МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

НЕЙРОМЕРЕЖЕВА ОБРОБКА ДАНИХ

Методичні рекомендації
до лабораторних робіт
для здобувачів вищої освіти спеціальності
126 "Інформаційні системи та технології"
освітньої програми "Інформаційні системи та технології"
першого (бакалаврського) рівня

Харків
ХНЕУ ім. С. Кузнеця
2024

УДК 004.8(072.034)
Н46

Укладачі: С. Г. Удовенко
О. В. Гороховатський

Затверджено на засіданні кафедри інформатики та комп'ютерної техніки.
Протокол № 1 від 29.08.2023 р.

Самостійне електронне текстове мережеве видання

Нейромережева обробка даних [Електронний ресурс] : Н46 методичні рекомендації до лабораторних робіт для здобувачів вищої освіти спеціальності 126 "Інформаційні системи та технології" освітньої програми "Інформаційні системи та технології" першого (бакалаврського) рівня / уклад. С. Г. Удовенко, О. В. Гороховатський. – Харків : ХНЕУ ім. С. Кузнеця, 2024. – 58 с.

Подано методичні рекомендації до виконання лабораторних робіт із навчальної дисципліни, метою яких є закріплення знань і набуття практичних навичок із розроблення й навчання штучних нейронних мереж для вирішення завдань класифікації та кластеризації. Для кожної лабораторної роботи визначено мету, завдання, засоби й порядок виконання, зміст звіту та контрольні запитання.

Рекомендовано для здобувачів вищої освіти спеціальності 126 "Інформаційні системи та технології" освітньої програми "Інформаційні системи та технології" першого (бакалаврського) рівня.

УДК 004.8(072.034)

Вступ

Поширення реалізацій штучних нейронних мереж для вирішення практичних завдань професійної діяльності набуло значних масштабів в останні декілька років. Нейромережеві підходи обробки даних широко використовують для вирішення проблем кластеризації, класифікації та розпізнавання образів і мови, для прогнозування, для символьної обробки існуючих та генерації нових даних. Опанування цих методів забезпечує спеціаліста з інформаційних технологій потужним інструментарієм для вирішення практичних завдань.

Навчальна дисципліна "Нейромережева обробка даних" є обов'язковою навчальною дисципліною, яку вивчають згідно з навчальним планом підготовки здобувачів вищої освіти за спеціальністю 126 "Інформаційні системи та технології" освітньої програми "Інформаційні системи та технології" першого (бакалаврського) рівня усіх форм навчання.

Мета викладання навчальної дисципліни – формування в майбутніх фахівців системи компетентностей з питань застосування нейромережевої обробки даних для розв'язання складних спеціалізованих проблем прогнозування, кластеризації та класифікації даних у сфері використання інформаційних технологій.

Завданням навчальної дисципліни є засвоєння основних методів та способів вирішення проблем обробки даних із застосуванням нейромережевих підходів.

Предметом навчальної дисципліни є методи конструювання, навчання та застосування штучних нейронних мереж.

Об'єктом навчальної дисципліни є процес вирішення завдань обробки даних із використанням нейромережевих підходів.

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Нейромережева обробка даних", складені відповідно до затвердженої робочої програми, охоплюють практичні питання щодо застосування та дослідження штучних нейронних мереж для вирішення практичних завдань. Розроблено п'ять лабораторних робіт, спрямованих як на вивчення базових принципів навчання та функціонування штучних нейронних мереж, так і для вирішення завдань класифікації та кластеризації. Кожна лабораторна робота містить завдання експериментального характеру для набуття досвіду застосування відповідних методів та підходів. Оцінку за лабораторну роботу здобувач отримує за наявності звіту

з лабораторної роботи, виконаних завдань лабораторної роботи, розгорнутих відповідей на запитання викладача.

Результати навчання та компетентності, які формує навчальна дисципліна, визначено в табл. 1.

Таблиця 1

Результати навчання та компетентності, які формує навчальна дисципліна

Результати навчання	Компетентності, якими повинен оволодіти здобувач вищої освіти
ПР1	КС4, КС13
ПР6	КС1
ПР13	КС10, КС16

Примітка.

ПР1. Знати лінійну та векторну алгебру, диференціальне та інтегральне числення, теорію функцій багатьох змінних, теорію рядів, диференціальні рівняння для функції однієї та багатьох змінних, операційне числення, теорію ймовірностей та математичну статистику в обсязі, необхідному для розробки та використання інформаційних систем, технологій та інфокомунікацій, сервісів та інфраструктури організації.

ПР6. Демонструвати знання сучасного рівня технологій інформаційних систем, практичні навички програмування та використання прикладних і спеціалізованих комп'ютерних систем та середовищ з метою їх запровадження у професійній діяльності.

ПР13. Застосовувати нейромережеву обробку даних для розв'язання задач прогнозування, кластеризації та класифікації, здійснювати інтерпретацію результатів роботи побудованої моделі, виконувати аналіз якості, вдосконалювати модель.

КС1. Здатність аналізувати об'єкт проєктування або функціонування та його предметну область.

КС4. Здатність проєктувати, розробляти та використовувати засоби реалізації інформаційних систем, технологій та інфокомунікацій (методичні, інформаційні, алгоритмічні, технічні, програмні та інші).

КС10. Здатність вибору, проєктування, розгортання, інтегрування, управління, адміністрування та супроводжування інформаційних систем, технологій та інфокомунікацій, сервісів та інфраструктури організації.

КС13. Здатність проводити обчислювальні експерименти, порівнювати результати експериментальних даних і отриманих рішень.

КС16. Здатність створювати та використовувати моделі штучних нейронних мереж для розв'язання прикладних задач обробки даних.

Лабораторна робота 1
Вирішення завдання бінарної класифікації одношаровим персептроном

Мета роботи: ознайомитися з методом бінарної класифікації даних за допомогою одношарового персептрону, дослідити залежність моделі персептрону від гіперпараметрів.

Теоретична частина

Класифікація персептроном – це один із найпростіших методів машинного навчання (МН). Модель одношарового персептрону може бути використана для вирішення завдання бінарної класифікації, де треба віднести об'єкт (заданий сукупністю числових ознак) до одного з двох відомих класів (наприклад, прогнозування того, чи має особа захворювання, на основі числових значень, таких як показники аналізів крові).

Розуміння того, як працює класифікація з використанням персептрону, часто вважають фундаментальним знанням для спеціалістів МН. Воно містить важливі прийоми, використовувані іншими методами МН, логістичною регресією та класифікацією із застосуванням глибоких нейронних мереж. Насправді найпростіший тип нейронної мережі часто називають багатошаровим персептроном.

Крім того, розуміння того, як працює класифікація з використанням моделі персептрону під час програмування системи "з нуля", дозволяє зрозуміти сильні та слабкі сторони цього методу на випадок, якщо ви зіткнетеся з ним у готовій реалізації.

Щоб зрозуміти принцип класифікації одношаровим персептроном розглянемо такий приклад (рис. 1). Маємо набір точок у двовимірному просторі, кожна з яких характеризується координатами X та Y і належить до одного з двох класів: із міткою −1 або +1. Необхідно створити модель, яка буде визначати клас точки за її координатами.

Розуміння даних

У прикладі використаємо набір із 50 точок. У різних практичних завданнях мітки класів можна кодувати по-різному, наприклад 0 та 1. Для

класифікації персептроном зручніше кодувати дві можливі мітки –1 та +1 замість 0 та 1.

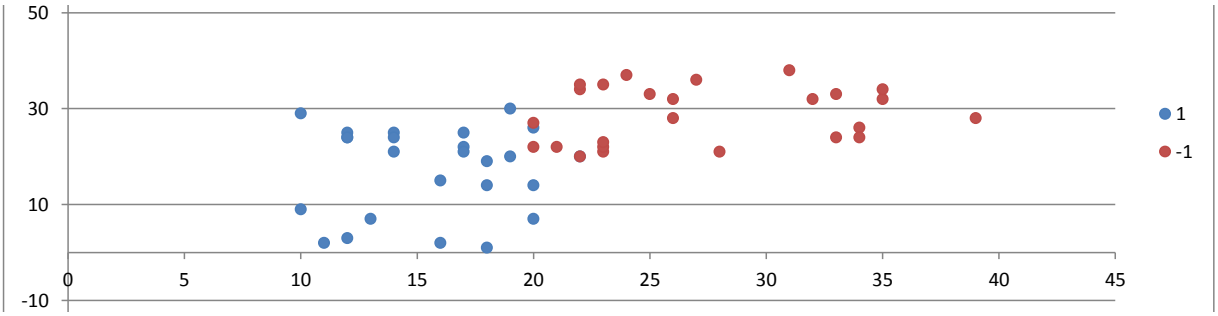

Рис. 1. **Початкові дані**

Ключовим моментом є те, що одношаровий персептрон добре працює лише з даними, які можна розділити лінійно. Для лінійно роздільних даних можна намалювати лінію (або гіперплощину для трьох або більше вимірів), яка розділяє дані такими чином, щоб усі елементи одного класу знаходилися з одного боку лінії, а елементи іншого класу – з другого. На рис. 1 видно, що лінією можна достатньо вдало розділити два вихідних класи. Основною проблемою є те, що не завжди відомо заздалегідь, чи є дані лінійно роздільними, чи ні.

Розуміння того, як працює класифікація персептроном

Класифікація персептроном достатньо проста. Для набору даних із n ознак (в нашому прикладі маємо дві ознаки: X та Y) необхідно знайти n вагових коефіцієнтів і одне спеціальне значення, яке називають зміщенням (bias). Вагові коефіцієнти (ваги) та зміщення – це просто числові константи. Для того щоб обчислити прогнозоване значення, потрібно підсумувати добутки кожного значення ознаки та пов'язаного з нею значення вагового коефіцієнта, а потім до результату додати зміщення. Якщо сума від'ємна, то такий набір ознак буде мати мітку –1, якщо ні – тоді мітка класу має бути +1.

Найскладніша частина полягає саме у визначенні вагових коефіцієнтів та значення зміщення для моделі класифікатора. Процес пошуку відповідних значень називаю навчанням моделі. Навчання – це ітераційний процес, який змінює значення вагових коефіцієнтів моделі та зміщення,

поки обчислювані результати не будуть збігатися з відомими правильними значеннями класу в навчальних даних.

Спосіб обчислення результатів класифікації персептроном передбачає нормалізацію навчальних даних для того, щоб невеликі значення вхідних ознак не були приглушені великими значеннями інших знак. У нашому прикладі нормалізацію даних не використовують, оскільки всі значення ознак (координат) перебувають приблизно в одному діапазоні. Трьома найпоширенішими методами нормалізації є нормалізація min-max, нормалізація z-оцінки (стандартизована оцінка) та нормалізація порядку величини (min-max normalization, z-score normalization, order of magnitude normalization).

Практична частина

Для створення прикладу було розроблено консольний проєкт (.NET Core або .NET Framework) у Visual Studio 2019 мовою C#. Замість Visual Studio можна використати онлайн-компілятор, який може працювати з файлами із вхідними даними, наприклад, https://www.jdoodle.com/compile-c-sharp-online/. У прикладі не використано додаткових бібліотек, тож його можна реалізувати із застосуванням багатьох мов програмування та іншого програмного забезпечення. Використано стандартні простори імен System, System.Globalization, System.IO.

Структуру головного методу програми наведено на рис. 2. Цей код містить:

- інформаційні повідомлення про вирішуване завдання у рядках 11 – 12;
- читання навчальних даних із файлу в рядках 14 – 16;
- установлення параметрів навчання персептрону (максимальна кількість ітерацій maxIter і швидкість навчання learningRate) у рядках 18 – 19. Обидва ці значення є гіперпараметрами, які необхідно встановлювати з практичних міркувань, результатів експериментів та загальних рекомендацій. Швидкість навчання впливає на те, наскільки суттєво змінюються вагові коефіцієнти і зміщення на кожній ітерації навчання. Якщо встановити її занадто малою, модель буде навчатися дуже повільно, якщо занадто великою – оптимальне рішення можна пропустити в процесі навчання й отримати недостатньо якісну модель;

- запуск процедури навчання в рядку 22;
- обчислення та друк точності навченої моделі на тренувальному наборі в рядках 26 – 28;
- роздруківка отриманих вагових коефіцієнтів у рядках 30 – 33;
- класифікація точки, яка не належить навчальному набору, у рядках 35 – 38.

