test, y_test))
```

7. **Створимо графік залежності точності на навчальній та тестовій вибірках в залежності від епохи**

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.legend(['Train', 'Test'])
plt.show()
```

8. **Виведемо оцінку точності навченої мережі на тестовому наборі даних**

```
loss, accuracy = model.evaluate(X_test, y_test)
print('Test accuracy:')
print(accuracy)
```

4.3. Індивідуальне завдання

1. Навчити нейронну мережу, використовуючи архітектуру в прикладі.
2. Модифікувати архітектуру мережі відповідно до власного варіанту (табл. 4.1). В таблиці вказано лише блоки, що необхідно змінити.
3. Наведіть графік точності навчання нової архітектури. Визначте точність навчання на тестовій вибірці. Порівняйте якість отриманої архітектури із початковою.
4. Визначити точність на навчальний вибірці та порівняти її з тестовою точністю.
5. Проаналізуйте як параметри згорток та шарів пулінгу впливають на якість мережі. Наведіть 2 графіки залежності точності від певних параметрів (на вибір студента) та визначте кращу архітектуру нейронної мережі.

Табл. 4.1. – Варіанти індивідуальних завдань.

Варіант	Архітектура	Епох
1	Conv2D(16, 3), Conv2D(32, 3), MaxPooling2D(2), …, Dense(128), …	2
2	Conv2D(24, 5), Conv2D(48, 5), MaxPooling2D(3), …, Dense(160), …	3
3	Conv2D(40, 3), Conv2D(80, 3), MaxPooling2D(2), …, Dense(192), …	4
4	Conv2D(32, 5), Conv2D(64, 5), MaxPooling2D(2), …, Dense(128), …	2
5	Conv2D(20, 3), Conv2D(40, 3), MaxPooling2D(2), …, Dense(100), …	3
6	Conv2D(48, 4), Conv2D(96, 4), MaxPooling2D(3), …, Dense(192), …	4
7	Conv2D(36, 3), Conv2D(72, 3), MaxPooling2D(2), …, Dense(144), …	2
8	Conv2D(28, 5), Conv2D(56, 5), MaxPooling2D(2), …, Dense(112), …	3
9	Conv2D(50, 5), Conv2D(100, 5), MaxPooling2D(2), …, Dense(200), …	4
10	Conv2D(34, 4), Conv2D(68, 4), MaxPooling2D(3), …, Dense(136), …	2
11	Conv2D(26, 3), Conv2D(52, 3), MaxPooling2D(2), …, Dense(104), …	3
12	Conv2D(42, 5), Conv2D(84, 5), MaxPooling2D(2), …, Dense(168), …	4
13	Conv2D(30, 4), Conv2D(60, 4), MaxPooling2D(3), …, Dense(120), …	2
14	Conv2D(38, 5), Conv2D(76, 5), MaxPooling2D(2), …, Dense(152), …	3
15	Conv2D(44, 4), Conv2D(88, 4), MaxPooling2D(3), …, Dense(176), …	4
16	Conv2D(22, 3), Conv2D(44, 3), MaxPooling2D(2), …, Dense(96), …	2
17	Conv2D(46, 5), Conv2D(92, 5), MaxPooling2D(2), …, Dense(184), …	3
18	Conv2D(32, 4), Conv2D(64, 4), MaxPooling2D(3), …, Dense(128), …	4

Варіант	Архітектура	Епох
19	Conv2D(40, 5), Conv2D(80, 5), MaxPooling2D(2), …, Dense(160), …	2
20	Conv2D(36, 4), Conv2D(72, 4), MaxPooling2D(3), …, Dense(144), …	3
21	Conv2D(48, 5), Conv2D(96, 5), MaxPooling2D(2), …, Dense(192), …	4
22	Conv2D(30, 5), Conv2D(60, 5), MaxPooling2D(2), …, Dense(120), …	2
23	Conv2D(42, 4), Conv2D(84, 4), MaxPooling2D(3), …, Dense(168), …	3
24	Conv2D(38, 4), Conv2D(76, 4), MaxPooling2D(3), …, Dense(152), …	4
25	Conv2D(44, 5), Conv2D(88, 5), MaxPooling2D(2), …, Dense(176), …	2

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, розрахунки, необхідні описи та графіки (відповідно до завдання вище) та вихідний код програм.

Назва файлу: Прізвище_група

Роботи, виконані не за варіантом, не приймаються.

4.4. Контрольні питання

1. Які основні задачі, що можна вирішити згортковими нейронними мережами?
2. Поясніть, що таке задача сегментації?
3. Як сегментація відрізняється від задачі класифікації?
4. Що таке задача пошуку об’єктів?
5. Яку іншу назву має задача пошуку об’єктів?
6. Якими є основні види шарів нейронної мережі?
7. Для чого використовується шар згортки?
8. За допомогою яких саме шарів збільшують глибину згорткової нейронної мережі?
9. Яку іншу назву шару пулінгу ви знаєте?
10. Які параметри необхідно вказати при навчанні нейронної мережі?
11. Для чого необхідно розбивати всю доступну вибірку на навчальну та тестову?
12. Що таке матриця помилок? Як вона дозволяє проаналізувати якість навчання мережі?
13. Для чого використовується шар пулінгу?
14. Чим цікавий набір даних MNIST? Що він містить?
15. Яким чином слід оцінити фінальну якість нейронної мережі?

Практична робота №5 – Класифікація текстових документів за допомогою рекурентної мережі

Мета роботи: закріплення теоретичних знань та розвинення навичок побудови рекурентної мережі із короткою, довгою пам'яттю. Обробка та класифікація текстів.

Практична робота присвячена знайомству із структурними блоками нейронних мереж спрямованих на обробку, класифікацію та генерацію текстів, набуттю вміння оброблювати текст, будувати та навчати такі нейронні мережі. Робота виконується із використанням мови програмування Python та бібліотек Keras та TensorFlow. За узгодженням з викладачем студент може запропонувати свій набір даних для розв'язання задачі практичної роботи. Допускається виконання роботи мовою програмування MATLAB.

5.1. Теоретичні відомості

В попередній практичній роботі ми навчились передбачувати клас зображення за самим зображенням. Наше передбачення залежало лише від даного конкретного зображення x_i. В той самий час, є цілих клас задач, що вимагають передбачення для послідовності $x_1, \dots, x_T$. Наприклад, звук, відео, текст, часовий ряд. Ми казатимемо, що кожна із таких послідовностей має часову компоненту t. Чи можна обійтись без послідовності? Уявіть собі, вам дають одне слово із тексту і просять передбачити тип документу: переписка із друзями, конспект, художня література. Так, іноді можна вгадати, але загалом за одним словом таку задачу якісно вирішити неможливо. Тому і було запропоновано так звані рекурентні нейронні мережі, що можуть враховувати історію спостережень, тобто часову компоненту. Далі давайте поговоримо про типи послідовностей із якими працюють нейронні мережі.

