Міністерство освіти і науки України
Національний авіаційний університет
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

Домашнє завдання
з дисципліни «Системи штучного інтелекту»
на тему «Дослідження нейронної мережі Хопфілда в задачах розпізнавання зображень»
Варіант № 8

Виконав:
студент ФККПІ
групи СП-425
Клокун В. Д.
Перевірила:
Росінська Г. П.

Київ 2020

Зміст

1. Загальні відомості

Нейронна мережа Хопфілда — це тип штучної рекурентної нейронної мережі, який у 1982 році популяризував Джон Хопфілд. Однак, варто зазначити, що до цього, у 1974 році, його описав Уільям Літл у своїй праці на тему моделювання людської пам'яті [1].

1.1. Використання

1.1.1. Асоціативна пам'ять

Мережу Хопфілда використовують декількома способами. Так як спочатку її описали як модель людської пам'яті, мережу використовують відповідним чином — як систему асоціативної (контентно-адресованої) пам'яті. *Асоціативна пам'ять* — це тип пам'яті, який дозволяє співставити пошуковий запит із таблицею усіх даних, що зберігаються, знайти відповідне йому значення та повернути його. Особливістю асоціативної пам'яті є швидкість роботи, тому вона використовується у високошвидкісних пошукових системах.

Наприклад, мережу можна використати для швидкого зберігання даних про столиці та країни, якщо представити їх у вигляді спеціального зв'язку (табл. 1).

Табл. 1: Схематичне зображення асоціативної пам'яті

(а) Бажані факти, записані в асоціативній пам'яті мережі

Київ	→	Україна
Осло	→	Норвегія
Москва	→	Росія
Оттава	→	Канада
Вашингтон	→	США

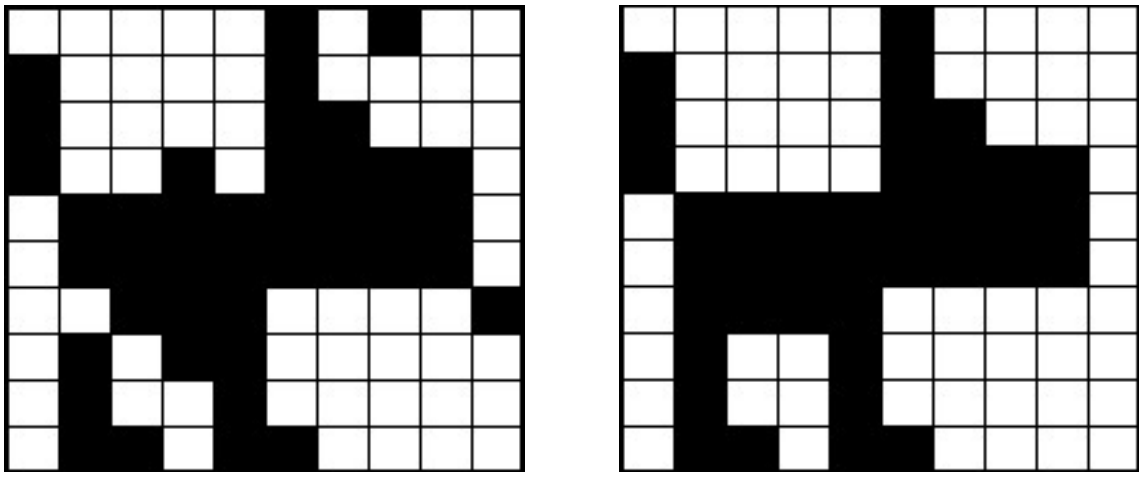
(б) Доповнення шаблонів і корекція помилок на їх основі

Київ	→	Україна
Ос о	→	Норвегія
Мкосва	→	Росія
От***а	→	Канада
Вашингитон	→	США

Як видно, навіть працюючи в якості асоціативної пам'яті, мережа Хопфілда дозволяє коригувати помилки. Ця властивість розповсюджується не лише на текстові дані. Дійсно, щоб використовувати мережу Хопфілда, необхідно представити вхідні дані в особливому форматі, але тим не менш, до цього формату можна звести початкові дані різних типів. Коли дані бажаного типу представлені у зручному для мережі форматі, вона так само може працювати як асоціативна пам'ять: швидко співставляти ключ зі збереженим значенням даних, а також коригувати помилки.

