ЛАБОРАТОРНА РОБОТА № 5

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

Мета роботи: *використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовувати прості нейронні мережі.*

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Теоретичні відомості подані на лекціях. Також доцільно вивчити матеріал поданий в літературі:

Джоши Пратик. Искусственный интеллект с примерами на Python. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 448 с. - Парал. тит. англ. ISBN 978-5-907114-41-8 (рус.)

Можна використовувати Google Colab або Jupiter Notebook.

https://python-scripts.com/intro-to-neural-networks

Штучна нейронна мережа - це модель, призначена для імітації процесів навчання, як і у мозку людини.

Штучні нейронні мережі проектуються таким чином, щоб вони могли розпізнавати базові образи (закономірності, стійкі взаємозв'язки, приховані в даних) та навчатися на них. Вони можуть застосовуватися для вирішення різних завдань, таких як класифікація, регресія, сегментація даних та ін. Перш ніж надати дані нейронній мережі, ми повинні перетворити їх у числову форму. Наприклад, ми можемо мати справу з даними різної природи, включаючи візуальні та текстові дані, часові ряди тощо. Нам доводиться приймати рішення щодо того, як слід представляти завдання, щоб вони були зрозумілі нейронним мережам.

Створення нейронної мережі

Процес навчання людини має ієрархічний характер. У нейронної мережі нашого мозку цей процес здійснюється в кілька стадій, для кожної з яких характерний свій ступінь гранулярності. На одних стадіях йде навчання простим речам, на інших – складнішим. Як приклад розглянемо візуальне розпізнавання об'єкта. Коли ми дивимося на ящик, спочатку ми просто ідентифікуємо такі його елементи, як кути та ребра. На наступній стадії ідентифікується форма ящика, а на наступній - що являє собою об'єкт. Цей процес відбувається по-різному для різних завдань, але, мабуть, ідея вам зрозуміла. Створюючи таку ієрархію, людський мозок швидко розділяє поняття та ідентифікує цей об'єкт.

Щоб імітувати процес навчання людини, при побудові штучних нейронних мереж використовують шари нейронів. Ідея цих нейронів

підказана біологічними процесами, про які йшлося вище. Кожен шар штучної нейронної мережі є множина незалежних нейронів. Кожен нейрон деякого шару з'єднаний із нейронами суміжного шару.

Тренування нейронної мережі

Якщо ми маємо справу з N-вимірними вхідними даними, то вхідний шар складатиметься з N нейронів. Якщо серед наших навчальних (тренувальних) даних виділяються M різних класів, то вихідний шар буде мати M нейронів.

Прошарки що містяться між вхідним і вихідним шаром носять назву – приховані прошарки. Проста нейронна мережа складається із невеликої кількості прошарків, а глибока нейронна мережа - з великої множини шарів.

Розглянемо випадок, коли хочемо використовувати нейронну мережу для класифікації даних. Першим кроком є збір відповідних тренувальних даних та їх маркування. Кожен нейрон діє як проста функція, і нейронна мережа тренується до того часу, поки помилка стане менше певного заданого значення. Як помилку переважно використовують різницю між передбаченим і фактичним виходами. Виходячи з того, наскільки велика помилка, нейронна мережа коригує саму себе і повторно навчається доти, доки не наблизиться до рішення.

У цій ЛР ми будемо використовувати бібліотеку NeuroLab, більш детальну інформацію про яку можна отримати на сайті https://pythonhosted. org/neurolab. Щоб установити цю бібліотеку, виконайте у вікні свого терміналу наступну команду:

```
pip3 install neurolab
```

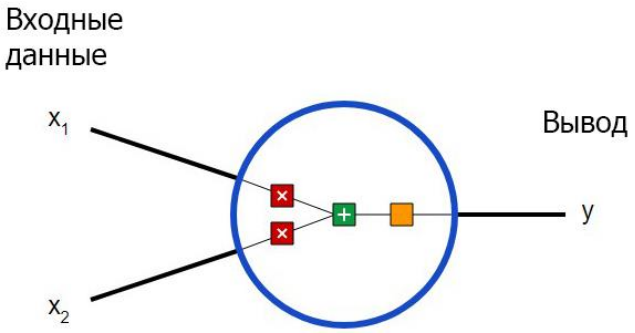
Створення класифікатора на основі перцептрону