Існує чотири типи послідовностей в залежності від кількості входів і виходів:

1. Один до одного.
2. Один до багатьох.
3. Багато до одного.
4. Багато до багатьох.

Приклад послідовності «один до одного зображено» на рис. 5.1. В даному випадку маємо звичайну нейронну мережу для класифікації табличних даних або окремих зображень без часової компоненти. З такою мережею ми вже знайомі з попередніх практичних робіт.

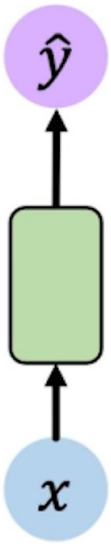


Рисунок 5.1. – Тип послідовності «один до одного».

Приклад послідовності типу «один до багатьох» зображено на рисунку 5.2. Типовою задачею із таким типом послідовності є анотація зображень: на вхід до мережі подається одне статичне зображення, мережа в свою чергу має описати зміст зображення. Така задача відрізняється від звичайної класифікації зображень, оскільки відсутній фіксований перелік класів, натомість генерується речення, що якомога точніше описує зображення.

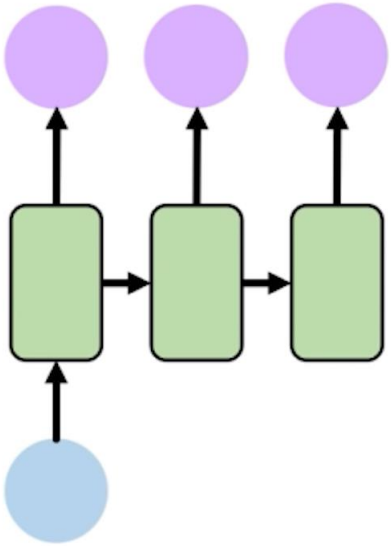


Рисунок 5.2. – Тип послідовності «один до багатьох».

Приклад послідовності типу «багато до одного» зображено на рис. 5.3. При роботі з рекурентними мережами найчастіше ви будете зустрічати саме такий тип послідовності. В даному випадку на вхід подається поелементно часовий ряд, закодований текст або інша послідовність, на виході необхідно отримати одне значення. Однією із задач є класифікація емоційного забарвлення тексту, над

якою ми будемо працювати в рамках індивідуального завдання в практичній роботі. На вхід маємо текст опису, наприклад, фільму, задача нейронні мережі визначити чи цей текст є «позитивним» або «негативним».

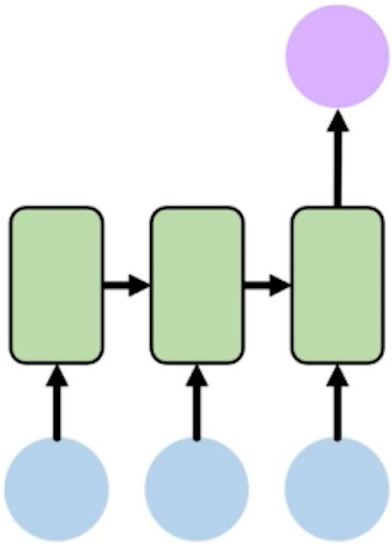


Рисунок 5.3. – Тип послідовності «багато до одного».

Приклад послідовності «багато до багатьох» зображено на рис. 5.4. В даному випадку і на вході, і на виході маємо послідовності. Одним із прикладів цієї проблеми є переклад. У перекладі ми надаємо кілька слів з однієї мови як вхідні дані та прогнозуємо кілька слів з другої мови як вихідні дані.

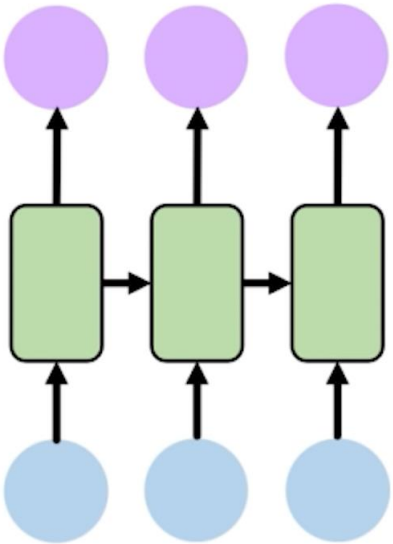


Рисунок 5.4. – Тип послідовності «багато до багатьох».

До алгоритмів обробки послідовностей ми висуваємо наступні вимоги:

1. Можливість роботи із послідовностями довільної довжини.
2. Відстеження далекої (у часі) залежності.

3. Розуміння порядку у сигналі.

Розглянемо яким чином можливо побудувати блок рекурентної нейронної мережі. Для цього повернемось до моделі повнозв'язного шару, спрощено зображеної на рис. 5.5, де $x_t \in \mathbb{R}^m$ – це деякий фіксований вектор вхідних ознак, а $\hat{y}_t \in \mathbb{R}^n$ – вектор вихідних ознак. Такий шар оброблює вхід лише за деякий конкретний момент часу t.

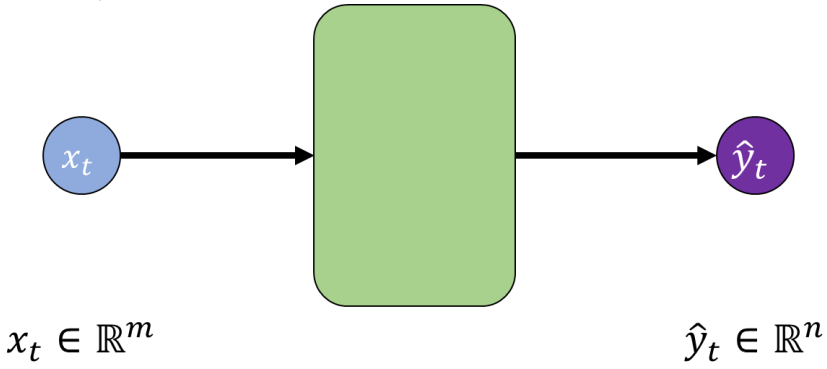


Рис. 5.5. Спрощений вигляд повнозв'язного шару.

Очевидно, що такий шар ніяким чином не враховує наявність послідовності в сигналі, та не може моделювати залежності в часі. Для виправлення зазначеного недоліку, зробимо наступні прості перетворення:

1. Розпишемо декілька повнозв'язних шарів для моментів у часі $t = 0,1,2$, як зображено на рис. 5.6.