1.1.2. Корекція помилок

По-друге, мережу Хопфілда використовують як фільтр для виправлення помилок. Наприклад, можна закодувати зображення як матриці пікселів, елементи якої містять виключно одне з двох значень: чорне або біле, і зберегти їх у пам'ять мережі. Коли зображення знаходяться у пам'яті мережі, вона зможе їх відновити, навіть якщо їй на вхід подаватимуться пошкоджені версії збережених зображень (рис. 1).



а) Пошкоджене зображення, подане на вхід мережі

б) Вихідне зображення, відновлене до еталонного

Рис. 1: Приклад відновлення пошкодженого зображення

1.1.3. Розпізнавання образів

В такому випадку виходить, що мережа Хопфілда відновлює початкове зображення — еталон, тому іноді стверджують, що мережу використовують для розпізнавання образів. Однак, це не так. Щоб розв'язати задачу розпізнавання образів, інструмент повинен віднести вхідні данні до певної наперед заданої категорії, тобто класифікувати їх. Натомість, мережа Хопфілда не виконує цю задачу, а виключно відновлює інформацію, збережену всередині своїх нейронів, тому не зовсім коректно стверджувати, що мережу Хопфілда використовують для розв'язку задач розпізнавання образів.

1.1.4. Задачі оптимізації

Тим не менш, область використання мережі Хопфілда не обмежується збереженням і відновлення даних. Ці властивості дозволяють використати мережу для

задач оптимізації на кшталт задачі комівояжера. Хопфілд і Тенк запропонували метод, суть якого полягає в тому, щоб представити тимчасовий розв'язок задачі як стан мережі з $I = K^2$ нейронів, де K — кількість міст у поставленій задачі комівояжера. Тоді кожен нейрон представляє припущення, що в оптимальному маршруті саме це місто знаходиться на саме цьому кроці, і ця комбінація відповідає даному нейрону (рис. 2).

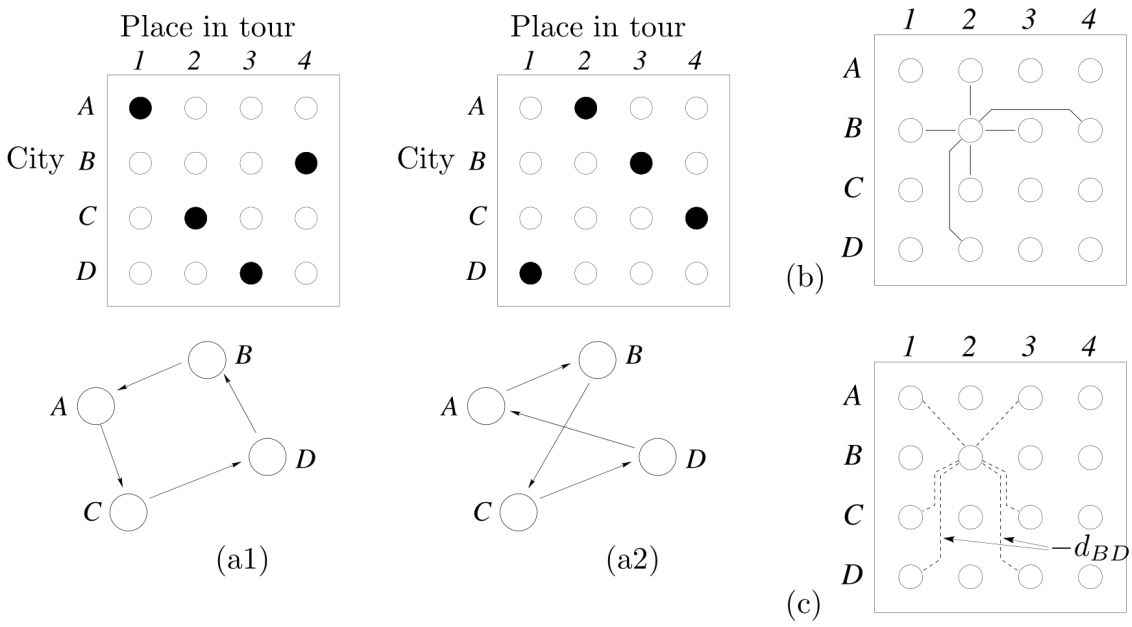


Рис. 2: Представлення можливих розв'язків задачі комівояжера у вигляді станів мережі Хопфілда

В такому випадку зручно представити стани нейронів як 0 і 1, а не -1 і 1. Тоді ваги покажуть функцію енергії, яка буде мінімізована лише тоді, коли стан мережі є правильним маршрутом. Правильний стан мережі, який відповідає правильному маршруту, — це такий стан, який виглядає як матриця перестановок, тобто в якій в кожному рядку і в кожній колонці є лише одне значення «1». Щоб мережа слідувала цьому правилу, можна встановити великі від'ємні значення між усіма парами нейронів, які знаходяться в одному рядку або одній колонці. Також можна встановити додатне значення зсуву, щоб переконатись, що K нейронів увімкнуться. Таким чином, оскільки мережа Хопфілда прагне мінімізувати свою енергію, вона також повинна мінімізувати довжину маршруту між містами, які необхідно обійти.