Перцептрон - будівельна цегла штучної нейронної мережі. Він є одиночний нейрон, який отримує вхідні дані (сигнал), виконує над ними обчислення і видає вихідний сигнал. Для ухвалення рішень перцептрон застосовує просту функцію. Припустимо, ми маємо справу з N-мірною вхідною точкою даних. Перцептрон обчислює виважену суму N чисел, після чого додає до них константу для отримання вихідного результату. Ця константа називається *зміщенням* нейрону. Цікаво відзначити, що прості перцептрони використовуються для проектування дуже складних глибоких нейронних мереж.

2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

Завдання 2.1. Створити простий нейрон

Нейрон приймає вхідні дані, виконує із нею певні математичні операції, та виводить результат. Нейрон з двома вхідними даними виглядає так:



Створення нейронних блоків

Тут відбуваються три речі. По-перше, кожен вхід множиться на вагу (на схемі позначений червоним):

$$x_1 \rightarrow x_1 * w_1$$

$$x_2 \rightarrow x_2 * w_2$$

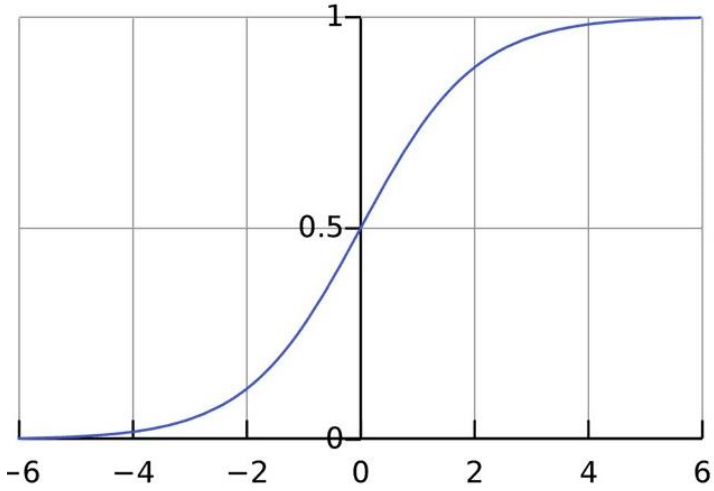
Потім усі зважені входи складаються разом зі зміщенням b (на схемі позначений зеленим):

$$(x_1 * w_1) + (x_2 * w_2) + b$$

Зрештою, сума передається через функцію активації (на схемі позначена жовтим):

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

Функція активації використовується для підключення незв'язаних вхідних даних із виходом, у якого проста та передбачувана форма. Як правило, як функція активації найбільш часто використовується функція сигмоїди:



Функція сигмоїди

Функція сигмоїда виводить лише числа у діапазоні (0, 1). Ви можете сприймати це як компресію від (−∞, +∞) до (0, 1). Великі негативні числа стають ~0, а великі позитивні числа стають ~1.

Простий приклад роботи з нейронами в Python

Припустимо, у нас є нейрон із двома входами, який використовує функцію активації сигмоїду та має наступні параметри:

$$w = [0, 1]$$

$$b = 4$$

w = [0,1] - це просто один із способів написання $w_1 = 0$, $w_2 = 1$ у векторній формі. Надамо нейрону вхід зі значенням x = [2, 3]. Для більш компактного уявлення буде використано скалярне множення.

$$\begin{aligned}(w \cdot x) + b &= ((w_1 * x_1) + (w_2 * x_2)) + b \\ &= 0 * 2 + 1 * 3 + 4 \\ &= 7\end{aligned}$$

$$y = f(w \cdot x + b) = f(7) = \boxed{0.999}$$

З урахуванням, що вхід був x = [2, 3], вихід дорівнює 0.999. Такий процес передачі вхідних даних для отримання виходу називається прямим розповсюдженням, або feedforward.