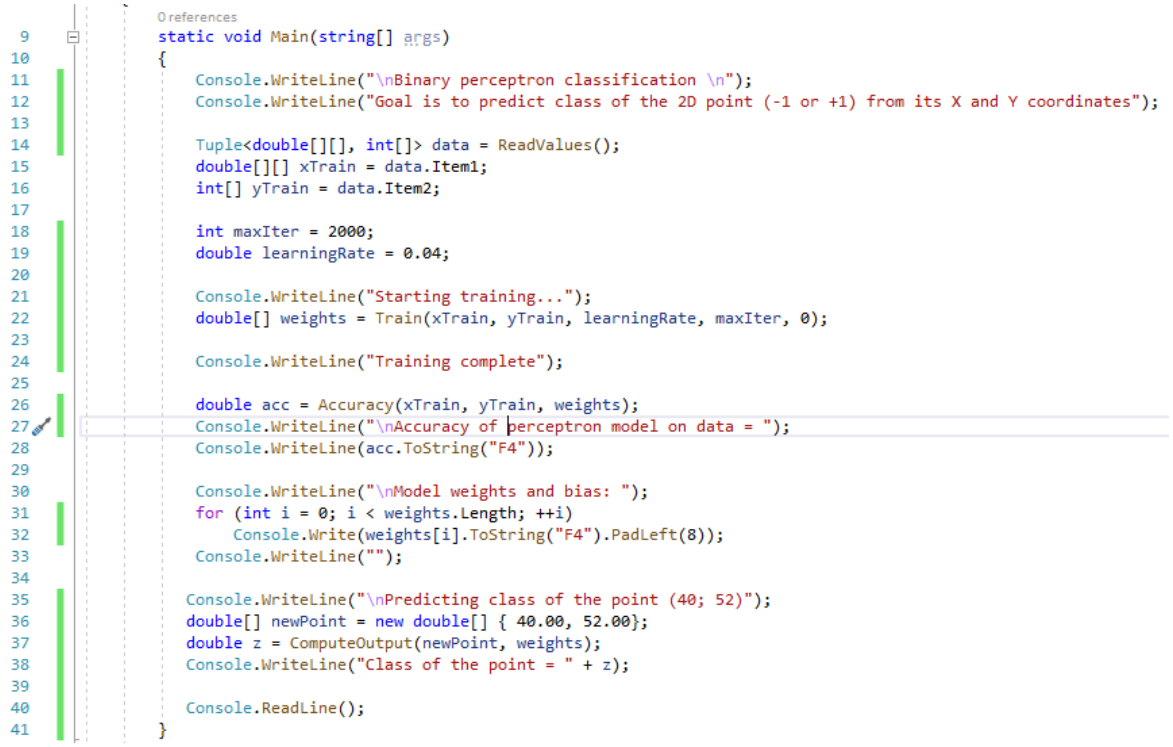

Рис. 2. **Код головного методу програми**

На рис. 3 наведено код методу ReadValues для завантаження вихідних даних для навчання. Дані містяться у файлі "Book1.csv", який повинен розташовуватися в одній директорії з файлом застосунку. Формат даних має вигляд:

50,2
18,14,1
12,3,1
…

Перший рядок містить кількість навчальних прикладів (50 точок) та розмірність прикладів. У нашому випадку кожна точка вимірюється двома значеннями: X та Y. У другому та наступних рядках розміщено елементи навчального набору, які складаються з координат і мітки класу. Значення "18,14,1" означає, що точка (18; 14) належить до класу з міткою "1".

```
1 reference
143 static Tuple<double[][], int[]> ReadValues()
144 {
145     String input = File.ReadAllText("Book1.csv");
146     string[] rows = input.Split('\n');
147
148     string[] firstRow = rows[0].Split(',');
149     int items = int.Parse(firstRow[0]);
150     int dimensions = int.Parse(firstRow[1]);
151
152     double[][] result = new double[items][];
153     int[] y = new int[items];
154
155     for (int i = 1; i < rows.Length; i++)
156     {
157         result[i - 1] = new double[dimensions];
158
159         string[] col = rows[i].Split(',');
160
161         for (int j = 0; j < col.Length - 1; j++)
162         {
163             result[i - 1][j] = double.Parse(col[j], CultureInfo.InvariantCulture);
164         }
165
166         y[i - 1] = int.Parse(col[col.Length - 1], CultureInfo.InvariantCulture);
167     }
168
169     return new Tuple<double[][], int[]>(result, y);
170 }
```

Рис. 3. **Код методу ReadValues**

На рис. 4 наведено реалізацію навчання персептрону. У рядку 61 створюється об'єкт Random, який використовують для сортування елементів навчального набору (перемішування порядку класів з мітками 1 та –1) випадковим чином. Таке перемішування даних загалом дозволяє мережі навчатися краще, оскільки на кожному етапі вона буде задіювати різноманітні нейрони для різних класів. Також зменшується ймовірність мінімізувати функцію навчання лише для одного класу, звівши її значення до локального мінімуму через навчання послідовних представників цього класу. Тож на кожній ітерації весь тренувальний набір перемішується (у рядку 71).

```
55      static double[] Train(double[][] xData, int[] yData, double lr, int maxIter, int seed)
56      {
57          int N = xData.Length;
58          int n = xData[0].Length;
59          double[] weights = new double[n + 1]; // one more weight for bias
60          int[] indices = new int[N];
61          Random rnd = new Random(seed);  // random value sets the order of training
62          int iter = 0;
63
64          for (int i = 0; i < N; i++)
65          {
66              indices[i] = i;
67          }
68
69          while (iter < maxIter)
70          {
71              Shuffle(indices, rnd);
72              foreach (int i in indices)
73              {
74                  int output = ComputeOutput(xData[i], weights);
75                  int target = yData[i];
76
77                  if (output != target)
78                  {
79                      double delta = target - output;
80
81                      for (int j = 0; j < n; ++j)
82                      {
83                          weights[j] = weights[j] + (lr * delta * xData[i][j]);
84                      }
85
86                      weights[n] = weights[n] + (lr * delta * 1);
87                  }
88              }
89              ++iter;
90          }
91
92          return weights;
93      }
```

Рис. 4. **Код методу Train**

Головну частину процесу навчання реалізовано в рядках 72 – 89. У ній усі елементи начального набору розглядаються в довільному порядку (завдяки попередньому перемішуванню). Метод ComputeOutput у рядку 74 обчислює результат класифікації поточного об'єкта навчального набору. Математично обчислення результату цього методу описують у такому вигляді:

$$\text{output} = \begin{cases} -1 \text{ якщо } \left(\sum_{i=0}^{N} x_i w_i + b\right) \le 0, \\ 1 \text{ в іншому випадку,} \end{cases}$$

де N – кількість ознак (в нашому випадку дві), x – вхідний навчальний об'єкт, w – поточні ваги моделі, b – поточне зміщення.

У рядку 75 міститься правильне значення класу для поточного навчального об'єкта. Якщо поточне значення класу (обчислене з використанням коефіцієнтів, які навчаються) не збігається з очікуваним (ря-

док 77), вагові коефіцієнти та зміщення змінюються відповідно до дельта-правила:

$$w_i = w_i + x_i learningRate\left(y_i^{target} - y_i\right),$$

$$b = b + learningRate\left(y_i^{target} - y_i\right),$$

де y_i^{target} – очікуваний клас, y_i – поточний клас.

Код методів Shuffle (алгоритм Фішера-Йєтса) та ComputeOutput наведено на рис. 5.

```
3 references
static int ComputeOutput(double[] x, double[] weights)
{
    double z = 0.0;
    for (int i = 0; i < x.Length; ++i)
        z += x[i] * weights[i];
    z += weights[weights.Length - 1];

    if (z < 0.0)
    {
        return -1;
    }
    else
    {
        return 1;
    }
}
```

```
1 reference
static void Shuffle(int[] indices, Random rnd)
{
    int n = indices.Length;
    for (int i = 0; i < n; ++i)
    {
        int ri = rnd.Next(i, n);
        int tmp = indices[ri];
        indices[ri] = indices[i];
        indices[i] = tmp;
    }
}
```

Рис. 5. **Код методів Shuffle та ComputeOutput**

Метод Accuracy (рис. 6) обчислює точність моделі за заданими ваговими коефіцієнтами та тренувальним набором. Точність у цьому випадку – це кількість елементів тренувального набору, які в результаті класифікації отримали правильну мітку класу.

```
1 reference
static double Accuracy(double[][] xData, int[] yData, double[] weights)
{
    int numCorrect = 0; int numWrong = 0;
    int N = xData.Length;
    for (int i = 0; i < N; ++i)
    {
        double[] x = xData[i];
        int target = yData[i];
        int computed = ComputeOutput(x, weights);
        if (target == computed)
        {
            ++numCorrect;
        }
        else
        {
            ++numWrong;
        }
    }
    return (1.0 * numCorrect) / (numCorrect + numWrong);
}
```

Рис. 6. **Код методу Accuracy**

Якщо ми запустимо програму на виконання, то повинні отримати результат, подібний до наведеного на рис. 7. Тут точність становила 92 %, що є задовільним показником. Це означає, що модель неправильно розрізняє лише 4 точки з 50, на яких вона тренувалася.

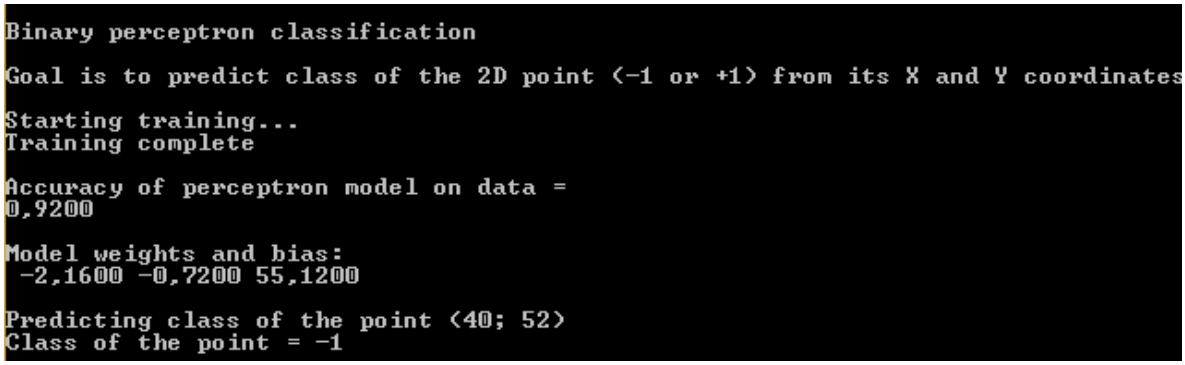

Рис. 7. **Результат роботи програми**

Цей результат показує вагові коефіцієнти –2.16, –0.72, а також те, що значення зміщення дорівнює 55.12. Повертаючись до їх інтерпретації, необхідно згадати, що одношаровий персептрон здатний розв'язувати лише лінійно роздільні задачі (їх також називають лінійно сепарабельними), чому і було присвячено наш приклад. Наша задача двовимірна (кожна точка характеризується координатами X та Y), тому інтерпретація вагових коефіцієнтів і зміщення проста – це і є формула власне лінії розділення наших класів із мітками –1 та 1 у двовимірному просторі. Отже, лінію можна записати у вигляді:

$$b + w_1 x + w_2 y = 0,$$

звідки

$$y = -(b + w_1 x) / w_2$$

при $b = 55.12$, $w_1 = -2.16$, $w_2 = -0.72$.

На рис. 8 показано побудовану лінію розділення та набір тренувальних даних. Тепер можна візуально побачити три синіх точки навчального набору, які потрапили в "червону" частину площини. Четверта точка, яку навчена модель не вміє класифікувати правильно, знаходиться за координатами (22; 20), візуально має червоний вигляд (клас із міт-

кою –1), але в навчальному наборі цю точку було віднесено до обох класів одночасно. Тож навчальний набір містив неоднозначний елемент, який із самого початку унеможливлював отримання 100 % точності навчання мережі на цьому наборі. Цей приклад підкреслює важливість попередньої обробки чи валідації навчальних даних.

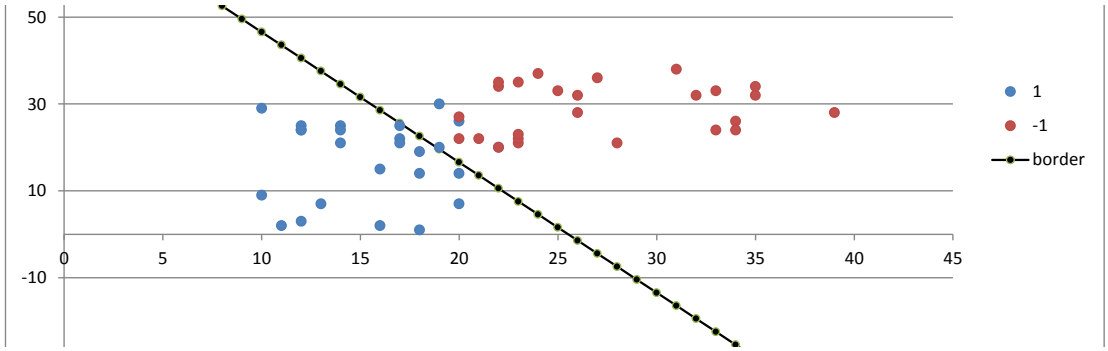

Рис. 8. **Візуалізація лінії розділення**

В останній частині роздрукованих результатів на рис. 7 можна побачити класифікацію точки з координатами (40; 52). Відповідно до рис. 8 вона знаходиться в "червоній" частині праворуч від лінії розділення. Точки цієї частини мають мітку –1, якою наша навчена модель правильно позначила цей тестовий приклад.

Індивідуальні завдання

1. Повторіть наведений приклад на даних, згенерованих самостійно. Обчисліть точність, візуалізуйте вихідні дані, лінію розділення. Використання першоджерела за посиланням із деякими початковими частинами коду (https://visualstudiomagazine.com/Articles/2020/01/07/perceptron-classification.aspx?Page=1) може бути корисним.

За домовленістю з викладачем також можна розв'язувати іншу задачу бінарної класифікації, наприклад:

Banknote authentication
https://archive.ics.uci.edu/ml/datasets/banknote+authentication,
Haberman's Survival
https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival,
Breast Cancer Wisconsin
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic),

Heart Disease

https://archive.ics.uci.edu/ml/datasets/heart+disease.

2. Порівняйте точність моделей залежно від вибраних гіперпараметрів (максимальна кількість ітерацій та швидкість навчання).

3. Дослідіть точність моделі залежно від параметру Seed функції Train.

4. Реалізуйте зменшення параметру швидкості навчання зі збільшенням поточної ітерації, наприклад, зробити його меншим після тисячної ітерації. Чи дозволяє це отримати кращі результати?

5. Адаптуйте код програми для випадку, коли вихідні класи мають мітки 0 та 1, а не −1 та 1. Протестуйте результат.

Підготуйте звіт за результатами виконаної роботи. Звіт має містити титульний аркуш; опис виконаних завдань із коментарями, кодом, скриншотами; висновки з кожного завдання та з лабораторної роботи загалом. Висновки не повинні дублювати мету.

Підготуйтеся до захисту роботи за контрольними запитаннями. Запитання, позначені символом *, передбачають вивчення додаткової літератури.