1.2. Актуальність

Властивості мережі Хопфілда дозволяють застосовувати її у декількох сферах, однак широкий спектр застосувань не завжди означає актуальність. Перш за все, станом на 2020 рік, мережу Хопфілда майже не використовують для реалізації асоціативної пам'яті у широких застосуваннях. Найбільша потреба в асоціативній пам'яті є у процесорах та мережевих пристроях. В обох цих випадках найважливішими параметрами є швидкість роботи, кількість пам'яті, простота та вартість реалізації. На превеликий жаль, характеристикою нейронних мереж є висока складність, особливо порівняно з іншими методами на кшталт буферів асоціативної трансляції (англ. *translation lookaside buffers, TSB*). Це призводить до низької надійності та високої вартості, тому мережа Хопфілда не отримала широкого розповсюдження у промислових застосуваннях.

Друга область використання знаходиться на межі між корекцією помилок і розпізнаванням образів, наприклад, виправлення рукописного вводу. Використовуючи здатність мережі Хопфілда запам'ятовувати бажані стани, можна натренувати її на еталонні форми символів і використовувати для «розпізнавання» рукописного вводу. Однак, усі ці функції точніше, швидше і краще виконує інший тип нейронних мереж — згорткові (англ. *convolutional neural networks, CNN*), тому і тут мережа Хопфілда не отримала широкого застосування.

Загалом, мережа Хопфілда є важливою академічною моделлю, яка підходить для того, щоб навчати студентів теорії штучних нейронних мереж.

2. Теоретичні відомості

Нейронна мережа Хопфілда визначається тим, що має повнозв'язну симетричну матрицю зв'язків. У процесі роботи динаміка таких мереж сходиться до одного з положень рівноваги. Ці положення рівноваги є локальними мінімумами функціоналу, який називається енергією мережі. На відміну від багатьох нейронних мереж, що працюють до отримання відповіді через певну кількість тактів, мережі Хопфілда працюють до досягнення рівноваги, коли наступний стан мережі дорівнює попередньому.

Залежно від функції активації нейронів, порядку оновлення та деяких інших параметрів роботи мережі, розрізняють декілька варіацій, зокрема двійкову та неперервну мережі Хопфілда.

2.1. Двійкова мережа Хопфілда

Двійкова мережа Хопфілда — це частковий випадок неперервного режиму роботи мережі, однак саме його найчастіше використовують для вирішення задач розпізнавання образів, тому почнемо розгляд мережі на саме з цього режиму.

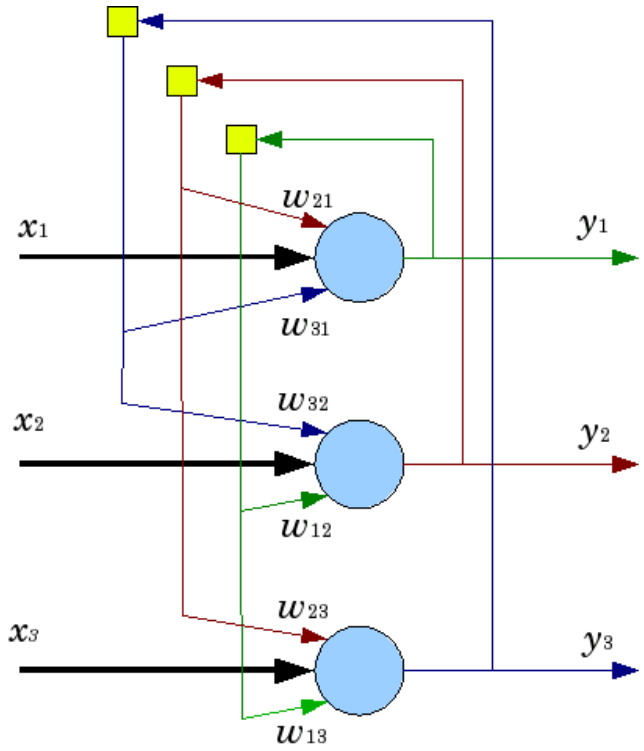


Рис. 3: Приклад мережі Хопфілда

2.1.1. Архітектура

Позначимо зв'язок від нейрона j до нейрона i як w_{ij}. Мережа Хопфілда складається з I нейронів. Зазвичай зв'язки між нейронами даної мережі підкорюються таким правилам:

1. Вони повністю зв'язані між собою за допомогою симетричних, двонаправлених зв'язків, тобто $w_{ij} = w_{ji}$. Це правило гарантує, що під час застосування правила активації, енергія функції монотонно спадає [2]. Інакше мережа може зациклюватись або вести себе хаотично.
2. Всередині мережі відсутні зв'язки нейрона самого з собою, тобто для будь-якого i вага $w_{ij} = 0$.