Створення нейрона з нуля в Python

```
import numpy as np

def sigmoid(x):
  # Наша функція активації: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))

class Neuron:
  def __init__(self, weights, bias):
    self.weights = weights
    self.bias = bias

  def feedforward(self, inputs):
    # Вхідні дані про вагу, додавання зміщення
    # і подальше використання функції активації

    total = np.dot(self.weights, inputs) + self.bias
    return sigmoid(total)
```

```
weights = np.array([0, 1])  # w1 = 0, w2 = 1
bias = 4  # b = 4
n = Neuron(weights, bias)

x = np.array([2, 3])  # x1 = 2, x2 = 3
print(n.feedforward(x))
```

Код програми та отриманий результат занести у звіт.

Завдання 2.2. Створити просту нейронну мережу для передбачення статі людини

Нейронна мережа по суті є групою пов'язаних між собою нейронів. Проста нейронна мережа виглядає так:

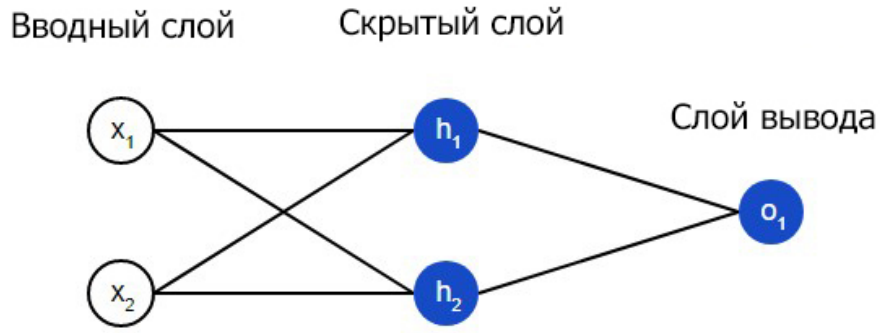


Схема нейронної мережі

На вступному шарі мережі два входи – x_1 та x_2. На прихованому шарі два нейтрони - h_1 і h_2. На шарі виведення є один нейрон – o_1. Зверніть увагу, що вхідні дані для o_1 є результатами виведення h_1 і h_2. Таким чином будується нейромережа.

Давайте використовуємо продемонстровану вище мережу і уявімо, що всі нейрони мають однакову вагу $w = [0, 1]$, однакове зміщення $b = 0$ і ту саму функцію активації сигмоїду. Нехай h_1, h_2 та o_1 самі відзначать результати виведення представлених ними нейронів.

Що трапиться, якщо як введення буде використано значення $x = [2, 3]$?

$$\begin{aligned} h_1 = h_2 &= f(w \cdot x + b) \\ &= f((0 * 2) + (1 * 3) + 0) \\ &= f(3) \\ &= 0.9526 \end{aligned}$$

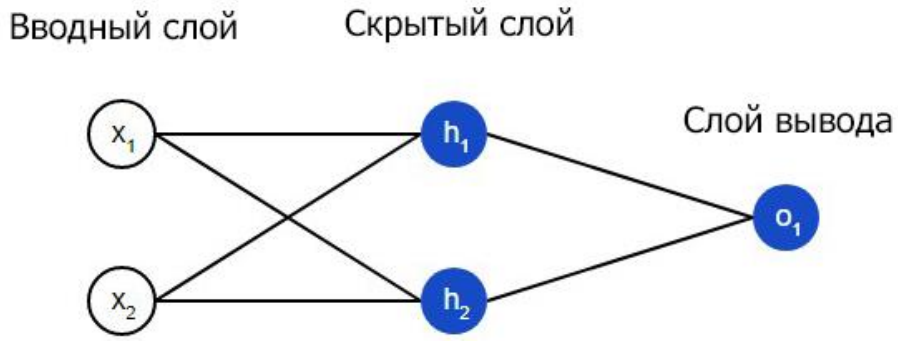
$$\begin{aligned} o_1 &= f(w \cdot [h_1, h_2] + b) \\ &= f((0 * h_1) + (1 * h_2) + 0) \\ &= f(0.9526) \\ &= \boxed{0.7216} \end{aligned}$$

Результат виведення нейронної мережі для вхідного значення $x = [2, 3]$ становить 0.7216.