Контрольні запитання до лабораторної роботи 1

1. Що таке бінарна класифікація?
2. Дайте визначення розмірності даних.
3. Які задачі можна розв'язувати одношаровим персептроном?
4. Які дані називають лінійно роздільними?
5. Що таке дельта-правило?
6. Навіщо перемішувати вхідні дані?
7. Яким чином вибрати параметр швидкості навчання?
8. Як обчислюють вихідне значення персептрону?
9. Навіщо потрібна нормалізація вихідних даних?
10*. Яку роль відіграє параметр зміщення?
11*. Навіщо обмежувати максимально можливу кількість ітерацій?
12*. Чи можна одношаровим персептроном розв'язувати задачі багатокласової класифікації?

Лабораторна робота 2
Кластеризація даних із використанням самоорганізованих карт Кохонена

Мета роботи: ознайомитися із методом кластеризації даних із використанням самоорганізованих карт Кохонена, а також методом кластеризації K-means.

Теоретична частина

Самоорганізовані карти Кохонена (СОК) – це відносно простий метод (об'єкт) машинного навчання (МН). Проте СОК досить важко описати, оскільки існує багато варіацій, а також тому, що СОК мають характеристики, які нагадують деякі інші методи МН, зокрема неконтрольовану кластеризацію та контрольовану класифікацію. Можна інтерпретувати СОК як своєрідний дослідницький аналіз кластеризації, метою якого є віднесення кожного з елементів даних, які схожі один на одного, до одного й того самого вузла карти.

Розглянемо, як працює СОК, на прикладі, першоджерело якого наведено за посиланням: https://docs.microsoft.com/en-us/archive/msdn-magazine/2019/january/test-run-self-organizing-maps-using-csharp#the-demo-program.

Використаємо набір даних рукописних цифр із репозиторію UCI: https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits. Виберемо тестовий набір даних (optdigits.tes) для прикладу.

Вибраний набір містить 1 797 векторів, кожен вектор містить 65 значень. 64 значення відповідають пікселям зображення символу розміром 8×8, кожне значення перебуває в діапазоні від 0 до 16. На останньому місці в цьому векторі знаходиться мітка класу, до якого належить вектор, від 0 до 9 включно.

Результат роботи методу самоорганізованих карт може виглядати так, як показано на рис. 9. Результат аналізу СОК зазвичай візуальний і призначений для аналізу людиною. Фактично він є відображенням вихідного 64-вимірного вектора у двовимірний простір розміром 5×5. Розмір карти зазвичай визначають експериментальним шляхом.

Рис. 9. **Можлива візуалізація СОК**

Кожну клітинку карти СОК називають вузлом. Кожен вузол містить узагальнене векторне значення всіх векторів, які віднесено до цього вузла. Із кожним вузлом також може бути асоційовано мітку класу, якщо він є в навчальному наборі, наприклад, це може бути мітка класу, що найчастіше трапляється в цьому вузлі.

СОК створюють так, що кожен вектор вузла є репрезентативним для деяких елементів даних, а також для того, щоб вузли карти, які знаходяться близько один до одного, геометрично представляли схожі елементи даних. Загальний алгоритм кластеризації даних методом самоорганізованих карт Кохонена такий:

1. Створити випадкову карту.

2. Поки номер поточної ітерації менший за максимальну кількість ітерацій:

- установити максимальний радіус близькості, швидкість навчання для поточної ітерації;
- вибрати випадковим чином навчальний вектор;
- знайти найближчий до нього вектор у вузлі карти;
- оновити вектор вузла карти та його сусідів у заданому радіусі зі встановленою швидкістю;
- перейти на наступну ітерацію.

Практична частина

Розглянемо демонстраційну програму та реалізацію методу самоорганізованих карт.

Головним є клас Map, методи якого ми будемо викликати для побудови карти. Цей клас містить такі поля (рис. 10): rnd буде використано під час навчання для заповнення перших випадкових значень; rows, cols містять розмір карти. Значення rangeMax містить максимальну відстань між вузлами карти встановленого розміру для вибраної відстані. У цьому прикладі використано Манхеттенську відстань, максимальне значення якої на карті дорівнює сумі вимірювань карти, тобто 10. LearnRateMax

зберігає максимальну швидкість навчання, яка буде поступово зменшуватися, dimensions – розмірність кожного вхідного вектора, dataItemsCount – загальну кількість навчальних векторів.

```
public class Map
{
    Random rnd = new Random(0);
    int rows, cols = 5;
    int rangeMax = 10;
    double LearnRateMax = 0.5;
    int StepsMax = 100000;
    int dimensions = 0;
    int dataItemsCount;

    public int[] labels;
    public double[][] data;
    public double[][][] map;  // [row][column][vector]
    public List<int>[][] mapping;
```

Рис. 10. **Поля класу Map**

Масив labels містить мітки всіх класів (від 0 до 9) для кожного навчального вектора, data містить усі навчальні вектори, map буде містити карту з узагальненим вектором у кожному вузлі, mapping буде містити карту з відображенням міток класів у кожному вузлі.

Конструктор класу наведено на рис. 11. У ході створення задають розмірність (dimension=64), розміри карти (mapSize=5), шлях до файлу з навчальними прикладами, а також дві опції структури цього файлу: чи знаходиться мітка класу на першій або останній позиції, а також чи містить файл технічну інформацію в першому рядку.

```
1 reference
public Map(int dimensions, int mapSize, string pathToFile, bool skipFirstLine = false, bool labelsFirst = false)
{
    this.rows = this.cols = mapSize;
    this.rangeMax = this.rows + this.cols;
    this.dimensions = dimensions;
    this.Initialize();
    this.ReadData(pathToFile, skipFirstLine, labelsFirst);
    this.NormalisePatternsMax();
}
```

Рис. 11. **Конструктор класу Map**

Отже, для нашого прикладу ми можемо створити такий екземпляр класу Map:

Map SOMmap = new Map(64, 5, "optdigits.tes").

Розглянемо, що відбуватиметься під час створення цього екземпляру (рис. 12).

Метод Initialize() заповнює початкову карту випадковими векторами потрібної розмірності зі значеннями від 0 до 1.

```
1 reference
private void Initialize()
{
    map = new double[this.rows][][];  // [row][column][vector]
    for (int i = 0; i < this.rows; ++i)
    {
        map[i] = new double[this.cols][];
        for (int j = 0; j < this.cols; ++j)
        {
            map[i][j] = new double[this.dimensions];
            for (int k = 0; k < this.dimensions; ++k)
                map[i][j][k] = rnd.NextDouble();
        }
    }
}
```

Рис. 12. **Ініціалізація карти випадковими векторами**

Метод ReadData() заповнює масив із даними data-векторами, які зчитуються з файлу (рис. 13).

```
private void ReadData(string path, bool skipFirstLine=false, bool labelsFirst = false)
{
    string[] lines = File.ReadAllLines(path);
    this.dataItemsCount = skipFirstLine? lines.Length - 1 : lines.Length;
    string[] tokens;
    data = new double[this.dataItemsCount][];

    for (int i = 0; i < this.dataItemsCount; ++i)
    {
        data[i] = new double[this.dimensions];
    }

    labels = new int[this.dataItemsCount];

    for (int i =0; i<lines.Length; i++)
    {
        string line = lines[i];

        if (skipFirstLine && i == 0)
        {
            continue;
        }

        tokens = line.Split(',');
        for (int j = 0; j < this.dimensions; ++j)
        {
            data[i][j] = double.Parse(tokens[j]);
        }

        if (labelsFirst)
        {
            labels[i] = int.Parse(tokens[0]); // label is in the first position
        }
        else
        {
            labels[i] = int.Parse(tokens[this.dimensions]); // label is in the last position
        }
    }
}
```

Рис. 13. **Код методу для читання даних**

Метод NormalizePatternsMax використовують для нормалізації даних векторів (рис. 14). У нашому випадку всі вони діляться на 16 – максимальне значення, яке відоме нам з опису датасету.

```
public void NormalisePatternsMax()
{
    for (int j = 0; j < dimensions; j++)
    {
        for (int i = 0; i < dataItemsCount; i++)
        {
            data[i][j] = data[i][j] / 16.0;
        }
    }
}
```

Рис. 14. **Нормалізація даних**

Тепер можна навчати карту. Головний метод Train наведено на рис. 15.

```
public void Train()
{
    for (int s = 0; s < StepsMax; ++s)
    {
        if (s % (int)(StepsMax / 5) == 0 && s > 0)
        {
            Console.WriteLine("step = " + s);
        }

        double pctLeft = 1.0 - ((s * 1.0) / StepsMax);
        int currRange = (int)(pctLeft * this.rangeMax);
        double currLearnRate = pctLeft * LearnRateMax;

        // Pick random data index
        int t = rnd.Next(0, this.dataItemsCount);

        // Get (row,col) of closest map node -- 'bmu'
        int[] bmuRC = ClosestNode(this.data, t, this.map);

        // Move each map mode closer to the bmu
        for (int i = 0; i < this.rows; ++i)
        {
            for (int j = 0; j < this.cols; ++j)
            {
                if (ManDist(bmuRC[0], bmuRC[1], i, j) <= currRange)
                {
                    for (int k = 0; k < this.dimensions; ++k)
                    {
                        this.map[i][j][k] = this.map[i][j][k] + currLearnRate * (this.data[t][k] - this.map[i][j][k]);
                    }
                }
            }
        }
    }
}
```

Рис. 15. **Метод навчання карти**

Змінна pctLeft обчислює відсоток кроків, які залишилося виконати. Наприклад, якщо для StepsMax установлено 100, а змінна лічильника поточного циклу s = 33, то відсоток кроків, що залишилися, дорівнює 67 %. Максимальну відстань currRange до сусідів для кроку s обчислюють з використанням цього відсотка. Змінна currLearnRate обчислює поточне значення швидкості навчання, яке поступово зменшується з початкового максимального значення LearnRateMax.

Беремо випадковий навчальний вектор t і знаходимо для нього Best Matching Unit (BMU) – найбільш близький у рамках Евклідової метрики вузол карти (рис. 16 – 18).

Після цього перераховується вся карта. Для кожного з вузлів карти на відстані, яка є меншою за максимально дозволену на цьому кроці, змінюються узагальнені вектори зі швидкістю, установленою на цьому етапі навчання. Значення узагальнювальних векторів для всіх цих вузлів наближаються до значення поточного навчального вектора.

```
1 reference
private static int[] ClosestNode(double[][] data, int t, double[][][] map)
{
    // Coords in map of node closest to data[t]
    double smallDist = double.MaxValue;
    int[] result = new int[] { 0, 0 };  // (row, col)
    for (int i = 0; i < map.Length; ++i)
    {
        for (int j = 0; j < map[0].Length; ++j)
        {
            double dist = EucDist(data[t], map[i][j]);
            if (dist < smallDist)
            {
                smallDist = dist; result[0] = i; result[1] = j;
            }
        }
    }
    return result;
}
```

Рис. 16. **Метод пошуку найближчого вузла**

```
static double EucDist(double[] v1, double[] v2)
{
    double sum = 0;
    for (int i = 0; i < v1.Length; ++i)
    {
        sum += (v1[i] - v2[i]) * (v1[i] - v2[i]);
    }

    return Math.Sqrt(sum);
}
```

Рис. 17. **Евклідова відстань**

```
static int ManDist(int x1, int y1, int x2, int y2)
{
    return Math.Abs(x1 - x2) + Math.Abs(y1 - y2);
}
```

Рис. 18. **Манхеттенська відстань**

Після того, як навчання закінчено, можна викликати метод AssignIndices() (наведений разом із допоміжним методом на рис. 19 та 20), який знайде для кожного навчального вектора найближчий вузол карти.

```
public void AssignIndices()
{
    mapping = new List<int>[this.rows][];

    for (int i = 0; i < this.rows; ++i)
        mapping[i] = new List<int>[this.cols];
    for (int i = 0; i < this.rows; ++i)
        for (int j = 0; j < this.cols; ++j)
            mapping[i][j] = new List<int>();

    for (int t = 0; t < this.dataItemsCount; ++t)  // each data item
    {
        // Find node map coords where node is closest to D(t)
        int[] rc = Map.ClosestNode(this.data[t], this.map);

        int r = rc[0]; int c = rc[1];
        mapping[r][c].Add(t);
    }
}
```

Рис. 19. **Призначення індексів**

```
private static int[] ClosestNode(double[] dataItem, double[][][] map)
{
    // Coords in map of node closest to data[t]
    double smallDist = double.MaxValue;
    int[] result = new int[] { 0, 0 };  // (row, col)
    for (int i = 0; i < map.Length; ++i)
    {
        for (int j = 0; j < map[0].Length; ++j)
        {
            double dist = EucDist(dataItem, map[i][j]);
            if (dist < smallDist)
            {
                smallDist = dist; result[0] = i; result[1] = j;
            }
        }
    }
    return result;
}
```

Рис. 20. **Метод пошуку найближчого вузла**

Для візуалізації результату можна обчислити та надрукувати для кожного з вузлів найчастішу мітку класу з усіх векторів, приписаних до цього вузла (рис. 21, 22).

```
for (int i = 0; i < size; ++i)
{
    for (int j = 0; j < size; ++j)
    {
        List<int> members = new List<int>();  // '0'- '9'
        foreach (int idx in SOMmap.mapping[i][j])
            members.Add(SOMmap.labels[idx]);
        int mcv = MostCommonVal(members);
        Console.Write(mcv + "  ");
    }
    Console.WriteLine("");
}
```

Рис. 21. **Обчислення найчастішої мітки**

```
static int MostCommonVal(List<int> list)
{
    if (list.Count == 0) return -1;
    int largestCount = 0; int mostCommon = 0;
    int[] counts = new int[10];
    foreach (int val in list)
    {
        ++counts[val];
        if (counts[val] > largestCount)
        {
            largestCount = counts[val];
            mostCommon = val;
        }
    }
    return mostCommon;
}
```

Рис. 22. **Метод обчислення частоти**

Якщо дані не мають міток класів, одним із поширених прийомів є створення U-матриці, яка становить сітку n-на-m, де значення кожного вузла – середня відстань між відповідним вектором вузла СОК та векторами сусідніх вузлів. U-матриця зазвичай відображається так, що кожне значення інтерпретують як значення пікселів у градаціях сірого.