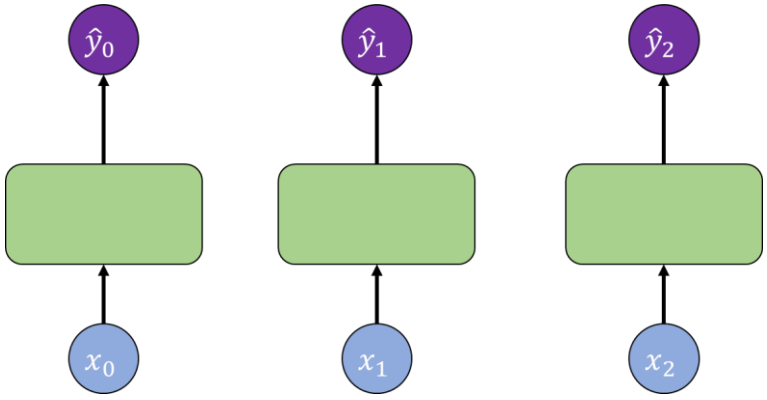


Рис. 5.6. Повнозв'язні шари для декількох моментів у часі.

2. Пов'яжемо окремі повнозв'язні шари додатковим зв'язком таким чином, щоб кожен повнозв'язний шар приймав деякий вектор із історією h_{t-1}, та видавав новий вектор із історією h_t, як показано на рис. 5.7.

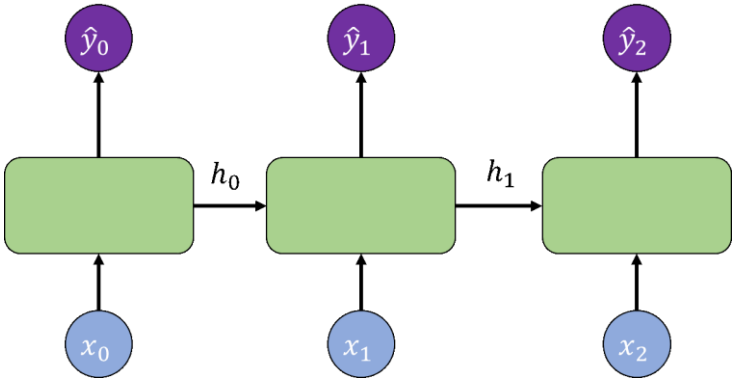


Рис. 5.7. Повнозв'язні шари із зв'язками між ними.

Отже таким чином, кожен блок буде рахувати вже 2 функції:

1. Функцію для передбачення виходу із блоку $\hat{y}_t = f(x_t, h_{t-1})$, де x_t – це елемент послідовності (слово), що подається на вхід, h_{t-1} – вектор із історією, що містить інформацію про попередні елементи послідовності, $\hat{y}_t$ – передбачення для поточного кроку t.
2. Функцію для передбачення вектору з історією для наступного блоку $h_t = g(x_t, h_{t-1})$, який також залежить від входу на поточному кроці x_t та історії h_{t-1}.

Те, що ми отримали, має назву рекурентного блоку RNN. Його зображено на рис. 5.8. Зверніть увагу на додаткові зв'язки між суміжними блоками. Такий блок вже може бути ефективно використаний для передбачення послідовностей. Важливим ускладненням даного блоку є LSTM – блок із довгою та короткою пам'яттю, що здебільшого є аналогічним до блоку RNN, але моделює окремо довгі та короткі послідовності. На практиці найчастіше використовується саме блок LSTM.

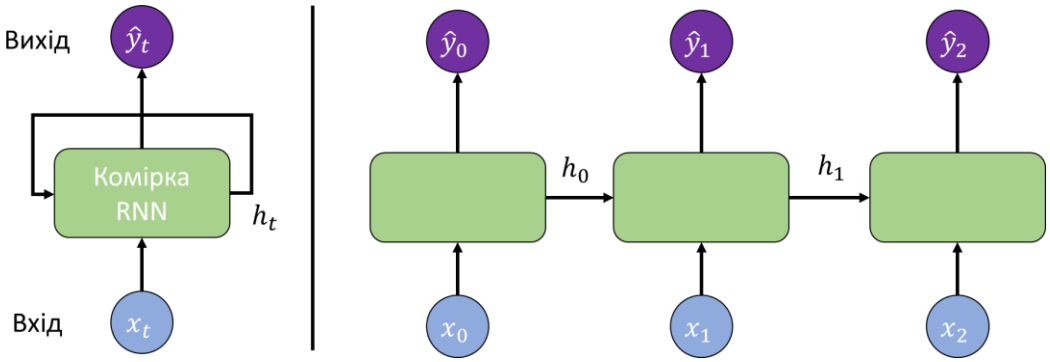


Рис. 5.8. Комірка рекурентної нейронної мережі.

Для навчання рекурентної нейронної мережі використовується алгоритм зворотного поширення в часі (BPTT).

Зворотне поширення в часі є розширенням традиційного алгоритму зворотного поширення, призначеного для роботи з рекурентними нейронними

мережами (RNN), які обробляють послідовні дані. Це метод для обчислення градієнтів у RNN, який є важливим для навчання цих моделей.

У традиційних нейронних мережах прямого зв'язку кожен шар обробляє вхідні дані лише один раз, а потім передає їх на наступний шар. Натомість, RNN зберігають внутрішній стан (також відомий як прихований стан), який фіксує інформацію з попередніх часових кроків. Це дозволяє їм моделювати часові зв'язки в даних. Однак це також означає, що помилки можуть поширюватися з часом, що ускладнює обчислення градієнтів для навчання.

Алгоритм зворотного поширення в часі.