Також у мережі можуть бути зсуви (англ. *bias*), які можна розглядати як зв'язки від нейрона 0, вихід якого завжди дорівнює $x_0 = 1$. Вихід нейрона позначимо як x_i, а сукупність усіх поточних значень виходів нейронів будемо називати станом мережі.

2.1.2. Оновлення і робота мережі

Грубо кажучи, правило роботи в двійковій мережі Хопфілда полягає в тому, щоб кожний нейрон оновлював свій стан, ніби він — єдиний нейрон, який має порогову функцію активації:

$$x(a) \equiv \Theta(a) \equiv \begin{cases} 1, \text{ якщо } a \geqslant \theta_i, \\ -1, \text{ у противному випадку.} \end{cases}$$

Тут θ_i — це порогове значення. Коли зважена сума у нейроні набуває значення, яке більше за θ_i, нейрон активується.

Всередині мережі Хопфілда є зворотній зв'язок, тобто вихід кожного нейрона є входом для усіх інших нейронів, тому необхідно зазначити порядок, у якому будуть виконуватись оновлення. Оновлення можуть бути:

— синхронними, коли спочатку усі нейрони оновлюють свої значення за формулою:

$$a_i = \sum_j w_{ij} x_j,$$

а потім одночасно оновлюють свої стани до значення x_i, яке розраховується так:

$$x_i = \Theta(a_i)$$

— асинхронними, коли у певний момент часу лише один нейрон обчислює своє значення. Порядок обраних нейронів може бути зафіксованим або випадковим.

Залежно від типу оновлень, властивості мережі Хопфілда можуть змінюватись.

2.1.3. Енергія

Кожне оновлення мережі змінює її стан. Із кожним станом мережі Хопфілда пов'язана скалярна величина, яку називають енергією мережі E. Вона обчислюється так:

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} x_i x_j + \sum_i \theta_i x_i$$

Ця величина називається енергією, бо після того, як нейрони мережі оновлюються, енергія може зменшуватись або залишатись незмінною (рис. 4).

Крім того, після повторних оновлень мережа так чи інакше зійдеться до стану, який є локальним мінімумом функції енергії. Отже, якщо стан мережі відповідає локальному мінімуму функції енергії, він вважається стабільним станом мережі.

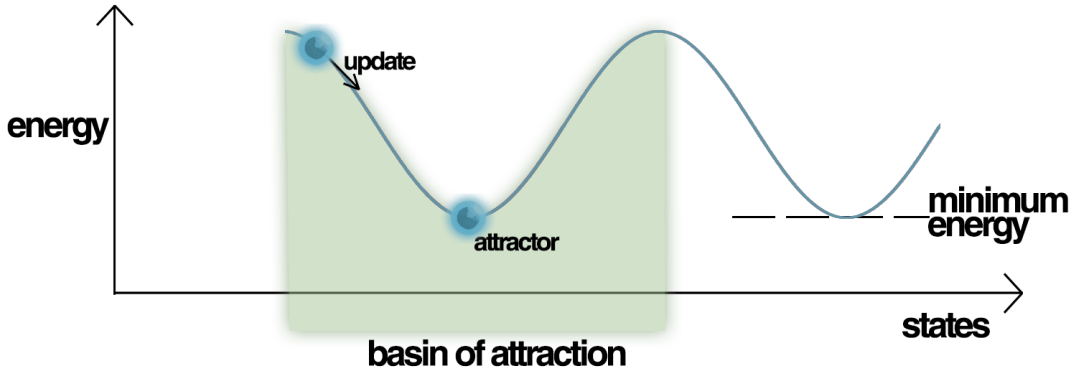


Рис. 4: Ландшафт енергії мережі Хопфілда, який показує поточний стан мережі (на горі), притягаючий стан, до якого так чи інакше зійдеться мережа, та впадину притяжіння, відтінену зеленим кольором