Іншою можливістю відображення СОК, створеної з даних без міток, є зіставлення кожного вектора вузла з кольором. Наприклад, якщо вектор вузла СОК має розмір шість, ви забарвлюєте кожну клітинку СОК за допомогою колірної моделі RGB, призначаючи середнє значення перших двох векторних значень R, другі два значення – G, а два останні значення – B. У демонстраційній програмі вектори вузлів СОК мають розмір 64, тому немає очевидного способу пов'язати кожен вектор із кольором.

K-Means

Найбільш поширеним алгоритмом кластеризації даних є метод k-середніх (k-means). Цей алгоритм належить до навчання без учителя та розподіляє дані на k кластерів за подібністю навчальних векторів.

Алгоритм складається з таких етапів:

1. Випадково обрати k об'єктів як центри кластерів.
2. Розподілити всі об'єкти до найближчого кластера.
3. Розрахувати новий центр кожного кластера як середнє арифметичне об'єктів, що в нього входять.
4. Повторювати попередні два кроки, поки центри кластерів не стануть стійкими.

Індивідуальні завдання

Підготуйте звіт за результатами виконаної роботи. Звіт має містити титульний аркуш; опис виконаних завдань із коментарями, кодом, скриншотами; висновки з кожного завдання та з лабораторної роботи загалом. Висновки не повинні дублювати мету.

Підготуйтеся до захисту роботи за контрольними запитаннями. Запитання, позначені символом *, передбачають вивчення додаткової літератури.

1. Реалізуйте наведений приклад. Порівняйте результати із наведеними.

2. Побудуйте декілька карт більшого розміру. Проаналізуйте результати.

3. Адаптуйте приклад для обробки набору даних "Іриси Фішера" (дані можна взяти тут: https://archive.ics.uci.edu/ml/datasets/iris або пошукати в csv-форматі). Проаналізуйте результати.

4. Реалізуйте метод k-means для кластеризації набору даних "Іриси Фішера". Проаналізуйте результати.

Контрольні запитання до лабораторної роботи 2

1. Для чого використовують самоорганізовані карти Кохонена?
2. Назвіть особливості й тип навчання, які використовуються в СОК.
3. Що таке Best Matching Unit?
4. Опишіть, яким чином змінюється швидкість навчання.
5*. Назвіть недоліки та переваги СОК.
6. Що таке кластеризація даних?
7. Для чого використовують k-means?
8*. Назвіть недоліки та переваги k-means.
9. Опишіть алгоритм реалізації k-means.
10*. Назвіть і опишіть методи кластеризації даних, окрім k-means.

Лабораторна робота 3
Бінарна класифікація текстових даних

Мета роботи: ознайомитися з засобами обробки текстових даних та бінарною класифікацією текстів із використанням Google Colab.

Теоретична частина

Google Colaboratory (Google Colab) – безкоштовний "хмарний" сервіс від Google, який дає змогу розробляти нейромережеві застосунки на Python із застосуванням Keras, OpenCV, Tensorflow та PyTorch і навчати нейронні мережі на GPU.

Colaboratory дозволяє писати й виконувати код мовою Python у веббраузері без попередніх налаштувань, з безкоштовним доступом до графічних процесорів, з легким спільним доступом.

Colab застосовують у сфері машинного навчання для таких завдань:

- ознайомлення з TensorFlow;
- розроблення й навчання нейронних мереж;
- експерименти з тензорними процесорами;
- поширення досліджень у сфері штучного інтелекту;
- створення навчальних посібників.

Приклади цікавих записників Colab із застосуванням машинного навчання можна переглянути тут: https://colab.research.google.com/notebooks/intro.ipynb?hl=uk#using-accelerated-hardware=.

Налаштування Google Colaboratory

Google Colab працює в середовищі Google Drive. Програми пишуть мовою python у файлах, що називають записниками (Jupyter notebook). Докладніше про проєкт Jupyter можна дізнатися на сайті jupyter.org.

Записник – це не статична вебсторінка, а інтерактивне середовище, яке дає змогу писати й виконувати код. У записниках Colab можна поєднувати виконуваний код і форматований текст в одному документі, а також додавати зображення, HTML, LaTeX тощо. Коли ви створюєте власні записники Colab, вони зберігаються в обліковому записі Google-диска. Ви можете надавати доступ до записників Colab співробітникам або друзям (так само, як і до інших документів, що зберігаються на Google Drive) – так вони зможуть коментувати або навіть редагувати їх.

Перед початком роботи в Colab зручно створити окрему папку на диску для групування всіх записників (рис. 23). Це можна зробити, натиснувши кнопку "Створити", вибрати в наступному вікні "Папка" та дати їй ім'я.

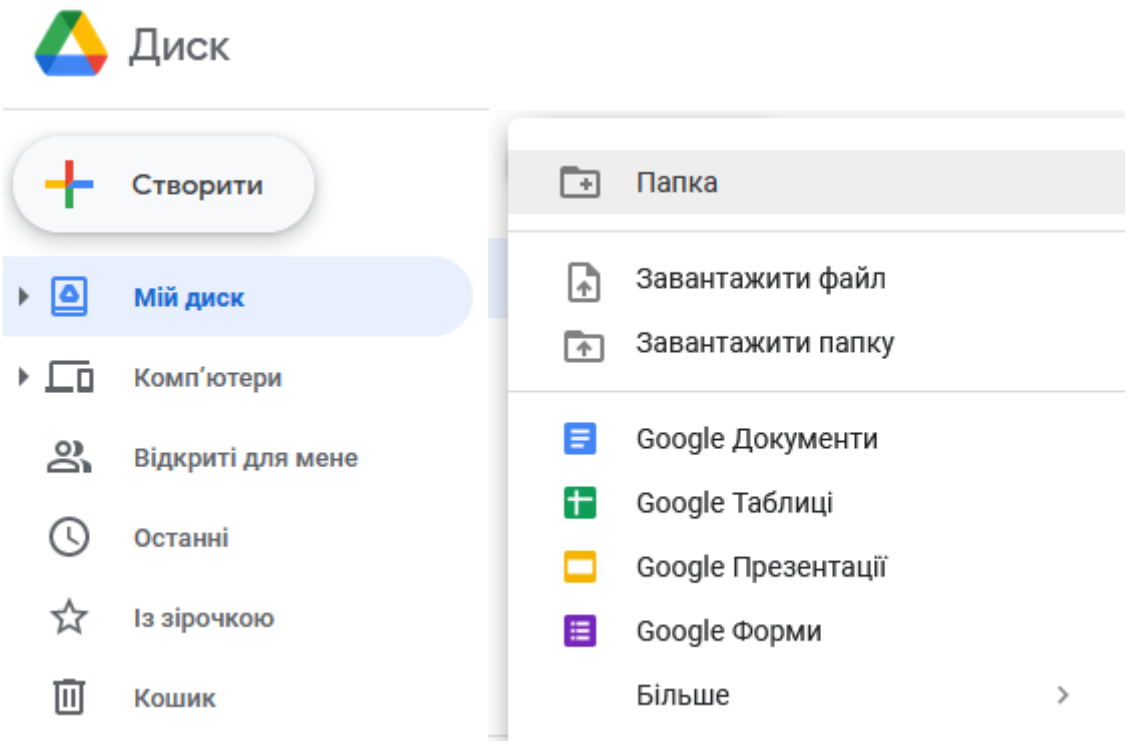

Рис. 23. **Створення нової папки**

Тепер у цій папці можна додати новий записник-файл. Це можна зробити, вибравши в такому самому меню створення нового елементу в секції "Більше" відповідний пункт Colab. Якщо цього пункту немає в меню, можна встановити додаток Google Colaboratory для Google Drive (за посиланням "Створити"/"Більше"/"Підключити інші додатки") або запустити чи копіювати на свій диск будь-який з доступних прикладів Colab-записників, що автоматично призведе до появи цього пункту меню (рис. 24).

Після вибору пункту Google Colaboratory буде створено новий файл записника. Перейменувати його можна, просто натиснувши на ім'я в лівому верхньому куті, після чого ввести нову назву.

Натиснувши "Змінити" / "Налаштування записника" або "Середовище виконання" / "Змінити тип середовища виконання", можна вибрати апаратне прискорення для записника у вигляді GPU чи TPU (рис. 25).

За замовчуванням Colab буде працювати з використанням CPU (Central Processing Unit) – звичайного центрального процесора, який виконує операції з даними.

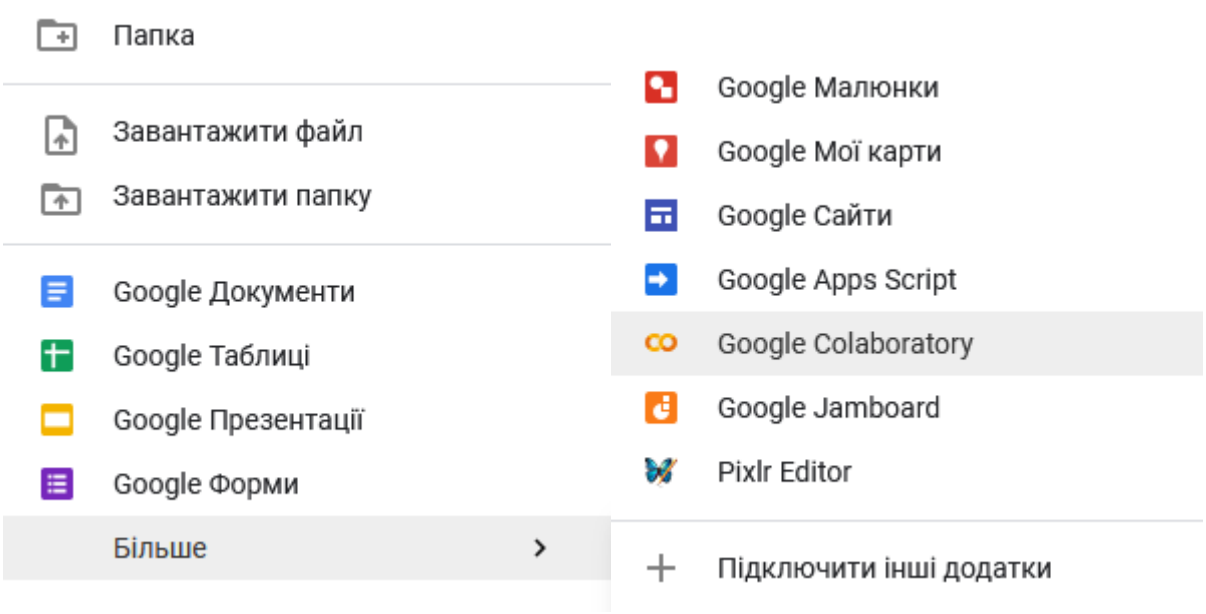

Рис. 24. **Встановлення Google Colaboratory**

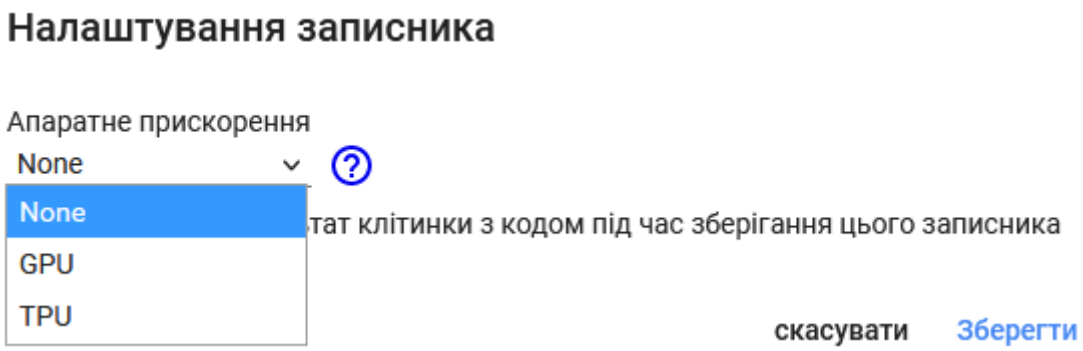

Рис. 25. **Встановлення типу середовища виконання**

GPU (Graphics Processing Unit) – графічний процесор. Обробляє дані швидше, адже обробляє дані паралельно, а не послідовно, як CPU. Його розроблено безпосередньо для роботи з графікою, але можна використовувати і для обчислень.

TPU (Tensor Processing Unit) – тензорний процесор, розроблений Google і призначений для тренування нейромереж. У цього процесора в рази вища продуктивність при великих обсягах обчислювальних завдань.

Google Colaboratory дозволяє безкоштовно і безперервно використовувати процесори протягом 12 год, після чого відбудеться автоматичне видалення всіх даних. Протягом довгого виконання коду записника на екрані може з'являтися повідомлення з проханням підтвердити, що людина все ще знаходиться перед комп'ютером. Крім того, Google

відключає блокноти після приблизно 30 хв бездіяльності, щоб не перевантажувати процесори. Система Colab так влаштована спеціально: багато факторів, у тому числі час простою, максимальна активність, загальні обмеження на обсяг пам'яті іноді динамічно змінюються. Активним учасникам ненадовго можуть обмежити доступ до GPU/TPU, щоб дати змогу використовувати процесор іншим.

Біля правого верхнього кута є налаштування підключення до віддаленого середовища з ресурсами, можливість перегляду цих ресурсів та керування сеансом (рис. 26).