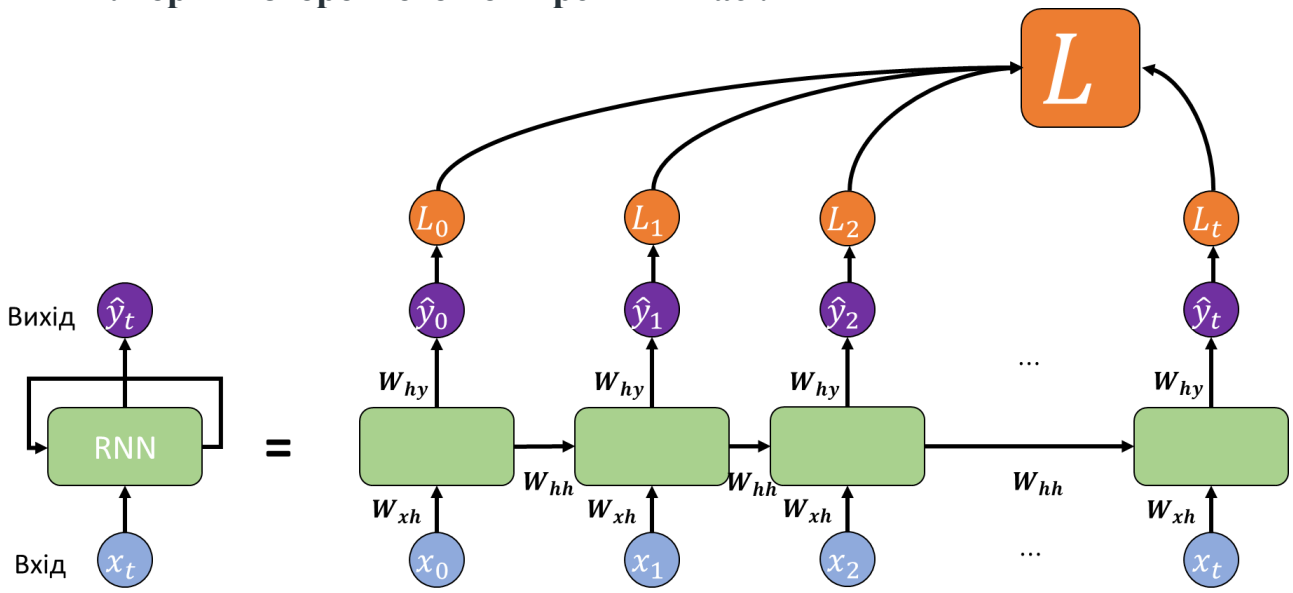


Рисунок 5.9. – Зворотне поширення в часі у RNN.

Схему алгоритму навчання рекурентної нейронної зображено на рис. 5.9. Основні кроки алгоритму складають:

1. Розгортання: розбиття RNN на окремі блоки, що відповідають різним моментам в часі.
2. Прямий прохід: обчислення виходу розгорнутої мережі, обробляючи вхідну послідовність покроково.
3. Зворотній прохід: обчислення градієнтів помилок на кожному кроці часу за допомогою правила ланцюга, а саме:
    a. Обчислення градієнту функції втрат відносно виходу на кожному часовому кроці.
    b. Зворотне поширення помилок через розгорнуту мережу, обчислення градієнтів втрат щодо параметрів моделі та вхідних послідовностей.
4. Накопичення градієнта: накопичення градієнтів, обчислених на кожному кроці часу, щоб отримати загальний градієнт для всієї послідовності.

5. Оновлення параметрів: оновлення параметрів моделі за допомогою накопичених градієнтів і алгоритму оптимізації.

5.2. Приклад розв'язування задачі

1. Імпорт необхідних модулів

```
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
```

2. Завантаження набору даних

Використаємо набір IMDB оглядів фільмів. Це набір даних для двійкової класифікації емоційного забарвлення відгуків на фільми та містить 50 тисяч прикладів, розмічених як «позитивний» або «негативний».

Приклади відгуків:

1. This is a wonderful film that will make you laugh, cry, and feel good about humanity. The acting is superb, the script is clever, and the direction is flawless. **Позитивний.**
2. I was blown away by the visual effects, the action scenes were heart-pumping, and the cast had great chemistry. **Позитивний.**
3. A beautifully crafted film that explores complex themes with sensitivity and nuance. The performances are outstanding. **Позитивний.**
4. This movie is a complete waste of time. The plot is convoluted, the acting is wooden, and the dialogue is cringe-worthy. **Негативний.**
5. I was bored out of my mind watching this. The pacing is slow, the characters are uninteresting, and the ending is predictable. **Негативний.**
6. A mess of a film with poor editing, terrible CGI, and a nonsensical plot. Avoid at all costs. **Негативний.**

```
num_words = 2000
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=num_words)
```

3. Первинна обробка даних

```
max_length = 180
padded_train = pad_sequences(train_data, maxlen=max_length)
padded_test = pad_sequences(test_data, maxlen=max_length)
```

4. Визначення відповідей як категорійної змінної

```
num_classes = 2
train_labels_onehot = to_categorical(train_labels, num_classes)
test_labels_onehot = to_categorical(test_labels, num_classes)
```

5. Визначення архітектури мережі

Мережа складається з шару, що закодовує текст (Embedding), шару із довгою-короткою пам'яттю, повнозв'язного шару.

```
model = Sequential()
model.add(Embedding(input_dim=num_words, output_dim=80,
input_length=max_length))
model.add(LSTM(units=64))
model.add(Dense(num_classes, activation='softmax'))

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

6. Навчання моделі

```
model.fit(padded_train, train_labels_onehot, epochs=3,
          validation_data=(padded_test, test_labels_onehot))
```

7. Визначення точності моделі на тестовій вибірці