2.1.4. Тренування мережі

Штучні нейронні мережі тренують за допомогою правил навчання. Існують різні правила навчання нейронних мереж, але бажано, щоб правило навчання мало такі 2 властивості:

1. Локальність. Правило навчання вважається локальним, якщо для оновлення ваги кожного нейронного зв'язку достатньо лише тої інформації, яка міститься виключно у тих нейронах, що з'єднані цим зв'язком.
2. Інкрементальність. Правило навчання вважається інкрементальним, коли з його допомогою мережа може вивчити нові поняття, не використовуючи інформацію про поняття, які вона вивчила раніше. Тобто для того, щоб оновити ваги зв'язків для вивчення нового поняття, мережі вистачить лише минулих значень ваг та власне нового поняття.

Одним з правил навчання, які мають обидві вищезазначені властивості, є правило Хебба, яке часто застосовують для тренування мережі Хопфілда. Як відомо, правило навчання оновлює ваги зв'язків між нейронами мережі таким чином, щоб мережа вчилась певним поняттям.

У нейронній мережі Хопфілда кожен факт представляється як двійковий набір даних $\mathbf{x}$, де $x_i \in \{-1, 1\}$, тому правило навчання має зробити множину $\{\mathbf{x}^{(n)}\}$, яка містить факти, що треба закріпити, стабільними станами мережі. Щоб зробити будь-які стани стабільними, тобто «записати» факти у мережу, необхідно зменшувати енергію, яку набуватиме нейронна мережа при переході до бажаного стану.

Щоб зменшувати енергію бажаних станів, необхідно оновлювати ваги зв'язків між нейронами відповідним способом, тому за правилом Хебба для мережі

Хопфілда ваги зв'язків між нейронами встановлюються так:

$$w_{ij} = \eta \sum_n x_i^{(n)} x_j^{(n)},$$

де η — певна неважлива константа. Щоб попередити ситуацію, при якій найбільша вага буде рости разом зі значенням N, можна обрати значення константи $\eta = 1/N$.

Ваги зв'язків між нейронами істотно впливають на значення виходів нейронів та навчання мережі. У мережі Хопфілда вони симулюють тяжіння та відштовхування нейронів між собою. Наприклад, розглянемо вагу між нейронами i і j, яка позначається w_{ij}. Якщо вага $w_{ij} > 0$, правило активації означає, що:

— коли значення нейрона $x_j = 1$, нейрон j збільшує зважену суму, отже, нейрон j притягує значення нейрона x_i до значення $x_i = 1$.

— коли значення нейрона $x_j = -1$, нейрон j зменшує зважену суму, отже, нейрон j відштовхує значення нейрона x_i до значення $x_i = -1$.

Таким чином, значення нейронів i і j будуть збігатись, якщо вага зв'язку між ними додатна, і, навпаки, розбігатись, якщо вага від'ємна.

3. Реалізація

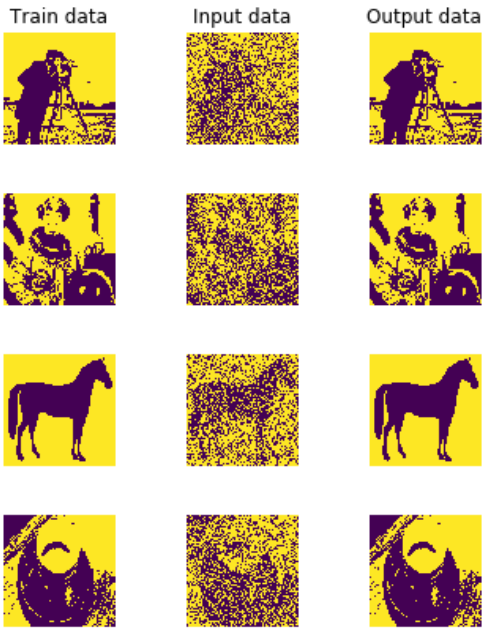
Описавши теоретичні відомості про мережу Хопфілда, використаємо її реалізацію [3] для вирішення задач, які найбільше схожі на розпізнавання зображення. Першою такою задачею буде відновлення пошкодженого початкового зображення. Тоді порядок роботи програми виглядатиме так:

1. Запам'ятати вхідні зображення.
2. Внести випадкові пошкодження в еталонне зображення.
3. Подати пошкоджене зображення на вхід нейронної мережі.
4. Запустити нейронну мережу.
5. Коли мережа зійдеться до стабільного стану, вивести результат.