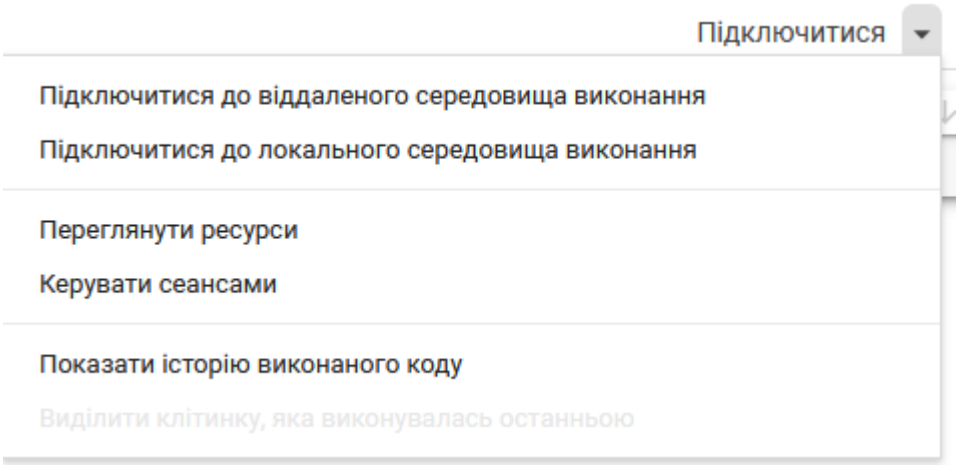

Рис. 26. **Опції підключення**

Після вдалого підключення відображається стан завантаженості та пам'яті, наведений на рис. 27.

Рис. 27. **Поточний стан використання ресурсів**

Якщо тепер ми наберемо якийсь приклад коду в python і запустимо цей фрагмент на виконання, побачимо результат обробки, наведений на рис. 28.

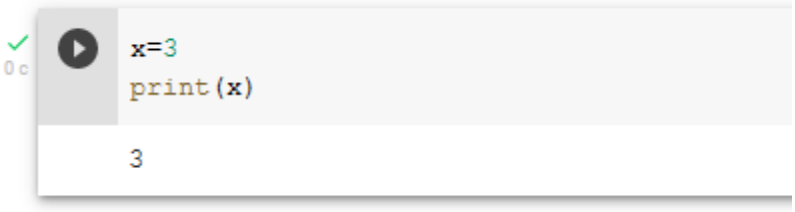

Рис. 28. **Приклад виконання коду в клітинці**

Для роботи з файлами, які зберігаються на гугл-диску, необхідно підключити його командами:

```
from google.colab import drive
drive.mount('/content/drive')
```

Після цього потрібно перейти за згенерованим посиланням нижче, авторизуватися, скопіювати код і ввести його у відповідне вікно в записнику нижче посилання (рис. 29). Потім у панелі меню ліворуч можна побачити всі файли на диску та використовувати їх у коді.

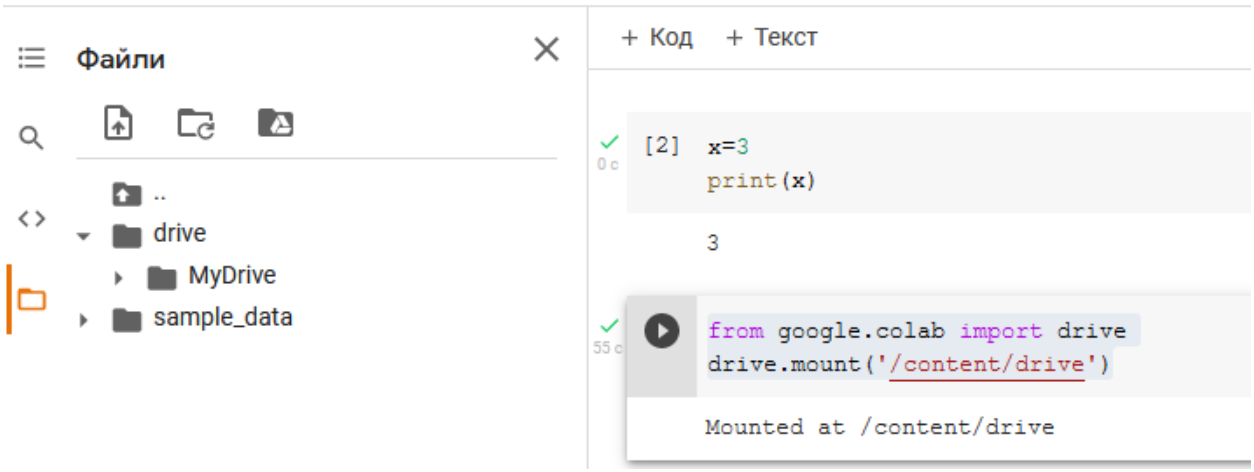

Рис. 29. **Підключення Google Drive**

Багато різних прикладів використання середовища (наприклад, клонування репозиторію Github для Colab) наведено тут: https://neptune.ai/blog/google-colab-dealing-with-files-2.

Запустити довільний ру-файл на виконання з гугл-диску можна командою (указавши середовище виконання python2 чи python3, файл для навчання та класифікації датасету MNIST для цього прикладу взято за посиланням https://gist.github.com/shravankumar147/9c9cc17e8c7ef7ef46617853377f45f5):

```
!python2 '/content/drive/MyDrive/Colab Notebooks/mnist_cnn.py'
```

Результат запуску цієї команди наведено далі (рис. 30).

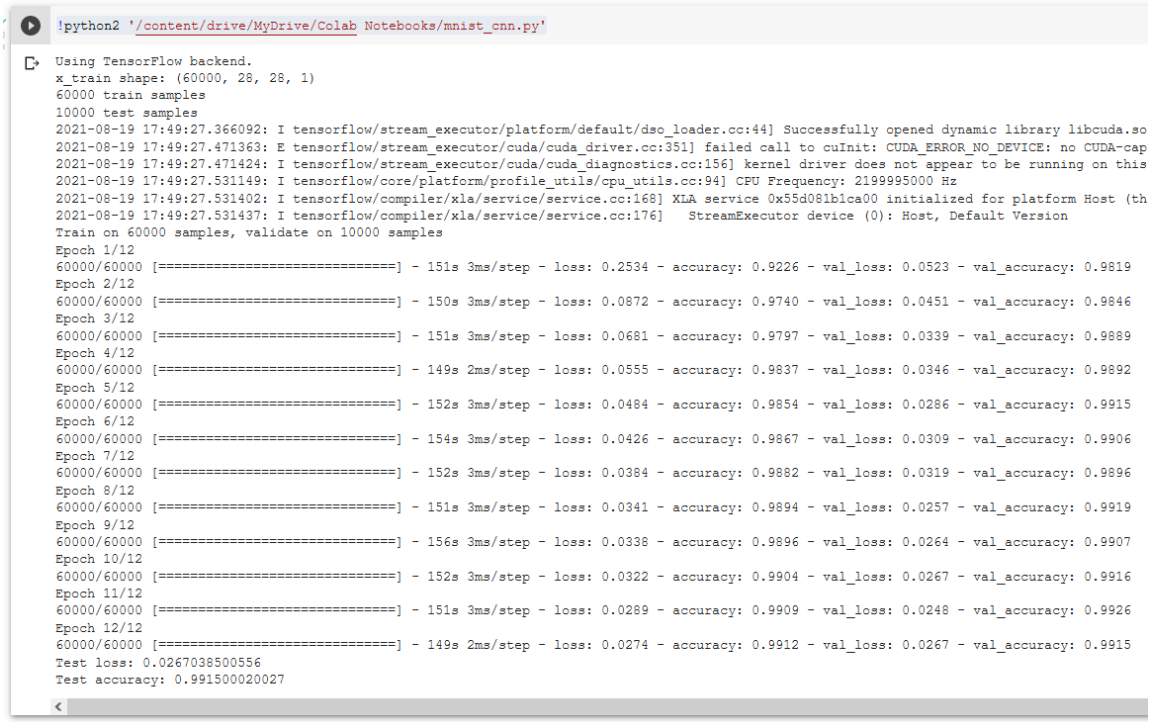

Рис. 30. **Результат запуску файлу**

Практична частина

У цьому прикладі ми розглянемо, як створити нейронну мережу та навчити її вирішувати проблему бінарної класифікації текстових даних. Будемо використовувати датасет IMDB Reviews (https://www.kaggle.com/utathya/imdb-review-dataset#imdb_master.csv), який містить тисячу відгуків на фільми. За текстом відгуку необхідно визначити, є відгук позитивним чи негативним.

Є багато способів оброблення тексту в машинному навчанні. Ми будемо використовувати вбудовування слів (word embedding).

Вбудовування слів – це метод, який використовують для подання документів у вигляді чисельного вектора. Текстові дані завжди треба конвертувати в числові, і під час оброблення тексту спосіб конвертації має вирішальне значення. Близькі в деякому контексті речення повинні відображатися в близькі чисельні вектори.

Словниковий запас (повний набір усіх слів у тренувальній вибірці) у текстових документах відображається у вектори дійсних чисел. Семантично схожі слова відображаються близько один до одного у векторному просторі. Існують готові до використання моделі вбудовування слів, такі як Word2Vec і GloVe. Однак у цьому прикладі ми використаємо Keras для навчання нашої власної моделі вбудовування слів.

Завантажимо датасет і надрукуємо декілька перших записів датасету (рис. 31).

```
import pandas as pd
import numpy as np

df = pd.read_csv('drive/MyDrive/Colab Notebooks/imdb_labelled.txt', delimiter = '\t', engine='python', quoting = 3)
df.head()
```

	Review	Status
0	A very, very, very slow-moving, aimless movie ...	0
1	Not sure who was more lost - the flat characte...	0
2	Attempting artiness with black & white and cle...	0
3	Very little music or anything to speak of.	0
4	The best scene in the movie was when Gerardo i...	1

Рис. 31. **Перегляд перших записів датасету**

Перелік необхідних модулів, які треба імпортувати, наведено на рис. 32:

- модуль array з NumPy буде використовуватися для конвертації датасету в масиви типу NumPy;
- one_hot з Keras – для кодування слів масивами цілих чисел;
- pad_sequences будуть використані для вирівнювання векторів речень до однакової довжини;
- Sequential – для побудови послідовної моделі нейронної мережі;
- Dense – для додавання повнозв'язних шарів;
- Flatten – для зміни розмірностей масиву;
- Embedding – для реалізації шару word embedding.

```
[3] from numpy import array
    from keras.preprocessing.text import one_hot
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from keras.models import Sequential
    from keras.layers import BatchNormalization
    from keras.layers import Dense
    from keras.layers import Flatten
    from keras.layers import Dropout
    from tensorflow.keras.layers import Embedding
    from tensorflow.keras.optimizers import Adam
```

Рис. 32. **Імпорт необхідних модулів**

Створимо змінні для зберігання оглядів фільмів та міток (рис. 33).

```
[3]  docs = df['Review']
     labels = array(df['Status'])
```

Рис. 33. **Змінні для зберігання фільмів та міток**

Використаємо вбудовану функцію зі sklearn для розбиття датасету на навчальний (80 %) і тестовий (20 %) набори. Надрукуємо для прикладу перший відгук навчального набору (рис. 34).

```
[4]  from sklearn.model_selection import train_test_split
     X_train, X_test , y_train, y_test = train_test_split(docs, labels , test_size = 0.20)

     print(X_train[1])

     Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.
```

Рис. 34. **Розбиття набору даних**

На наступному етапі ми конвертуємо кожне речення в набір чисел із використанням функції one_hot (рис. 35). Вона має такі аргументи:

- текст для конвертації;
- розмір словника;
- фільтр, який відкидає з результатів знаки пунктуації;
- булеве значення, яке вказує, чи потрібно трансформувати всі великі літери в малі;
- split, який указує розділовий знак між словами.

```
[22] vocab_size = 500

     X_train = [one_hot(d, vocab_size,filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~',lower=True, split=' ') for d in X_train]
     X_test = [one_hot(d, vocab_size,filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~',lower=True, split=' ') for d in X_test]
```

Рис. 35. **Код для конвертації тексту в числові значення**

Подивимося на результат (рис. 36). Функція one_hot не гарантує унікальності, якщо розмір словника великий (>1 000), тобто різні слова можуть мати однакові індекси.

```
[86] print(X_train[1])

     [343, 196, 438, 90, 427, 71, 118, 388, 374]
```

Рис. 36. **Речення, представлене вектором**

Якщо подивитися на інше речення в числовому вигляді, то воно, найпевніше, буде мати іншу довжину. Традиційно нейронні мережі будують таким чином, що всі вхідні вектори (так само, як і вектори, які потім будуть використовувати для класифікації) мають однакову довжину. Тож функція pad_sequences дозволяє вирівняти всі чисельні вектори речень у тренувальному наборі до однакової довжини. Ця функція як параметри отримує масив значень (у нашому випадку – масив векторів чисел, кожне з яких відповідає слову), максимально дозволену його довжину та спосіб додавання нулів – на початку чи в кінці (рис. 37).

```
max_length = 25
X_train = pad_sequences(X_train, maxlen=max_length, padding='pre')
X_test = pad_sequences(X_test, maxlen=max_length, padding='pre')
```

```
print(X_train[1])

[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 343 196
 438  90 427  71 118 388 374]
```

Рис. 37. **Вирівнювання довжин векторів**

Тепер можна будувати модель нейронної мережі. Це буде мережа прямого поширення, тож використаємо модель Sequential. Наступний шар називають Embedding, він отримує розмір словника, розмір вектора на виході з цього шару та розмір вхідного вектора. Цей шар будує вектор embedding, який визначає унікальність кожного текстового речення та їхню близькість. Ці вектори будуть змінюватися під час навчання.

Вихід із шару Embeddings "сплющується" в одновимірний масив за допомогою Flatten і передається на два послідовні повнозв'язні шари Dense із відповідними функціями активації relu та sigmoid. Між ними розташовано шар відкидання певної частини нейронів (Dropout) для запобігання перенавчанню.

Після створення моделі вона компілюється (compile) із відповідним методом оптимізації (було вибрано Adam), функцією втрат

(binary_crossentropy) та метрикою оцінювання якості (metrics). У наступному рядку вказано метод summary, який друкує структуру всієї моделі.