```
loss, accuracy = model.evaluate(padded_test, test_labels_onehot)
print('Test accuracy')
print(accuracy)
```

5.3. Індивідуальне завдання

1. Навчити нейронну мережу, використовуючи архітектуру в прикладі.
2. Модифікувати архітектуру мережі відповідно до власного варіанту (табл 5.1). В таблиці вказано конфігурацію блоків, що необхідно змінити.
3. Наведіть графік точності навчання нової архітектури. Визначте точність навчання на тестовій вибірці. Порівняйте якість отриманої архітектури із початковою.
4. Визначити точність на навчальний вибірці та порівняти її з тестовою точністю.

5. Проаналізуйте вплив розміру шару LSTM на кінцеву точність навчання. Зробіть висновки.

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, розрахунки, необхідні описи та графіки (відповідно до завдання вище) та вихідний код програм.

Назва файлу: Прізвище_група

Роботи, виконані не за варіантом, не приймаються.

Табл. 5.1. – Варіанти індивідуальних завдань.

Варіант	Макс. довжина	Розмір Embedding	Розмір LSTM
1	100	20	32
2	120	40	48
3	140	60	56
4	160	80	64
5	180	20	32
6	100	40	48
7	120	60	56
8	140	80	64
9	160	20	32
10	180	40	48
11	100	60	56
12	120	80	64
13	140	20	32
14	160	40	48
15	180	60	56
16	100	80	64
17	120	20	32
18	140	40	48
19	160	60	56
20	180	80	64
21	100	20	32
22	120	40	48
23	140	60	56
24	160	80	64
25	180	20	32

5.4. Контрольні питання

1. Чому для задачі обробки текстів ми використали рекурентну нейронну мережу, а не інші, знайомі нам із попередніх лабораторних?
2. Які ви можете навести приклади задач, що можуть бути розв’язані за допомогою рекурентних нейронних мереж?
3. В чому особливість рекурентної нейронної мережі у порівнянні із звичайною повнозв’язною?
4. Які функції обчислює кожен блок рекурентної мережі?
5. В чому особливість блоку LSTM?
6. Яким чином можна подати текст на вхід до нейронної мережі?
7. Як можна використати хмару тегів для аналізу набору текстових даних?
8. Запустіть навчену нейронну мережу із різними власними прикладами. Проаналізуйте, в яких випадках мережа найчастіше помиляється? Чому?
9. Яка команда в MATLAB відповідає за створення шару LSTM?
10. Яка команда відповідає за навчання нейронної мережі?
11. Які вимоги висуваються до алгоритмів, що працюють із послідовностями?
12. Чи можна ефективно передбачити клас послідовності за допомогою лише повнозв’язного шару? Чому?
13. Навіщо ми вводимо поняття історії для рекурентної нейронної мережі?
14. З якою метою вхідний набір даних розбивається на навчальну та тестові послідовності?
15. Яким чином можна вказати кількість епох для навчання нейронної мережі?

Практична робота №6 – Відновлення спотвореного сигналу за допомогою мережі-автокодувальника

Мета роботи: закріплення теоретичних знань та розвинення навичок побудови нейронної мережі із архітектурою автокодувальника, її навчання для задачі відновлення сигналу.

Практична робота присвячена знайомству з алгоритмами машинного навчання для обробки неповних, частково визначених або пошкоджених сигналів.. Робота виконується із використанням мови програмування Python та бібліотек Keras та TensorFlow. За узгодженням з викладачем студент може запропонувати свій набір даних для розв'язання задачі практичної роботи. Допускається виконання роботи мовою програмування MATLAB.

6.1. Теоретичні відомості

Автокодувальник – це нейронна мережа, навчена копіювати вхідні дані на вихід мереж. Всередині автокодувальник має прихований шар h, який описує *код*, який використовується для представлення вхідних даних. Мережа може розглядатися як така, що складається з двох частин: функції кодувальника $h = f(x)$ і декодувальника, який виробляє реконструкцію $r = g(h)$.

Загальну структуру мережі автокодувальника представлено на рис. 6.1. Мережа відображає вхід x на вихід r (що зветься реконструкцією) через внутрішнє представлення або код h. Автокодувальник має 2 компоненти: кодувальник f (відображення x в h), декодувальник g (відображення h в r).

Якщо автокодувальнику вдається просто навчитися встановлювати $g(f(x)) = x$ усюди, то це не особливо корисно. Натомість автокодувальники розроблені таким чином, що вони не можуть навчитися ідеально копіювати. Зазвичай вони обмежені способами, які дозволяють копіювати лише приблизно та копіювати лише вхідні дані, які нагадують навчальні дані. Оскільки модель змушена визначати пріоритетність того, які аспекти вхідних даних слід скопіювати, вона часто вивчає корисні властивості даних.

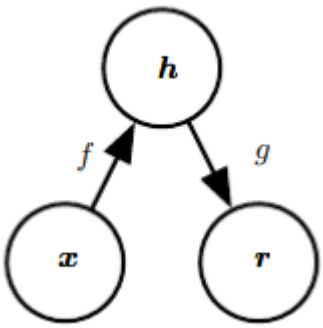


Рис. 6.1. Загальна структура мережі автокодувальника.

Сучасні автокодувальники узагальнили ідею кодувальника та декодувальника за межі детермінованих функцій до стохастичних відображень $p_{\text{кодувальник}}(h|x)$ та $p_{\text{декодувальник}}(x|h)$.

Ідея автокодувальників була частиною історичного ландшафту нейронних мереж протягом десятиліть. Традиційно автокодувальники використовувалися для зменшення розмірності або вивчення ознак. Нещодавно теоретичні зв'язки між автокодувальниками та моделями латентних змінних вивели автокодувальники на передній план генеративного моделювання. Автокодувальники можна розглядати як окремий випадок мереж прямого зв'язку, і їх можна навчати з усіма тими ж методами, як правило, градієнтний спуск за градієнтами, обчисленими шляхом зворотного поширення. На відміну від загальних мереж прямого зв'язку, автокодувальники також можна навчити за допомогою рециркуляції, алгоритму навчання, заснованого на порівнянні активації мережі на оригінальному вході з активаціями на реконструйованому вході. Рециркуляція вважається більш біологічно правдоподібною, ніж зворотне розповсюдження, але рідко використовується для програм машинного навчання

Стискаючі автокодувальники. Копіювання входу на вихід може здатися марним, але зазвичай нас не цікавить вихід декодувальника. Замість цього ми сподіваємося, що навчання автокодувальника виконувати завдання копіювання вхідних даних призведе до того, що h набере корисних властивостей.

Один із способів отримати корисні ознаки від автокодувальника – обмежити розмірність h величиною, меншою за x. Автокодувальник, розмірність коду якого менша за вхідну, називається *стискаючим*. Вивчення неповного представлення змушує автокодувальник захоплювати найбільш помітні характеристики навчальних даних.

Процес навчання описується просто як мінімізація функції втрат

$$L(x, g(f(x))), \tag{6.1}$$

де L є функцією втрат, яка штрафує $g(f(x))$ за відмінність від x, наприклад середньоквадратична помилка.

Коли декодувальник є лінійним і L є середньоквадратичною помилкою, стискаючий автокодувальник навчається охоплювати той самий підпростір, що й метод головних компонент. У цьому випадку автокодувальник, навчений виконувати завдання копіювання, вивчив головний підпростір навчальних даних як побічний ефект.

Автокодувальники з нелінійними функціями f кодувальника і нелінійними функціями g декодувальника можуть вивчати більш потужне нелінійне узагальнення методу головних компонент. На жаль, якщо кодувальнику та декодувальнику надано надто велику ємність, автокодувальник може навчитися виконувати задачу копіювання, не витягуючи корисну інформацію про розподіл даних. Теоретично можна уявити, що автокодувальник з одновимірним кодом, але із дуже потужний нелінійний кодувальником, може навчитися представляти

кожен навчальний приклад $x^{(i)}$ за допомогою коду i. Декодувальник може навчитися відображати ці цілочисельні індекси назад до значень конкретних навчальних прикладів. Цей конкретний сценарій не зустрічається на практиці, але він чітко ілюструє, що автокодувальник, навчений виконувати задачу копіювання, може не дізнатися нічого корисного про набір даних, якщо ємність автокодувальника стане занадто великою.

Регуляризовані автокодувальники. Стискаючі автокодувальники з розмірністю коду, меншою за вхідну, можуть вивчати найважливіші особливості розподілу даних. Як було описано вище, ці автокодувальники не можуть навчитися нічому корисному, якщо кодувальнику та декодеру надано занадто велику ємність.

Подібна проблема виникає, якщо прихованому коду дозволено мати розмірність, що дорівнює вхідній інформації, а також у випадку розтискаючого автокодувальника, коли прихований код має розмірність, більшу за вхідну. У цих випадках навіть лінійний кодувальник і лінійний декодувальник навчитися копіювати вхідні дані на вихід, не вивчивши нічого корисного про розподіл даних.

В ідеалі можна було б успішно навчити будь-яку архітектуру автокодувальника, вибравши розмірність коду та потужність кодувальника та декодувальника на основі складності розподілу, що моделюється. Регуляризовані автокодувальники надають можливість це зробити. Замість того, щоб обмежувати ємність моделі, зберігаючи кодувальник і декодувальник неглибокими, а розмір коду малим, регуляризовані автокодувальники використовують функцію втрат, яка заохочує модель мати інші властивості, окрім здатності копіювати вхідні дані на вихід. Ці інші властивості включають розрідженість подання, малість похідної подання та стійкість до шуму або відсутніх вхідних даних. Регуляризований автокодувальник може бути нелінійним і розтискаючим, але все одно вивчить щось корисне про розподіл даних, навіть якщо ємність моделі достатньо велика, щоб вивчити тривіальну функцію тотожності.