Нейронна мережа може мати будь-яку кількість шарів з будь-якою кількістю нейронів у цих шарах.

Створення нейронної мережі прямого поширення FeedForward

Далі буде показано, як реалізувати пряме розповсюдження ваг по відношенню до нейронної мережі. Як опорна точка буде використана наступна схема нейронної мережі:




```
import numpy as np

# ... Тут код із попереднього завдання
#  Увага!  Наступний клас OurNeuralNetwork називаєте з використанням
# замість Our вашого прізвища англ. мовою, наприклад: IvanovNeuralNetwork

class OurNeuralNetwork:

   def __init__(self):
      weights = np.array([0, 1])
      bias = 0

      # Класс Neuron із попереднього завдання
      self.h1 = Neuron(weights, bias)
      self.h2 = Neuron(weights, bias)
      self.o1 = Neuron(weights, bias)

   def feedforward(self, x):
      out_h1 = self.h1.feedforward(x)
      out_h2 = self.h2.feedforward(x)

      # Входи для o1 є виходами h1 и h2
      out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))

      return out_o1

network = OurNeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x))  # 0.7216325609518421
```

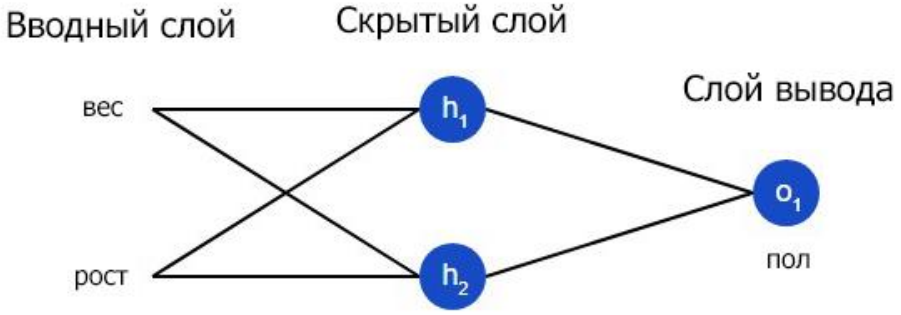
Якщо отримали 0.7216, то все працює правильно.

Приклад тренування нейронної мережі - мінімізація втрат.

Припустимо, у нас є такі параметри:

Ім'я/Name	Вага/Weight (фунти)	Зріст/Height (дюйми)	Стать/Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

Давайте натренуємо нейронну мережу таким чином, щоб вона передбачала стать заданої людини в залежності від її ваги і зросту.



Тренування нейронної мережі

Чоловіки Male будуть представлені як 0, а жінки Female як 1. Для простоти представлення дані також будуть дещо зміщені.

Ім'я/Name	Вага/Weight (мінус 135)	Зріст/Height (мінус 66)	Стать/Gender
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

Для оптимізації тут зроблено довільні зміщення 135 і 66. Проте, зазвичай для зміщення вибираються середні показники.

Втрати

Перед тренуванням нейронної мережі потрібно вибрати спосіб оцінки того, наскільки добре мережа справляється із завданнями. Це необхідно для подальших спроб виконувати поставлене завдання краще. Такий принцип розрахунку якості – через втрати.

В даному випадку буде використовуватися середньоквадратична помилка (MSE) втрати:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

Це формула середньоквадратичної помилки

Тут:

n – число об'єктів, що розглядаються, яке в даному випадку дорівнює 4. Це Alice, Bob, Charlie і Diana;

y – змінні, які будуть передбачені. У даному випадку це стать людини;

y_{true} – справжнє значення змінної, тобто так звану правильну відповідь. Наприклад, для Alice значення ytrue буде 1, тобто Female;

y_{pred} – передбачуване значення змінної. Це результат виведення мережі.