Тепер можна запускати навчання (fit). Під час навчання вказують навчальну вибірку з двох частин X_train, y_train, кількість ітерацій, коефіцієнт розбиття між навчальною та валідаційною вибіркою (рис. 38). Додатково параметром verbose можна вказати рівень деталізації друку результатів.

```
model = Sequential()
model.add(Embedding(vocab_size, 32, input_length=max_length))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

print(model.summary())

history = model.fit(X_train, y_train, epochs=50, verbose=1, validation_split=0.4)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 25, 32)            16000
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 32)                25632
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 41,665
Trainable params: 41,665
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
15/15 [==============================] - 1s 17ms/step - loss: 0.6912 - acc: 0.5063 - val_loss: 0.7013 - val_acc: 0.4500
```

Рис. 38. **Створення мережі, її компіляція та запуск навчання**

Після завершення процедури навчання за допомогою evaluate можна оцінити якість моделі на навчальній і тестовій вибірках (рис. 39).

```
[90] loss, accuracy = model.evaluate(X_train, y_train, verbose=1)
     print('Training Accuracy is {}'.format(accuracy*100))

     25/25 [==============================] - 0s 1ms/step - loss: 0.5838 - acc: 0.8438
     Training Accuracy is 84.375

[91] loss, accuracy = model.evaluate(X_test,y_test)
     print('Testing Accuracy is {} '.format(accuracy*100))

     7/7 [==============================] - 0s 2ms/step - loss: 1.1817 - acc: 0.6500
     Testing Accuracy is 64.99999761581421
```

Рис. 39. **Оцінювання точності моделі**

Наступний фрагмент коду можна використати для побудови графіків навчання (рис. 40).

```
import matplotlib.pyplot as plt

def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])

plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plot_graphs(history, 'acc')
plt.ylim(None, 1)
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
plt.ylim(0, None)
```

```
(0.0, 1.5313767280313186)
```

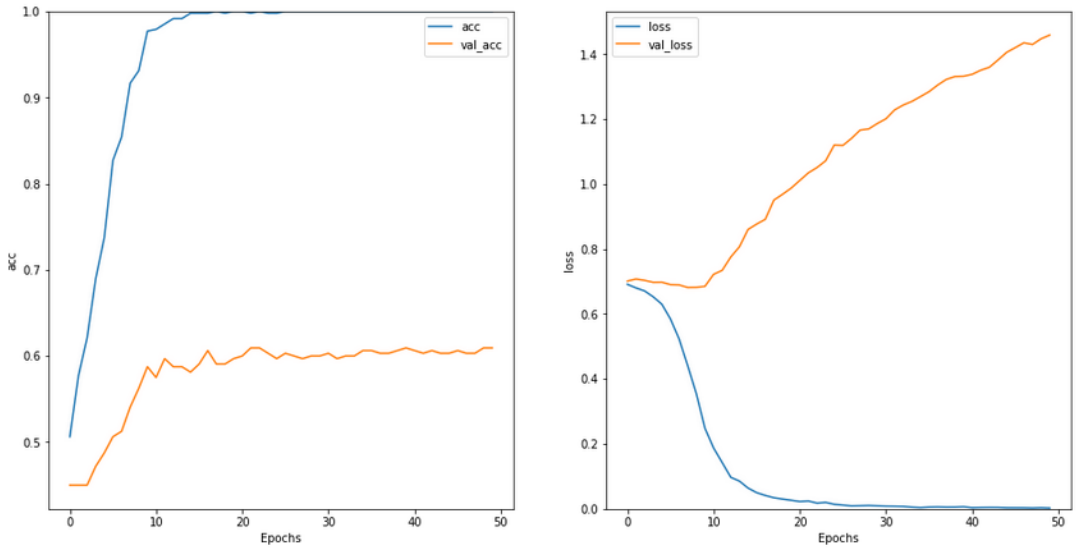

Рис. 40. **Графіки процесу навчання**

На графіку ліворуч відображаються значення сумарного відхилення та точність навчання для окремих ітерацій навчальної вибірки, праворуч – валідаційної вибірки.

Зверніть увагу, що відхилення тренувального набору зменшується з кожною ітерацією (епохою), а точність навчання збільшується з кожною епохою. Це очікувано за умови використання оптимізації градієнтного спуску, яка повинна мінімізувати значення помилки на кожній ітерації. Але це не стосується похибки і точності для валідаційної вибірки. Це є прикладом перенавчання (overfitting): модель працює набагато краще з навчальними даними, ніж з даними, яких вона ніколи не бачила. У процесі такого навчання з певного моменту (зростання сумарної помилки val_loss) модель починає надмірно оптимізувати навчальний набір, але не узагальнюється для довільних векторів. У цьому конкретному випадку

можна було б запобігти такому перенавчанню, просто зупинивши навчання вчасно. Проблема перенавчання часто виникає через надмірно складну модель нейронної мережі.

Для прикладу можна також оцінити класифікацію довільних речень (рис. 41). Перший відгук є позитивним, результат його класифікації дорівнює 0.867, що ближче до 1 – класу позитивних відгуків, ніж до 0 – класу негативних відгуків. Другий відгук є негативним, відповідно результат його класифікації дорівнює 0.018.

Рис. 41. **Класифікація речень**

Індивідуальні завдання

1. Повторіть наведений приклад. Першоджерело прикладу з описом можна знайти тут: https://heartbeat.fritz.ai/using-a-keras-embedding-layer-to-handle-text-data-2c88dc019600.

2. Оскільки нейронні мережі мають стохастичну природу, у результаті кожного нового запуску записника ми будемо отримувати нову модель нейронної мережі, яка відрізняється від попередніх, навіть якщо код змінено не було. Переконайтеся в цьому, порівнявши точність роботи мережі для кількох перезапусків.

Подібна особливість нейронних мереж ускладнює роботу з поліпшення моделей, адже стає незрозумілим, що саме впливає на результат роботи – випадкова природа мережі чи якісь зміни коду, моделі, процесу навчання тощо.

Для того, щоб результати експерименту можна було стабільно відтворити, треба прибрати технічну причину випадковості в коді. Вона забезпечується випадковим параметром seed, який кожного разу ініціюється наново у використовуваних бібліотеках.

Наприклад, випадкову поведінку в бібліотеці numpy може бути зафіксовано так:

```
from numpy.random import seed
seed(42)
```

для загального коду:

```
import random as rn
rn.seed(42)
```

для бібліотеки TensorFlow:

```
import tensorflow as tf
tf.compat.v1.set_random_seed(42)
```

Вибране значення зерна seed установлюють будь-яким (може бути різним для різних бібліотек), але фіксованим. У цьому прикладі встановлено значення 42.

Описаний приклад застосовує numpy й tensorflow. Зафіксуйте для них випадкову ініціалізацію та переконайтесь у стабільності отриманого результату.

3. Приклад використовує фіксовану довжину словника в 500 слів. Невідомо, чи є ця довжина достатньою. Спробуйте встановити її відповідно до загальної кількості унікальних слів у навчальний вибірці. Знайти цю кількість можна таким кодом:

```
all_words = []
for sent in X_train:
  tokenize_word = sent.lower().split(' ')
  for word in tokenize_word:
    all_words.append(word)

unique_words = set(all_words)
print(len(unique_words))
```

Чи поліпшила ця зміна якість класифікації?

4. Спробуйте змінити інші параметри, щоб отримати кращий результат. Можна розглянути зміну структури мережі, метод навчання

(замість Adam), максимальну довжину вектора, функції активації, розмір вибірок, кількість елементів під час навчання (batch_size для функції fit), розмір словника, кількість ітерацій тощо.

5. Застосуйте код до обробки іншого датасету IMBD Movies Review dataset: https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews/version/1# (адаптований файл "movie_data.csv" можна знайти на сторінці курсу). Які параметри довелося змінити для адаптації коду під цей датасет?

Для зчитування цього набору даних можна використати такий рядок:

df = pd.read_csv(movie_data.csv', engine='python', quoting = 1, sep=',')

6. Порівняйте швидкість навчання на CPU та GPU.

Підготуйте звіт за результатами виконаної роботи. Звіт має містити титульний аркуш; опис виконаних завдань із коментарями, кодом, скрин-шотами; короткі висновки з кожного завдання та з лабораторної роботи загалом. Висновки не повинні дублювати мету.

Підготуйтеся до захисту роботи за контрольними запитаннями. Запитання, позначені символом *, передбачають вивчення додаткової літератури.

Контрольні запитання до лабораторної роботи 3

1. Що таке Google Colaboratory? Для чого її використовують?
2. Що таке GPU та TPU?
3. Для чого використовують шар Embedding?
4. Для чого використовують шари Dense і Flatten?
5. Чому нейронні мережі навчаються кожного разу по-новому навіть без зміни коду?
6. Навіщо штучно прибирати випадковість процесу навчання нейронних мереж ?

7*. Що таке перенавчання?

8*. Яку роль відіграє шар Dropout?

9*. Для чого використовують навчальний, тестовий і валідаційний набори?

10*. Для чого використовують one_hot?

Лабораторна робота 4
Класифікація зображень із використанням конволюційних ШНМ

Мета роботи: ознайомитися з методом побудови конволюційних нейронних мереж із використанням Google Colab і Keras для класифікації зображень.

Теоретична частина

Розглянемо приклад класифікації зображень одягу на прикладі датасету Fashion MNIST. Він містить 10 класів одягу: T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle Boot. Цей датасет містить 60 000 навчальних напівтонових зображень розміром 28×28 пікселів і 10 000 тестових зображень такого самого розміру.

Отже, перший фрагмент коду (рис. 42) імпортує необхідні бібліотеки tensorflow, numpy, matplotlib, завантажує навчальну та тестову вибірки датасету Fashion MNIST і друкує їхній розмір.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.fashion_mnist.load_data()
print("x_train shape:", x_train.shape, "y_train shape:", y_train.shape)
```

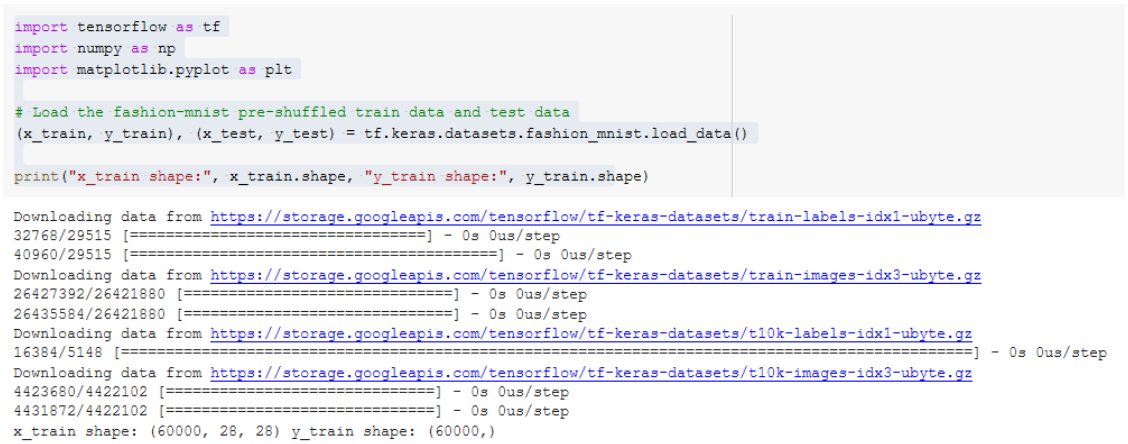

Рис. 42. **Імпорт бібліотек та завантаження даних**

Наступний фрагмент заповнює масив міток класів, який буде використано для візуалізації:

```
fashion_mnist_labels = ["T-shirt/top",  # index 0
                        "Trouser",      # index 1
                        "Pullover",     # index 2
                        "Dress",        # index 3
                        "Coat",         # index 4
                        "Sandal",       # index 5
                        "Shirt",        # index 6
                        "Sneaker",      # index 7
                        "Bag",          # index 8
                        "Ankle boot"]   # index 9
```

Далі слід виконати нормалізацію завантажених зображень (можна просто розділити всі значення на максимум 255, отримавши таким чином усі дані в діапазоні від 0 до 1 включно).

```
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

Тепер необхідно трохи підготувати навчальну та тренувальну вибірку, створити валідаційну (візьмемо для неї перші 5 000 елементів тренувального набору). Одне значення номера класу Y для всіх трьох вибірок потрібно трансформувати в категоріальний вектор за допомогою функції to_categorical.

```
(x_train, x_valid) = x_train[5000:], x_train[:5000]
(y_train, y_valid) = y_train[5000:], y_train[:5000]

w, h = 28, 28
x_train = x_train.reshape(x_train.shape[0], w, h, 1)
x_valid = x_valid.reshape(x_valid.shape[0], w, h, 1)
x_test = x_test.reshape(x_test.shape[0], w, h, 1)

y_train = tf.keras.utils.to_categorical(y_train, 10)
y_valid = tf.keras.utils.to_categorical(y_valid, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Створюємо модель нейронної мережі. Обробка починається із застосування конволюції з 64 фільтрів розміру 2×2, на наступному шарі виконується об'єднання (max-pooling) також розміром 2×2. padding='same' вказує, що конволюція буде зберігати розмір початкового зображення за допомогою заповнення нулями значень на межах зображення.

Після конволюції та шару об'єднання використано регуляризаційний шар Dropout, який деактивує 30 % випадкових нейронів для запобігання перенавчанню.

Після цих трьох початкових шарів застосовано таку саму комбінацію, але замість 64 фільтрів у шарі згортки тепер використано 32.

Результат пошуку ознак за допомогою конволюцій "сплющується" шаром Flatten, після якого використано два повнозв'язні шари Dense із шаром Dropout між ними. Останній шар містить 10 нейронів, що відповідає 10 класам у датасеті, із функцією активації softmax, оскільки класи є взаємовиключні.

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=2, padding='same', activation='relu', input_shape=(28,28,1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Dropout(0.3))

model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Dropout(0.3))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.summary()
```

Роздруківка структури моделі (summary, рис. 43) показує, скільки параметрів навчаються на кожному шарі. Наприклад, на першому конволюційному шарі навчаються 320 параметрів: 64 фільтри розміром 2×2,

а також 64 зміщення. На другому конволюційному шарі маємо 8 224 параметри: 32 фільтри розміром 2×2 для кожного з 64 результатів фільтрації, що прийшли з попереднього шару, а також 32 зміщення для нового конволюційного шару: $32 \times 2 \times 2 \times 64 + 32 = 8\,224$.