На додаток до описаних тут методів, які найбільш природним чином інтерпретуються як регуляризовані автокодувальники, майже будь-яку генеративну модель із латентними змінними та оснащену процедурою висновку (для обчислення латентних уявлень на вхідних даних) можна розглядати як особливу форму автокодувальника. Два підходи до генеративного моделювання, які підкреслюють цей зв’язок з автокодувальники, є нащадками машини Гельмгольца, наприклад, варіаційний автокодувальник і генеративні стохастичні мережі. Ці моделі природним чином навчаються високоємним, розтискаючим кодуванням вхідних даних і не вимагають регуляризації, щоб ці кодування були корисними. Їхнє кодування природно корисне, оскільки моделі були навчені приблизно максимізувати ймовірність навчальних даних, а не копіювати вхідні дані у вихідні дані.

Розріджений автокодувальник. Розріджений автокодувальник – це автокодувальник, критерій навчання якого передбачає штраф розрідженості $\Omega(h)$ на кодовому шарі h, на додаток до помилки реконструкції:

$$L\left(x, g\left(f(x)\right)\right) + \Omega(h), \tag{6.2}$$

де $g(h)$ – це вихідний сигнал декодувальника, $h = f(x)$ – вихідний сигнал кодувальника.

Розріджені автокодувальники зазвичай використовуються для вивчення ознак для іншої задачі, наприклад класифікації. Автокодувальник, який було регуляризовано, щоб бути розрідженим, повинен реагувати на унікальні статистичні характеристики набору даних, на якому його було навчено, а не просто діяти як тотожна функція. Таким чином, навчання виконанню задачі копіювання зі штрафом за розрідженість може дати модель, яка вивчила додатково корисні ознаки.

Можна думати про штраф $\Omega(h)$ просто як про член регуляризатора, доданий до мережі прямого зв'язку, головною задачею якої є копіювання вхідних даних на вихід (з цільовою функцією неконтрольованого навчання) і, можливо, також виконання певного контрольованого завдання (з цільовою функцією контрольованого навчання), що залежить від цих розріджених ознак.

Замість того, щоб розглядати штраф розрідженості як регуляризатор для задачі копіювання, розглянемо всю структуру розрідженого автокодувальника як наближене навчання максимальної правдоподібності генеративної моделі, яка має латентні змінні. Припустимо, що ми маємо модель з видимими змінними x і прихованими змінними h, з явним спільним розподілом $p_{model}(x, h) = p_{model}(h)p_{model}(x|h)$. Тоді $p_{model}(h)$ називають апріорним розподілом моделі за латентними змінними, що представляє переконання моделі до того, як побачити x. Логарифм правдоподібності можна розкласти як

$$\log p_{model}(x) = \log \sum_{h} p_{model}(h, x) \tag{6.3}$$

Тобто автокодувальник наближає цю суму за допомогою точкової оцінки лише для одного дуже ймовірного значення h. Це схоже на генеративну модель розрідженого кодування, але h є виходом параметричного кодувальника, а не результатом оптимізації, яка виводить найбільш імовірний h. З цієї точки зору, з вибраним h ми максимізуємо

$$\log p_{model}(h, x) = \log p_{model}(h) + \log p_{model}\,(x|h) \tag{6.4}$$

Член $\log p_{model}(h)$ може стимулювати розрідженість. Наприклад, апріорний розподіл Лапсаса,

$$p_{model}(h_i) = \frac{\lambda}{2} e^{-\lambda|h_i|}, \tag{6.5}$$

відповідає розрідженому штрафу з абсолютним значенням. Виражаючи логарифм апріорного розподілу як штраф за абсолютним значенням, отримуємо:

$$\Omega(h) = \lambda \sum_i |h_i|, \tag{6.6}$$

$$-\log p_{model}(h) = \sum_i \left(\lambda |h_i| - \log \frac{\lambda}{2}\right) = \Omega(\mathrm{h}) + const, \tag{6.7}$$

де постійний член залежить лише від λ, а не від h. Зазвичай розглядаємо λ як гіперпараметр і відкидаємо постійний член, оскільки він не впливає на вивчення параметра. Інші розподіли, такі як t-розподіл Стьюдента, також можуть стимулювати розрідженість. З цієї точки зору розрідженості як результату впливу $p_{model}(h)$ на апроксимоване навчання максимальної правдоподібності, штраф за розрідженість взагалі не є членом регуляризації. Це лише наслідок розподілу моделі по її латентним змінним. Ця точка зору забезпечує іншу мотивацію для навчання автокодувальника: це спосіб апроксимованого навчання генеративної моделі. Це також надає іншу причину чому ознаки, отримані автокодувальником, корисні: вони описують латентні змінні, які пояснюють вхід.

Ідея одного із способів досягнення фактичних нулів у h для розріджених автокодувальників (і тих, що усувають шуми) полягає в тому, щоб використовувати ReLU для створення шару коду. З апріорним розподілом, що фактично обнуляє представлення (наприклад, штраф абсолютної величини), таким чином можна опосередковано контролювати середню кількість нулів у представленні.

Автокодувальники, що усуваються шум. Замість того, щоб додавати штраф Ω до функції вартості, можемо отримати автокодувальник, який вивчає щось корисне, змінюючи член помилки реконструкції функції вартості.

Зазвичай, автокодувальник мінімізує певну функцію

$$L\left(x, g\left(f(x)\right)\right) \tag{6.8}$$

де L є функцією втрат, яка штрафує $g(f\,(x))$ за відмінність від x. Однією з функцій втрат може бути середньоквадратична помилка. Це заохочує $g \circ f$ навчитися бути просто тотожною функцією, якщо вони мають на це здатність.

Натомість автокодувальник, що усуває шум, мінімізує

$$L\left(x, g\left(f(\tilde{x})\right)\right) \tag{6.9}$$

де $\tilde{x}$ є копією x, яка була пошкоджена певною формою шуму. Таким чином, автокодувальни, що усувають шум, повинні скасовувати це пошкодження, а не просто копіювати вхідні дані.

Усунення шуму змушує f і g неявно вивчати структуру $p_{data}(x)$. Таким чином, автокодувальники, що усувають шум надають є ще одним прикладом того, як корисні властивості можуть з'являтися як побічний продукт мінімізації помилок реконструкції. Вони також є прикладом того, як розтискаючі моделі з

великою ємністю можна використовувати як автокодувальники, якщо вжити заходів щодо запобігання вивчання тотожної функції.

Нормалізація шляхом штрафування похідних. Інша стратегія регулярізації автокодувальника полягає у використанні штрафу Ω, як у розріджених автокодувальниках,

$$L\left(x, g\left(f(x)\right)\right) + \Omega(h, x) \tag{6.10}$$

але з іншою формою Ω:

$$\Omega(h, x) = \lambda \sum_i \|\nabla_x h_i\|^2 \tag{6.11}$$

Це змушує модель вивчати функцію, яка не сильно змінюється, коли x змінюється незначно. Оскільки цей штраф застосовується лише до навчальних прикладів, він змушує автокодувальник вивчати ознаки, які збирають інформацію про навчальний розподіл. Регуляризований таким чином автокодувальник називається контрастним автокодувальником.

Потужність представлення, розмір шару та глибина

Автокодувальники часто навчаються лише з одношаровим кодувальником і одношаровим декодувальником. Однак це не обов'язкова вимога. Насправді використання глибоких кодувальників та декодувальників дає багато переваг.

Згадайте попередні розділи що глибина в мережі прямого зв'язку має багато переваг. Оскільки автокодувальники є мережею прямого зв'язку, ці переваги також стосуються автокодувальників. Крім того, кодувальник сам є мережею прямого зв'язку, як і декодувальник, тому кожен із цих компонентів автокодувальника може окремо отримати переваги від глибини.

Одна з головних переваг нетривіальної глибини полягає в тому, що універсальна теорема про апроксиматор гарантує, що повнозв'язна нейронна мережа з принаймні одним прихованим шаром може представляти апроксимацію будь-якої функції (в межах широкого класу) з довільним ступенем точності, за умови, що вона має достатньо прихованих нейронів. Це означає, що автокодувальник з одним прихованим шаром здатний як завгодно добре представляти тотожну функцію в області даних. Однак відображення вхідних даних у код неглибоке. Це означає, що ми не можемо застосувати довільні обмеження, наприклад те, що код має бути розрідженим. Глибокий автокодувальник із принаймні одним додатковим прихованим шаром у самому кодувальнику може як завгодно добре апроксимувати будь-яке відображення вхідних даних у код, враховуючи достатню кількість прихованих одиниць.

Глибина може експоненційно зменшити обчислювальні витрати на представлення деяких функцій. Глибина також може експоненційно зменшити кількість навчальних даних, необхідних для вивчення деяких функцій. Експериментально глибокі автокодувальники дають набагато краще стиснення, ніж відповідні неглибокі або лінійні автокодувальники.

6.2. Приклад розв’язування задачі

1. Імпорт необхідних модулів