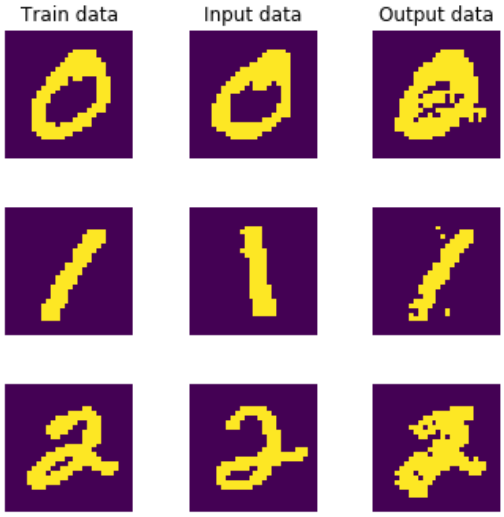
Наступний режим роботи реалізації, який в контексті даного завдання можна розуміти як розпізнавання зображень — це відновлення рукописних цифр. В якості тренувальних даних, тобто набору рукописних цифр, використаємо деякі цифри із набору даних MNIST. При цьому програма не буде класифікувати цифри, адже нейронна мережа Хопфілда не призначена для категоризації вхідних даних. Тем не менш, вона зможе відновити більшість вхідних рукописних цифр до значень, схожих на еталонні. У цьому випадку порядок роботи програмної реалізації буде аналогічним.

Отже, щоб використати реалізацію, встановлюємо її залежності, запускаємо і тренуємо. Під час роботи програми отримаємо результат у вигляді графічних звітів (рис. 5).

Як видно з результатів відновлення зображень (рис. 5а), реалізована нейронна мережа здатна досить точно відновити 4 пошкоджених еталонних зображень,



а) Відновлення пошкодженого зображення



б) Відновлення рукописних цифр

Рис. 5: Результати роботи реалізації мережі Хопфілда

причому варто зазначити, що вхідні дані були суттєво змінені випадковими даними.

Що до відновлення рукописних цифр, результати менш вражаючі (рис. 5б): видно, що мережа намагалась відновити еталонні зображення, однак в процесі відновлення створила навіть більше артефактів, ніж на вхідних даних. Оскільки результати роботи мережі не повернулись до точного еталонного стану, їх не можна точно порівняти з еталонними і визначити, що вхідні дані належать певному класу. Отже, мережі бракує точності, щоб використовувати її для задачі розпізнавання зображень.

Висновки

Виконуючи дане домашнє завдання, ми ознайомились із нейронною мережею Хопфілда, дослідили області її використання, актуальність, архітектуру та принцип роботи, а також використали реалізацію, щоб розв'язати завдання, суміжні з розпізнаванням зображень.

Визначено, що нейронна мережа Хопфілда може бути використана як асоціативна пам'ять, тобто пам'ять, яка дозволяє відновити повний зміст певної інформації лише за її частиною. Також мережу Хопфілда можна використати як фільтр, який коригуватиме помилки у вхідній інформації так, щоб вона співпадала з еталонною. Крім цього, мережа Хопфілда може евристично розв'язувати задачі оптимізації на кшталт задачі комівояжера. Для цього достатньо представити задачу у форматі, зручному для роботи мережі.

Незважаючи на те, що мета домашнього завдання полягала у дослідженні роботи мережі Хопфілда для розпізнавання зображень, виявилось, що вона взагалі не може розв'язувати ці задачі у їх строгому формулюванні. Річ у тім, що мережа Хопфілда не здатна відносити вхідні дані до певного класу — вона може виключно відновлювати їх до певного стабільного стану. Так, цей стан можна задати параметрично, як це потрібно для розв'язання задачі комівояжера, але цю властивість неможливо застосувати для класифікації даних, і тому після відновлення еталонного зображення класифікація буде окремою і часто дуже складною задачею.

Також було виявлено, що мережа Хопфілда не отримала широкого розповсюдження у повсякденних застосуваннях. Для реалізації асоціативної пам'яті у сучасних пристроях, які її потребують, мережу витіснили асоціативні буфери трансляції; як для реалізації асоціативної структури даних у програмі, то нейронна мережа буде занадто складною. Із задачами розпізнавання образів значно краще справляються згорткові нейронні мережі, тому станом на 2020 рік вони отримали найбільше розповсюдження для їх розв'язання. В задачах оптимізації зазвичай використовуються інші алгоритми, не пов'язані з машинним навчанням, на кшталт симплекс-метода, метода Нелдера—Меда та еліпсоїдного метода. Тому можна зробити висновок, що мережа Хопфілда актуальна лише для навчання людей основам нейронних мереж.