Після обробки шаром Dense у результаті отримуємо вектор із 1 568 елементів, який подається на вхід повнозв'язного шару Dense із 256 нейронами, кожен із яких також має зміщення. Отримуємо $1\,568 \times 256 + 256 = 401\,664$ коефіцієнти, які треба знайти під час навчання.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 64)        320

max_pooling2d (MaxPooling2D  (None, 14, 14, 64)        0
)

dropout (Dropout)            (None, 14, 14, 64)        0

conv2d_1 (Conv2D)            (None, 14, 14, 32)        8224

max_pooling2d_1 (MaxPooling  (None, 7, 7, 32)          0
2D)

dropout_1 (Dropout)          (None, 7, 7, 32)          0

flatten (Flatten)            (None, 1568)              0

dense (Dense)                (None, 256)               401664

dropout_2 (Dropout)          (None, 256)               0

dense_1 (Dense)              (None, 10)                2570

=================================================================
Total params: 412,778
Trainable params: 412,778
Non-trainable params: 0
_________________________________________________________________
```

Рис. 43. **Роздрукована структура моделі**

Можна помітити типову особливість конволюційних нейронних мереж: початкові шари мають невелику кількість параметрів для навчання, але виконують функцію пошуку характерних ознак навчальних даних і характеризуються великими вимогами до пам'яті (наприклад, якщо б ми мали кольорові зображення, а не напівтонові, то все це потребувало б збереження втричі більшої кількості даних – для кожної з матриць R, G, B окремо). Останні шари мають велику кількість параметрів, які треба навчити, і це відбувається повільніше, порівняно з шарами на початку, але тут уже не треба великих обсягів пам'яті для збереження та обробки зображень.

Компілюємо модель:

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

та навчаємо її (зі збереженням коефіцієнтів найкращої моделі у файл):

```
from keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint(filepath='model.weights.best.hdf5',
verbose = 1, save_best_only=True)

model.fit(x_train, y_train, batch_size=64, epochs=10,
validation_data=(x_valid, y_valid), callbacks=[checkpointer])
```

Процес навчання наведено на рис. 44.

```
Epoch 1/10
855/860 [============================>.] - ETA: 0s - loss: 0.6061 - accuracy: 0.7757
Epoch 00001: val_loss improved from inf to 0.37878, saving model to model.weights.best.hdf5
860/860 [==============================] - 10s 11ms/step - loss: 0.6053 - accuracy: 0.7759 - val_loss: 0.3788 - val_accuracy: 0.8666
Epoch 2/10
859/860 [============================>.] - ETA: 0s - loss: 0.4188 - accuracy: 0.8468
Epoch 00002: val_loss improved from 0.37878 to 0.32972, saving model to model.weights.best.hdf5
860/860 [==============================] - 9s 10ms/step - loss: 0.4187 - accuracy: 0.8469 - val_loss: 0.3297 - val_accuracy: 0.8810
Epoch 3/10
859/860 [============================>.] - ETA: 0s - loss: 0.3748 - accuracy: 0.8624
Epoch 00003: val_loss improved from 0.32972 to 0.29974, saving model to model.weights.best.hdf5
860/860 [==============================] - 9s 10ms/step - loss: 0.3747 - accuracy: 0.8625 - val_loss: 0.2997 - val_accuracy: 0.8914
Epoch 4/10
858/860 [============================>.] - ETA: 0s - loss: 0.3461 - accuracy: 0.8730
Epoch 00004: val_loss improved from 0.29974 to 0.28943, saving model to model.weights.best.hdf5
860/860 [==============================] - 9s 10ms/step - loss: 0.3463 - accuracy: 0.8730 - val_loss: 0.2894 - val_accuracy: 0.8956
Epoch 5/10
857/860 [============================>.] - ETA: 0s - loss: 0.3285 - accuracy: 0.8795
Epoch 00005: val_loss improved from 0.28943 to 0.26982, saving model to model.weights.best.hdf5
860/860 [==============================] - 9s 10ms/step - loss: 0.3285 - accuracy: 0.8795 - val_loss: 0.2698 - val_accuracy: 0.9016
Epoch 6/10
859/860 [============================>.] - ETA: 0s - loss: 0.3148 - accuracy: 0.8827
Epoch 00006: val_loss improved from 0.26982 to 0.26265, saving model to model.weights.best.hdf5
860/860 [==============================] - 9s 10ms/step - loss: 0.3148 - accuracy: 0.8827 - val_loss: 0.2626 - val_accuracy: 0.9034
Epoch 7/10
859/860 [============================>.] - ETA: 0s - loss: 0.2997 - accuracy: 0.8887
Epoch 00007: val_loss improved from 0.26265 to 0.24419, saving model to model.weights.best.hdf5
860/860 [==============================] - 9s 10ms/step - loss: 0.2996 - accuracy: 0.8887 - val_loss: 0.2442 - val_accuracy: 0.9120
Epoch 8/10
857/860 [============================>.] - ETA: 0s - loss: 0.2919 - accuracy: 0.8924
Epoch 00008: val_loss did not improve from 0.24419
860/860 [==============================] - 9s 10ms/step - loss: 0.2918 - accuracy: 0.8925 - val_loss: 0.2485 - val_accuracy: 0.9056
Epoch 9/10
856/860 [============================>.] - ETA: 0s - loss: 0.2846 - accuracy: 0.8929
Epoch 00009: val_loss improved from 0.24419 to 0.23553, saving model to model.weights.best.hdf5
860/860 [==============================] - 9s 10ms/step - loss: 0.2845 - accuracy: 0.8928 - val_loss: 0.2355 - val_accuracy: 0.9134
Epoch 10/10
856/860 [============================>.] - ETA: 0s - loss: 0.2707 - accuracy: 0.8988
Epoch 00010: val_loss improved from 0.23553 to 0.23051, saving model to model.weights.best.hdf5
860/860 [==============================] - 9s 10ms/step - loss: 0.2706 - accuracy: 0.8987 - val_loss: 0.2305 - val_accuracy: 0.9148
<keras.callbacks.History at 0x7fcdd1eb5650>
```

Рис. 44. **Процес навчання мережі**

Для перевірки роботи завантажуємо найкращу збережену модель (якщо декілька ітерацій не привели до поліпшення моделі, вагові коефіці-

єнти все одно змінюються і модель після останньої ітерації може не бути найкращою):

```
model.load_weights('model.weights.best.hdf5')
```

Перевіряємо точність на тестовому наборі:

```
score = model.evaluate(x_test, y_test, verbose=0)
print('\n', 'Test accuracy:', score[1])
```

Практична частина
Індивідуальні завдання

1. Повторіть наведений приклад. Першоджерело прикладу з описом можна знайти тут: https://github.com/margaretmz/deep-learning/blob/master/fashion_mnist_keras.ipynb.

2. Дослідіть, чи впливає нормалізація зображень на результати навчання та класифікації.

3. Додайте ще один конволюційний шар у модель мережі. Перевірте, чи це поліпшує її якість.

4. Замість додавання ще одного конволюційного шару спробуйте підвищити потужність одного з двох шарів, що існують. Перевірте, чи поліпшує це якість моделі.

5. Спробуйте змінити інші параметри, щоб отримати кращий результат.

Підготуйте звіт за результатами виконаної роботи. Звіт має містити титульний аркуш; опис виконаних завдань із коментарями, кодом, скриншотами; висновки з кожного завдання та з лабораторної роботи загалом. Висновки не повинні дублювати мету.

Підготуйтеся до захисту роботи за контрольними запитаннями. Запитання, позначені символом *, передбачають вивчення додаткової літератури.

Контрольні запитання до лабораторної роботи 4

1*. Що таке конволюція? У чому полягає ця операція?

2. Яку функцію виконують конволюційні шари в нейронній мережі?

3. Навіщо використовувати шари об'єднання pooling?

4*. Що таке регуляризація?

5. Для вирішення яких типових завдань використовують конволюційні нейронні мережі?

6. Чим принципово відрізняються перші та останні шари конволюційної мережі?

7*. Для чого використовують зворотні виклики callbacks у keras?

8. Які параметри навчаються в конволюційному шарі?

Лабораторна робота 5
Класифікація зображень із використанням передавального навчання

Мета роботи: ознайомитися з технологією трансферного (передавального) навчання із застосуванням попередньо навчених моделей у середовищі Google Colab.

Теоретична частина

Розглянемо класифікацію зображень порід собак і котів на прикладі датасету The Oxford-IIIT Pet Dataset (https://www.robots.ox.ac.uk/~vgg/data/pets/). Він містить 37 класів порід: 25 класів собак і 12 класів котів, приблизно по 200 зображень різного розміру для кожної породи

Цей датасет поширюється з додатковими файлами: test.txt, який містить 3 669 імен тестових зображень, і trainval.txt, який містить 3 680 файлів для навчання та валідації.

Розв'язання цієї задачі є достатньо складним, а кількість зображень не є великою, тож ми розіб'ємо 3 680 файлів на 3 310 тренувальних і 370 валідаційних (по 10 для кожного з класів). Також додатково зробимо для зручності файл із усіма назвами класів.

Таким чином, у нас для роботи є такі файли: images.zip – архів із усіма зображеннями (або архів images.tar.gz, завантажений за посиланням вище), train2.txt – список тренувальних файлів, val10.txt – список валідаційних файлів, text.txt – список тестових файлів, labels_catsdogs.txt – список усіх 37 класів.

Після того, як усі файли розташовано у відповідному каталозі Google Drive, можна зробити записник із першою командою, яка буде відкривати архів із усіма зображеннями:

```
!unzip "drive/MyDrive/Colab Notebooks/data/images.zip"
```

Результат розархівування:

```
inflating: images/Egyptian_Mau_90.jpg
inflating: images/Egyptian_Mau_91.jpg
inflating: images/Egyptian_Mau_92.jpg
inflating: images/Egyptian_Mau_93.jpg
inflating: images/Egyptian_Mau_94.jpg
inflating: images/Egyptian_Mau_95.jpg
inflating: images/Egyptian_Mau_96.jpg
```

Для відкриття архіву tar.gz можна використати такий рядок:

```
!tar -xzvf "drive/MyDrive/Colab Notebooks/data/images.tar.gz"
```

У роботі будемо застосовувати дві нескладні функції, які дозволять знайти ім'я файлу та номер класу з рядка в тому форматі, у якому зберігаються значення у файлах датасету.

```
def class_name(path):
    pos = path.rfind('_')
    clearClass = path[0: pos]
    return clearClass

def image_name(path):
    pos = path.find(' ')
    imagename = path[0: pos]
    return imagename
```

Наступна клітинка встановлює основні параметри: усі зображення будуть перед використанням масштабовані до розміру 250×250 пікселів, кількість ітерацій навчання (50), допоміжні змінні, завантаження всіх класів, а також тренувального та валідаційного наборів у вигляді імен файлів.

```
IMAGE_SIZE = 250
EPOCHS = 50
path = 'images/'
labelRows=[]
values=[]
x_testnames=[]
```

```
labels=[]
with open("drive/MyDrive/Colab Notebooks/data/labels_catsdogs.txt") a
s f:
    labels = f.read().splitlines()
print(labels)

label_map = {l: i for i, l in enumerate(labels)}

trainfold =[]
testfold = []
valfold = []
rootdir = path

with open("drive/MyDrive/Colab Notebooks/data/train2.txt", 'r') as f:
    trainfold = [line.rstrip('\n') for line in f]
with open("drive/MyDrive/Colab Notebooks/data/val10.txt", 'r') as f:
    valfold = [line.rstrip('\n') for line in f]
```

Імпортуємо всі необхідні бібліотеки та класи:

```
import numpy as np
np.random.seed(42)
import pandas as pd
import os
import gc
import re
import keras
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, BatchNormalization,
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import Adam
import cv2
import string
import sys
from random import shuffle
import random

import tensorflow as tf

from keras.applications.densenet import DenseNet201
from keras.preprocessing import image
from keras.applications.densenet import preprocess_input
```

Створюємо першу модель на основі DenseNet201 – pre-trained моделі, з якої візьмемо конволюційні шари для визначення ознак, а останні шари відкинемо. Завантажимо до неї вагові коефіцієнти після навчання на ImageNet.

model = DenseNet201(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3), weights='imagenet', include_top=False)

"Заморожуємо" вже навчені вагові коефіцієнти:

model.trainable = False

Після виконання цих клітинок Colab автоматично завантажить указану модель DenseNet201 із відповідними ваговими коефіцієнтами:

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet201_weights_tf_dim_ordering_tf_kernels_notop.h5
74842112/74836368 [==============================] - 1s 0us/step
74850304/74836368 [==============================] - 1s 0us/step

Додамо два зворотних виклики, щоб зупиняти навчання моделі, якщо вона не показує прогресу, а також щоб зберегти модель із найкращою точністю.

check_point=ModelCheckpoint(filepath="tempWeights2_checkpoint.h5", monitor="val_sparse_categorical_accuracy", mode="max", save_best_only=True, save_weights_only = True, verbose = 1)

early_stop=EarlyStopping(monitor="val_sparse_categorical_accuracy", patience = 3)

Додаємо останні шари до моделі. Шар Flatten буде "сплющувати" результат із виходу попередніх конволюційних шарів із моделі DenseNet201, після чого додано ще два повнозв'язних шари з 1 024 і 37 нейронів відповідно.

```
model2 = Sequential()
model2.add(Flatten(input_shape = (7, 7, 1920)))
model2.add(BatchNormalization())
model2.add(Dense(1024, activation='relu'))
model2.add(Dropout(0.5))
model2.add(BatchNormalization())
model2.add(Dense(37, activation='softmax'))

if os.path.isfile('tempWeights2_checkpoint.h5'):
  os.remove('tempWeights2_checkpoint.h5')

if os.path.isfile('tempWeights2.h5'):
  os.remove('tempWeights2.h5')
```

Компілюємо модель:

```
model2.compile(loss='sparse_categorical_crossentropy', optimizer='ada
m', metrics=['sparse_categorical_accuracy'])
```

Заповнюємо набір валідаційних зображень. Функція preprocess_output реалізує звичну для DenseNet201 попередню обробку кожного зображення. Також можна помітити, що валідаційні ознаки для навчання (так само, як тестові та навчальні потім) обчислюються як результат обробки методом predict зображення попередньою моделлю DenseNet201. Саме ці ознаки і будуть передаватися на другу частину мережі далі.