```
from keras.models import Model
from keras.layers import Input, Dense
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
```

2. Завантаження набору даних

Використаємо вже відомий нам набір даних MNIST для тестування можливостей відновлення зашумленого сигналу за допомогою мережі авто-кодувальника.

```
(X_train, _), (X_test, _) = mnist.load_data()
X_train = X_train.reshape(-1, 784).astype('float32') / 255.0
X_test = X_test.reshape(-1, 784).astype('float32') / 255.0
```

3. Додамо штучно створений шум до зображень

Параметр noise_std контролює розмах зашумлення. Чим більшим він є, тим більше спотворюється зображення.

```
noise_std = 0.5
noisy_X_train = X_train + np.random.normal(0, noise_std, X_train.shape)
noisy_X_test = X_test + np.random.normal(0, noise_std, X_test.shape)
noisy_X_train = np.clip(noisy_X_train, 0, 1)
noisy_X_test = np.clip(noisy_X_test, 0, 1)
```

4. Візуалізуємо отримані зображення

Порівняйте дані зображення із отриманими в попередніх лабораторних роботах. Чи можете ви впізнати цифру, зображену на рисунку?

```
plt.figure()
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(noisy_X_test[i].reshape(28, 28), cmap='gray')
plt.show()
```

5. Створимо мережу авто-кодувальник із використанням повнозв’язних шарів

Спочатку створимо частину шарів, що кодує зображення:

```
input_img = Input(shape=(784,))
x = Dense(128, activation='relu')(input_img)
```

```
x = Dense(64, activation='relu')(x)
x = Dense(32, activation='relu')(x)
encoded_img = Dense(16, activation='relu')(x)
```

А далі частину, що його відновлює до початково стану. Така архітектура дозволить мережі навчитись прибирати шум із зображення та відновлювати його до початкового стану:

```
x = Dense(32, activation='relu')(encoded_img)
x = Dense(64, activation='relu')(x)
x = Dense(128, activation='relu')(x)
decoded_img = Dense(784, activation='sigmoid')(x)

autoencoder = Model(input_img, decoded_img)

autoencoder.compile(optimizer='adam', loss='mean_squared_error')
```

6. Проведемо навчання такої мережі.

```
autoencoder.fit(noisy_X_train, X_train,
                epochs=10,
                batch_size=128,
                validation_data=(noisy_X_test, X_test))
```

7. Застосуємо мережу автокодувальник для відновлення декількох прикладів з навчальної вибірки.

```
restored_img = autoencoder.predict(noisy_X_test[:10])
```

8. Зобразимо відновлені зображення.
Порівняйте їх із зашумленими. Наскільки вдало на вашу думку було проведено відновлення? Чи можете ви тепер розпізнати цифри?