Ще у домашньому завданні були викладені теоретичні відомості про мережу Хопфілда, а згодом на їх основі був показаний приклад роботи мережі для розв'язання задач, що найближче схожі на задачі розпізнавання образів. Результати роботи вкотре підкреслили, що нейронна мережа Хопфілда підходить виключно для відновлення початкових даних, але не є повноцінним рішенням для розпізнавання зображень.

Література

1. *Kevin Gurney*. An Introduction to Neural Networks. — Wiley, 1997. — ISBN 1857286731.
2. *David J.C. MacKay*. Information Theory, Inference, and Learning Algorithms. — Cambridge University Press, 2003. — ISBN 9780521642989.
3. *takyamamoto*. Hopfield Network : Hopfield Network implemented with Python. — URL: `https://github.com/takyamamoto/Hopfield-Network` (дата зверн. 01.04.2020).

А. Лістинг реалізації

Лістинг А.1: Файл `network.py`

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jul 29 08:40:49 2018

@author: user
"""

import numpy as np
from matplotlib import pyplot as plt
import matplotlib.cm as cm
from tqdm import tqdm

class HopfieldNetwork(object):
    def train_weights(self, train_data):
        print("Start to train weights...")
        num_data =  len(train_data)
        self.num_neuron = train_data[0].shape[0]

        # initialize weights
        W = np.zeros((self.num_neuron, self.num_neuron))
        rho = np.sum([np.sum(t) for t in train_data]) /
         ↪  (num_data*self.num_neuron)

        # Hebb rule
        for i in tqdm(range(num_data)):
            t = train_data[i] - rho
            W += np.outer(t, t)

        # Make diagonal element of W into 0
        diagW = np.diag(np.diag(W))
        W = W - diagW
```

```
        W /= num_data

        self.W = W

    def predict(self, data, num_iter=20, threshold=0, asyn=False):
        print("Start to predict...")
        self.num_iter = num_iter
        self.threshold = threshold
        self.asyn = asyn

        # Copy to avoid call by reference
        copied_data = np.copy(data)

        # Define predict list
        predicted = []
        for i in tqdm(range(len(data))):
            predicted.append(self._run(copied_data[i]))
        return predicted

    def _run(self, init_s):
        if self.asyn==False:
            """
            Synchronous update
            """
            # Compute initial state energy
            s = init_s

            e = self.energy(s)

            # Iteration
            for i in range(self.num_iter):
                # Update s
                s = np.sign(self.W @ s - self.threshold)
                # Compute new state energy
                e_new = self.energy(s)

                # s is converged
                if e == e_new:
                    return s
                # Update energy
                e = e_new
            return s
        else:
            """
            Asynchronous update
            """
            # Compute initial state energy
```

```
78              s = init_s
79              e = self.energy(s)
80
81              # Iteration
82              for i in range(self.num_iter):
83                  for j in range(100):
84                      # Select random neuron
85                      idx = np.random.randint(0, self.num_neuron)
86                      # Update s
87                      s[idx] = np.sign(self.W[idx].T @ s - self.threshold)
88
89                  # Compute new state energy
90                  e_new = self.energy(s)
91
92                  # s is converged
93                  if e == e_new:
94                      return s
95                  # Update energy
96                  e = e_new
97              return s
98
99
100     def energy(self, s):
101         return -0.5 * s @ self.W @ s + np.sum(s * self.threshold)
102
103     def plot_weights(self):
104         plt.figure(figsize=(6, 5))
105         w_mat = plt.imshow(self.W, cmap=cm.coolwarm)
106         plt.colorbar(w_mat)
107         plt.title("Network Weights")
108         plt.tight_layout()
109         plt.savefig("weights.png")
110         plt.show()
```

Лістинг А.2: Файл `train.py`

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Jul 29 08:40:49 2018
4
5  @author: user
6  """
7
8  import numpy as np
9  np.random.seed(1)
10 from matplotlib import pyplot as plt
```

```
import skimage.data
from skimage.color import rgb2gray
from skimage.filters import threshold_mean
from skimage.transform import resize
import network

# Utils
def get_corrupted_input(input, corruption_level):
    corrupted = np.copy(input)
    inv = np.random.binomial(n=1, p=corruption_level, size=len(input))
    for i, v in enumerate(input):
        if inv[i]:
            corrupted[i] = -1 * v
    return corrupted

def reshape(data):
    dim = int(np.sqrt(len(data)))
    data = np.reshape(data, (dim, dim))
    return data

def plot(data, test, predicted, figsize=(5, 6)):
    data = [reshape(d) for d in data]
    test = [reshape(d) for d in test]
    predicted = [reshape(d) for d in predicted]

    fig, axarr = plt.subplots(len(data), 3, figsize=figsize)
    for i in range(len(data)):
        if i==0:
            axarr[i, 0].set_title('Train data')
            axarr[i, 1].set_title("Input data")
            axarr[i, 2].set_title('Output data')

        axarr[i, 0].imshow(data[i])
        axarr[i, 0].axis('off')
        axarr[i, 1].imshow(test[i])
        axarr[i, 1].axis('off')
        axarr[i, 2].imshow(predicted[i])
        axarr[i, 2].axis('off')

    plt.tight_layout()
    plt.savefig("result.png")
    plt.show()

def preprocessing(img, w=128, h=128):
    # Resize image
    img = resize(img, (w,h), mode='reflect')