```
x_val = []
y_val = []
val_features = []

   for filename in valfold:
      clearClass = class_name(filename)
      img = cv2.imread(path + image_name(filename) + ".jpg")
      if img is not None:

         targets=label_map[clearClass]

         proc_image = cv2.resize(img, (IMAGE_SIZE, IMAGE_SIZE))
         x_val.append(preprocess_input(proc_image))
         y_val.append(targets)
```

```
y_val = np.array(y_val, np.uint8)
x_val = np.array(x_val, np.float16)
val_features = model.predict(x_val, verbose=2)
```

Навчаємо модель, завантажуючи по 4 000 зображень одночасно (у нас їх усього 3 310, і вони завантажуються та обробляються за один раз; таким прийомом ми хочемо показати, що навчальну множину даних також можна подавати й обробляти частинами).

```
STEP = 4000

    for i in range(0, len(trainfold), STEP):
        x_train = []
        y_train = []

        for filename in trainfold[i:i+STEP]:
            clearClass = class_name(filename)
            img = cv2.imread(path + image_name(filename) + ".jpg")
            if img is not None:

                targets=label_map[clearClass]

                proc_image = cv2.resize(img, (IMAGE_SIZE, IMAGE_SIZE))
                x_train.append(preprocess_input(proc_image))
                y_train.append(targets)

        y_train = np.array(y_train, np.uint8)
        x_train = np.array(x_train, np.float16)

        train_features = model.predict(x_train, verbose=2)

        model2.fit(train_features, y_train,
                   batch_size = 32,
                   epochs = EPOCHS,
                   verbose = 2,
                   validation_data=(val_features, y_val),
                   callbacks=[check_point, early_stop]
                  )

        model2.save_weights('tempWeights2.h5')

        if os.path.isfile('tempWeights2_checkpoint.h5'):
            model2.load_weights('tempWeights2_checkpoint.h5')
```

Результат навчання:

104/104 - 690s - 690s/epoch - 7s/step
Epoch 1/50

Epoch 00001: val_sparse_categorical_accuracy improved from -inf to 0.88378, saving model to tempWeights2_checkpoint.h5
104/104 - 85s - loss: 1.0475 - sparse_categorical_accuracy: 0.7015 - val_loss: 0.4049 - val_sparse_categorical_accuracy: 0.8838 - 85s/epoch - 820ms/step
Epoch 2/50

Epoch 00002: val_sparse_categorical_accuracy improved from 0.88378 to 0.89730, saving model to tempWeights2_checkpoint.h5
104/104 - 88s - loss: 0.1206 - sparse_categorical_accuracy: 0.9692 - val_loss: 0.3206 - val_sparse_categorical_accuracy: 0.8973 - 88s/epoch - 849ms/step
Epoch 3/50

Epoch 00003: val_sparse_categorical_accuracy improved from 0.89730 to 0.91622, saving model to tempWeights2_checkpoint.h5
104/104 - 85s - loss: 0.0420 - sparse_categorical_accuracy: 0.9934 - val_loss: 0.3071 - val_sparse_categorical_accuracy: 0.9162 - 85s/epoch - 818ms/step
Epoch 4/50

Epoch 00004: val_sparse_categorical_accuracy improved from 0.91622 to 0.91892, saving model to tempWeights2_checkpoint.h5
104/104 - 85s - loss: 0.0240 - sparse_categorical_accuracy: 0.9961 - val_loss: 0.2842 - val_sparse_categorical_accuracy: 0.9189 - 85s/epoch - 821ms/step
Epoch 5/50

Epoch 00005: val_sparse_categorical_accuracy did not improve from 0.91892
104/104 - 84s - loss: 0.0142 - sparse_categorical_accuracy: 0.9994 - val_loss: 0.2779 - val_sparse_categorical_accuracy: 0.9108 - 84s/epoch - 806ms/step
Epoch 6/50

Epoch 00006: val_sparse_categorical_accuracy did not improve from 0.91892

```
104/104 - 84s - loss: 0.0116 - sparse_categorical_accuracy: 0.9985 -
val_loss: 0.2911 - val_sparse_categorical_accuracy: 0.9162 - 84s/epoch -
807ms/step
Epoch 7/50

Epoch 00007: val_sparse_categorical_accuracy improved from 0.91892 to
0.92703, saving model to tempWeights2_checkpoint.h5
104/104 - 87s - loss: 0.0065 - sparse_categorical_accuracy: 1.0000 -
val_loss: 0.2829 - val_sparse_categorical_accuracy: 0.9270 - 87s/epoch -
836ms/step
Epoch 8/50

Epoch 00008: val_sparse_categorical_accuracy did not improve from
0.92703
104/104 - 85s - loss: 0.0055 - sparse_categorical_accuracy: 1.0000 -
val_loss: 0.3006 - val_sparse_categorical_accuracy: 0.9189 - 85s/epoch -
816ms/step
Epoch 9/50

Epoch 00009: val_sparse_categorical_accuracy did not improve from
0.92703
104/104 - 86s - loss: 0.0038 - sparse_categorical_accuracy: 0.9997 -
val_loss: 0.2877 - val_sparse_categorical_accuracy: 0.9162 - 86s/epoch -
827ms/step
Epoch 10/50

Epoch 00010: val_sparse_categorical_accuracy did not improve from
0.92703
104/104 - 85s - loss: 0.0026 - sparse_categorical_accuracy: 1.0000 -
val_loss: 0.2904 - val_sparse_categorical_accuracy: 0.9216 - 85s/epoch -
813ms/step
```

Надрукуємо точність моделі на навчальному та валідаційному наборах:

```
scores = model2.evaluate(train_features, y_train, verbose=0)
print("Metrics and scores (train)")
print(model2.metrics_names)
print(scores)

scores = model2.evaluate(val_features, y_val, verbose=0)
print("Metrics and scores (val)")
print(model2.metrics_names)
print(scores)
```

Результати показують точність у 100 % для навчального набору та 92,7 % – для валідаційного, що є непоганим результатом. Слід зазначити, що точність у 100 % на навчальному наборі часто може свідчити про перенавчання, але в цьому випадку точність на валідаційному наборі також висока.

```
Metrics and scores (train)
['loss', 'sparse_categorical_accuracy']
[0.000749908562283963, 1.0]
Metrics and scores (val)
['loss', 'sparse_categorical_accuracy']
[0.28286245465278625, 0.9270270466804504]
```

Перевіримо точність класифікації тестового набору, обробивши послідовно кожне зображення з нього та зібравши статистику за окремими класами:

```
with open("test.txt", 'r') as f:
    testfold = [line.rstrip('\n') for line in f]

correct_test = 0
  all_test = 0
  stat_test = {}
  print("Each test image evaluation:")
  for line in testfold:
    x_test=[]
    etalon_label = class_name(line)
    img = cv2.imread(path + image_name(line) + ".jpg")

    if (img is not None):

      if etalon_label not in stat_test.keys():
        stat_test[etalon_label] = 0
      all_test = all_test + 1

      proc_image = cv2.resize(img, (IMAGE_SIZE, IMAGE_SIZE))
      x_test.append(preprocess_input(proc_image))
      x_test = np.array(x_test, np.float16)# / 255.
      test_features = model.predict(x_test)

      scores = model2.predict(test_features)
```

```
            scores_list = scores.tolist()
            flattened = [val for sublist in scores_list for val in sublist]
            max_ = max(flattened)
            if (labels[flattened.index(max_)] == etalon_label):
                correct_test = correct_test + 1
                stat_test[etalon_label] = stat_test[etalon_label] + 1
        else:
            print(line + "Image read error")

    for k, v in stat_test.items():
        print("{:<8} {:<15}".format(k, v))

    print("Correct test = " + str(correct_test))
    print("All test = " + str(all_test))
    print(1.0 * correct_test/all_test)
```

Результат виконання цієї клітинки наведено нижче. Як можна помітити, загальна точність моделі становила 88,47 %. Ураховуючи, що кожен клас тестового набору містить приблизно по 100 зображень, треба звернути увагу на незадовільну роботу моделі для класів american_pit_bull_terrier, Ragdoll і staffordshire_bull_terrier.

```
Each test image evaluation:
Abyssinian 80
american_bulldog 90
american_pit_bull_terrier 46
basset_hound 94
beagle   85
Bengal   73
Birman   81
Bombay   79
boxer    93
British_Shorthair 79
chihuahua 86
Egyptian_Mau 85
english_cocker_spaniel 95
english_setter 88
german_shorthaired 99
great_pyrenees 98
havanese 93
japanese_chin 100
keeshond 98
leonberger 96
```

Maine_Coon 77
miniature_pinscher 90
newfoundland 100
Persian 88
pomeranian 96
pug   91
Ragdoll 65
Russian_Blue 70
saint_bernard 96
samoyed 99
scottish_terrier 99
shiba_inu 98
Siamese 93
Sphynx   97
staffordshire_bull_terrier 61
wheaten_terrier 95
yorkshire_terrier 93
Correct test = 3246
All test = 3669
0.884709730171709

Практична частина

Індивідуальні завдання

1. Повторіть наведений приклад.

2. Додайте в кінці код для класифікації трьох вибраних зображень різних порід собак чи котів з Інтернету.

3. Дослідіть, як впливає встановлення параметру STEP на якість моделі.

4. Ураховуючи досвід попередніх лабораторних робіт, спробуйте змінити параметри, щоб отримати кращу модель.

5. Виконайте точне налаштування (fine tuning) мережі. Чи поліпшує це результати?

Підготуйте звіт за результатами виконаної роботи. Звіт має містити титульний аркуш; опис виконаних завдань із коментарями, кодом, скриншотами; висновки за кожним завданням та з лабораторної роботи загалом. Висновки не повинні дублювати мету.

Підготуйтеся до захисту роботи за контрольними запитаннями. Запитання, позначені символом *, передбачають вивчення додаткової літератури.

Контрольні запитання до лабораторної роботи 5

1*. Для чого використовують шари BatchNormalization?

2. Що становить трансферне навчання?

3. Як виконують точне налаштування (fine tuning)?

4*. Навіщо "заморожувати" значення вагових коефіцієнтів під час трансферного навчання? Що може трапитися, якщо цього не робити?

5. Чому точне налаштування повинно відбуватися з маленькою швидкістю навчання?

6. Яку роль відіграють конволюційні шари попередньо навченої моделі в трансферному навчанні?

7. Чому останні повнозв'язні шари попередньо навченої моделі найчастіше не переносять у нову модель?

Рекомендована література

Основна

1. Субботін С. О. Нейронні мережі: теорія та практика : навч. посіб. / С. О. Субботін. – Житомир : Вид. О. О. Євенок, 2020. – 184 с.

2. Терейковський І. А. Штучні нейронні мережі: базові положення [Електронний ресурс] : навчальний посібник для здобувачів ступеня бакалавра за освітньою програмою "Системне програмування та спеціалізовані комп'ютерні системи" спеціальності 123 Комп'ютерна інженерія / І. А. Терейковський, Д. А. Бушуєв, Л. О. Терейковська ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,5 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2022. – 123 с.

3. Ткаліченко С. В. Штучні нейронні мережі : навч. посіб. / С. В. Ткаліченко. – Кривий Ріг : Державний університет економіки і технологій, 2023. – 150 с.

Додаткова

4. Руденко О. Г. Штучні нейронні мережі : навч. посіб. для студ. вищ. навч. закл. / О. Г. Руденко, Є. В. Бодянський. – Київ : Компанія СМІТ, 2006. – 404 с.

5. Chollet F. Deep Learning with Python. – Manning Publications Co, 2017. – 384 p.

6. Gorokhovatskyi O. Explanation of CNN Image Classifiers with Hiding Parts / O. Gorokhovatskyi, O. Peredrii, V. Gorokhovatskyi, N. Vlasenko // Explainable Deep Learning AI / ed. by J. Benois-Pineau, R. Bourqui, D. Petkovic, G. Quenot. – Academic Press, 2023. – P. 125–146.

7. Gorokhovatskyi O. Interpretability of Neural Network Binary Classification with Part Analysis / O. Gorokhovatskyi, O. Peredrii, V. Gorokhovatskyi // The Third IEEE International Conference on DataStream Mining & Processing, 21 – 25 August 2020, Lviv, Ukraine. – 2020. – P. 136–141.

8. Gorokhovatskyi O. Recursive Division of Image for Explanation of Shallow CNN Models / O. Gorokhovatskyi, O. Peredrii // Del Bimbo A. et al. (eds) Pattern Recognition. ICPR International Workshops and Challenges. ICPR 2021. Lecture Notes in Computer Science. – 2021. – Vol. 12663. – P. 274–286.

Інформаційні ресурси

9. Удовенко С. Нейромережева обробка даних [Електронний ресурс] / С. Удовенко, О. Гороховатський. – Режим доступу : https://pns.hneu.edu.ua/course/view.php?id=8006.

10. Goodfellow I. Deep Learning [Electronic resource] / I. Goodfellow, Y. Bengio, A. Courville. – Access mode : https://www.deeplearningbook.org/.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

НЕЙРОМЕРЕЖЕВА ОБРОБКА ДАНИХ

Методичні рекомендації до лабораторних робіт

для здобувачів вищої освіти спеціальності 126 "Інформаційні системи та технології" освітньої програми "Інформаційні системи та технології" першого (бакалаврського) рівня

Самостійне електронне текстове мережеве видання

Укладачі: **Удовенко** Сергій Григорович
Гороховатський Олексій Володимирович

Відповідальний за видання *С. Г. Удовенко*

Редактор *Н. Г. Войчук*

Коректор *В. О. Дмитрієва*

План 2024 р. Поз. № 102 ЕВ. Обсяг 58 с.

Видавець і виготовлювач – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Науки, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру ***ДК № 4853 від 20.02.2015 р.***