```
plt.figure()
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(restored_img[i].reshape(28, 28), cmap='gray')
plt.show()
```

Загалом, порівнюючи зашумлені зображення із відновленими автокодувальником (рис. 6.2) можна помітити, автокодувальник є достатньо ефективним інструментом з боротьби навіть з досить сильним шумом.

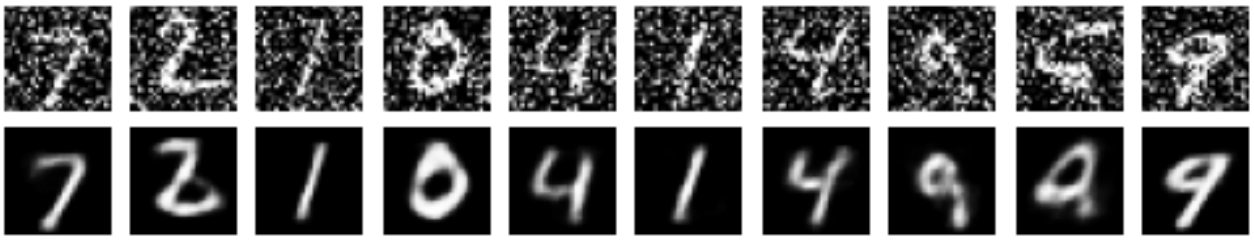


Рис. 6.1. – Зашумлені зображення (зверху), відновлені автокодувальником (знизу).

6.3. Індивідуальне завдання

1. Навчити нейронну мережу, використовуючи архітектуру в прикладі.
2. Модифікувати архітектуру мережі відповідно до власного варіанту (табл. 6.1). В таблиці вказано конфігурацію блоків, що необхідно змінити.
3. Наведіть графік точності навчання нової архітектури. Визначте середньоквадратичну помилку навчання на тестовій вибірці. Порівняйте якість отриманої архітектури із початковою.
4. Визначити середньоквадратичну помилку на навчальний вибірці та порівняти її з тестовою точністю.
5. Експериментально визначити максимальне значення розмаху шуму при якому ще відбувається відновлення результату. Навести зашумлені та відновлені зображення.

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, розрахунки, необхідні описи та графіки (відповідно до завдання вище) та вихідний код програм.

Назва файлу: Прізвище_група

Роботи, виконані не за варіантом, не приймаються.

Табл. 6.1. – Варіанти індивідуальних завдань.

Варіант	Розмах шуму	Розміри повнозв’язних шарів
1	0.42	[20, 40, 80, 80, 40, 20]
2	0.51	[24, 48, 96, 96, 48, 24]
3	0.38	[16, 32, 64, 64, 32, 16]
4	0.45	[28, 56, 112, 112, 56, 28]
5	0.49	[32, 64, 128, 128, 64, 32]
6	0.41	[18, 36, 72, 72, 36, 18]
7	0.53	[22, 44, 88, 88, 44, 22]

Варіант	Розмах шуму	Розміри повнозв’язних шарів
8	0.46	[26, 52, 104, 104, 52, 26]
9	0.39	[19, 38, 76, 76, 38, 19]
10	0.48	[30, 60, 120, 120, 60, 30]
11	0.43	[21, 42, 84, 84, 42, 21]
12	0.50	[25, 50, 100, 100, 50, 25]
13	0.44	[23, 46, 92, 92, 46, 23]
14	0.52	[27, 54, 108, 108, 54, 27]
15	0.40	[17, 34, 68, 68, 34, 17]
16	0.47	[29, 58, 116, 116, 58, 29]
17	0.36	[20, 40, 80, 80, 40, 20]
18	0.54	[31, 62, 124, 124, 62, 31]
19	0.42	[22, 44, 88, 88, 44, 22]
20	0.49	[26, 52, 104, 104, 52, 26]
21	0.41	[18, 36, 72, 72, 36, 18]
22	0.46	[24, 48, 96, 96, 48, 24]
23	0.51	[28, 56, 112, 112, 56, 28]
24	0.45	[21, 42, 84, 84, 42, 21]
25	0.53	[30, 60, 120, 120, 60, 30]

6.4. Контрольні питання

1. Якими є основні 2 компоненти мережі автокодувальника?
2. Які види автокодувальників бувають?
3. Які способи регуляризації автокодувальників існують? Які особливості кожного способу?
4. Як автокодувальник можна використовувати для відновлення сигналу?
5. В наведеному прикладі скільки шарів містила мережа автокодувальник?
6. Наскільки сильний шум можливо відновити за допомогою мережі автокодувальника?
7. Чи є вивчення тотожного відображення автокодувальником його позитивною властивістю або негативною? Відповідь обґрунтуйте.

Оцінювання результатів навчання

Оцінювання практичних робіт

Кожна робота оцінюється за 100 бальною шкалою. Оцінювання відбувається строго у відповідності до пунктів індивідуального завдання, наведених в кожній практичній роботі, та пропорційно до ступеня повноти та коректності виконання кожного з них. Важливими вимогами до звіту з практичної роботи є:

1. Чіткий структурований опис проведеного дослідження, наведення необхідних графічних ілюстрацій.
2. Аналіз результатів, власні припущення та висновки студента, щодо отриманих закономірностей.
3. Кожна робота має обов'язково містити лістинг програмного коду, коректність якого та відповідність завданню оцінюється викладачем, програмний код студент має розуміти та виконати самостійно.
4. Роботи, виконані не за варіантом, не приймаються.

Вимоги до звіту

Звіт з виконання практичної роботи має містити:

- обкладинку в корпоративному стилі НТУ «Дніпровська політехніка» із зазначенням групи, ПІБ студента, викладача та теми роботи;
- номер варіанта, мета роботи, повний текст завдання;
- за кожним пунктом завдання опис, що було виконано, які результати отримано. Якщо завдання вимагає включити всі необхідні графіки. Результати кожного пункту мають бути проаналізовані здобувачем;
- в кінці роботи має бути обов'язково наявний лістинг коду.

Звіт розміщується на сайті дистанційної освіти у відповідному розділі із завданням і оцінюється виходячи з 100 балів.

Рекомендовані джерела інформації

Базові:

1. Субботін С.О. Нейронні мережі: теорія і практика. Навчальний посібник. Житомир, 2020. 184с. URL: http://eir.zntu.edu.ua/bitstream/123456789/6800/1/Subbotin_Neural.pdf
2. І.А. Терейковський, Д.А. Бушуєв, Л.О. Терейковська Штучні нейронні мережі: базові положення. Навчальний посібник. Київ, 2022. 123с. URL: https://ela.kpi.ua/server/api/core/bitstreams/9fee52b6-83fc-4e99-8541-c2767f634c7c/content
3. Bishop C.M., Bishop H. Deep learning – foundations and concepts. Springer, 2024. 649p. DOI: 10.1007/978-3-031-45468-4.
4. TensorFlow documentation. [Online] URL : https://www.tensorflow.org/learn
5. I. Goodfellow, Y. Bengio, A. Courville The Deep Learning Book. MIT Press, 2016. URL: https://www.deeplearningbook.org/

Додаткові:

1. Deep Residual Learning for Image Recognition / K. He та ін. 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. – IEEE Computer Society, 2016. – P. 770-778. – DOI: 10.1109/CVPR.2016.90.
2. Khabarlak K., Koriashkina L. Fast Facial Landmark Detection and Applications: A Survey. Journal of Computer Science and Technology. – 2022. – Vol. 22. – № 1. – P. 02. – DOI: 10.24215/16666038.22.E02.

Навчальне видання

Желдак Тимур Анатолійович
Хабарлак Костянтин Сергійович
Гаранжа Дмитро Миколайович

САМОНАВЧАННЯ СКЛАДНИХ СИСТЕМ

Методичні рекомендації до виконання практичних робіт
для здобувачів ступеня магістра
освітньо-професійної програми «Системний аналіз»
зі спеціальності 124 Системний аналіз

Видано в авторській редакції.

Електронний ресурс.
Підписано до видання 24.06.2024. Авт. арк. 4,99.

Національний технічний університет «Дніпровська політехніка».
49005, м. Дніпро, просп. Дмитра Яворницького, 19.