```

```
58      # Thresholding
59      thresh = threshold_mean(img)
60      binary = img > thresh
61      shift = 2*(binary*1)-1 # Boolian to int
62
63      # Reshape
64      flatten = np.reshape(shift, (w*h))
65      return flatten
66
67  def main():
68      # Load data
69      camera = skimage.data.camera()
70      astronaut = rgb2gray(skimage.data.astronaut())
71      horse = skimage.data.horse()
72      coffee = rgb2gray(skimage.data.coffee())
73
74      # Marge data
75      data = [camera, astronaut, horse, coffee]
76
77      # Preprocessing
78      print("Start to data preprocessing...")
79      data = [preprocessing(d) for d in data]
80
81      # Create Hopfield Network Model
82      model = network.HopfieldNetwork()
83      model.train_weights(data)
84
85      # Generate testset
86      test = [get_corrupted_input(d, 0.3) for d in data]
87
88      predicted = model.predict(test, threshold=0, asyn=False)
89      print("Show prediction results...")
90      plot(data, test, predicted)
91      print("Show network weights matrix...")
92      #model.plot_weights()
93
94  if __name__ == '__main__':
95      main()
```

Лістинг А.3: Файл `train_mnist.py`

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Jul 29 08:40:49 2018
4
5  @author: user
```

```
"""

import numpy as np
from matplotlib import pyplot as plt
from skimage.filters import threshold_mean
import network
from keras.datasets import mnist

# Utils
def reshape(data):
    dim = int(np.sqrt(len(data)))
    data = np.reshape(data, (dim, dim))
    return data

def plot(data, test, predicted, figsize=(3, 3)):
    data = [reshape(d) for d in data]
    test = [reshape(d) for d in test]
    predicted = [reshape(d) for d in predicted]

    fig, axarr = plt.subplots(len(data), 3, figsize=figsize)
    for i in range(len(data)):
        if i==0:
            axarr[i, 0].set_title('Train data')
            axarr[i, 1].set_title("Input data")
            axarr[i, 2].set_title('Output data')

        axarr[i, 0].imshow(data[i])
        axarr[i, 0].axis('off')
        axarr[i, 1].imshow(test[i])
        axarr[i, 1].axis('off')
        axarr[i, 2].imshow(predicted[i])
        axarr[i, 2].axis('off')

    plt.tight_layout()
    plt.savefig("result_mnist.png")
    plt.show()

def preprocessing(img):
    w, h = img.shape
    # Thresholding
    thresh = threshold_mean(img)
    binary = img > thresh
    shift = 2*(binary*1)-1 # Boolian to int

    # Reshape
    flatten = np.reshape(shift, (w*h))
    return flatten
```

```
53
54 def main():
55     # Load data
56     (x_train, y_train), (_, _ )= mnist.load_data()
57     data = []
58     for i in range(3):
59         xi = x_train[y_train==i]
60         data.append(xi[0])
61
62     # Preprocessing
63     print("Start to data preprocessing...")
64     data = [preprocessing(d) for d in data]
65
66     # Create Hopfield Network Model
67     model = network.HopfieldNetwork()
68     model.train_weights(data)
69
70     # Make test datalist
71     test = []
72     for i in range(3):
73         xi = x_train[y_train==i]
74         test.append(xi[1])
75     test = [preprocessing(d) for d in test]
76
77     predicted = model.predict(test, threshold=50, asyn=True)
78     print("Show prediction results...")
79     plot(data, test, predicted, figsize=(5, 5))
80     print("Show network weights matrix...")
81     model.plot_weights()
82
83 if __name__ == '__main__':
84     main()
```