$(y_{true} - y_{pred})^2$ називають квадратичною помилкою (MSE). Тут функція втрат просто бере середнє значення з усіх квадратичних помилок. Звідси й назва помилки. Чим краще передбачення, тим нижчі втрати.

Найкращим прогнозам відповідають найменші втрати.

Тренування нейронної мережі має на меті прагнення мінімізувати її втрати.

Приклад підрахунку втрат у тренуванні нейронної мережі

Скажімо, наша мережа завжди видає 0. Іншими словами, вона впевнена, що всі люди – чоловіки. Якою буде втрата?

Ім'я/Name	y_{true}	y_{pred}	$(y_{true} - y_{pred})^2$
Alice	1	0	1
Bob	0	0	0
Charlie	0	0	0
Diana	1	0	1

Підрахунок втрат

$$\text{MSE} = \frac{1}{4}(1 + 0 + 0 + 1) = \boxed{0.5}$$

Python код для середньоквадратичної помилки (MSE)

Нижче наведено код для підрахунку втрат:

```
import numpy as np

def mse_loss(y_true, y_pred):
    # y_true и y_pred є масивами numpy з однаковою довжиною
    return ((y_true - y_pred) ** 2).mean()

y_true = np.array([1, 0, 0, 1])
y_pred = np.array([0, 0, 0, 0])

print(mse_loss(y_true, y_pred))  # 0.5
```

При виникненні складнощів з розумінням роботи коду варто ознайомитись з quickstart у NumPy для операцій з масивами.

Тренування нейронної мережі - багатоваріантні обчислення

Поточна ціль зрозуміла - це мінімізація втрат нейронної мережі. Тепер стало ясно, що вплинути на прогнози мережі можна за допомогою зміни її ваги та зміщення. Проте як мінімізувати втрати?

Тут будуть використані багатоваріантні обчислення. Якщо ви не знайомі з цією темою, фрагменти з математичними обчисленнями можна пропускати.

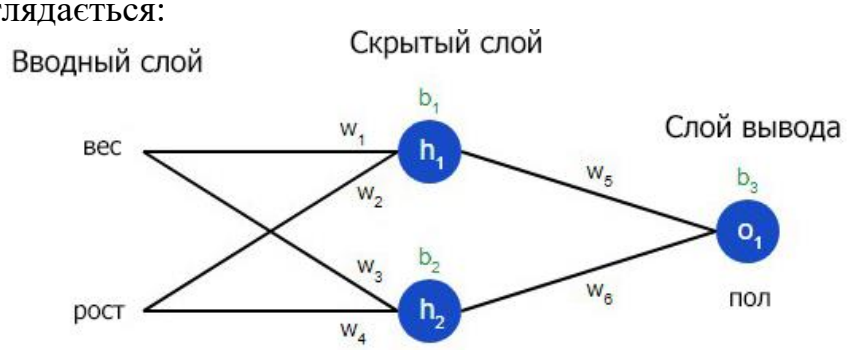
Для простоти давайте уявімо, що в наборі даних розглядається тільки Alice:

Ім'я/Name	Вага/Weight (мінус 135)	Зріст /Height (мінус 66)	Стать/Gender
Alice	-2	-1	1

Потім середньоквадратична помилка втрат буде просто квадратичною помилкою для Alice:

$$\text{MSE} = \frac{1}{1}\sum_{i=1}^{1}(y_{true} - y_{pred})^2 = (y_{true} - y_{pred})^2 = (1 - y_{pred})^2$$

Ще один спосіб розуміння втрат – уявлення їх як функції ваги та зміщення. Давайте позначимо кожну вагу і зміщення в мережі, що розглядається:



Тренування нейронної мережі схема

Потім можна прописати втрату як багатоваріантну функцію:

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

Припустимо, що нам потрібно трохи відредагувати w_1. У такому разі, як зміниться втрата L після внесення змін у w_1?

На це запитання може відповісти приватна (часткова) похідна $\frac{\partial L}{\partial w_1}$. Як же її вирахувати?

Для початку, перепишемо часткову похідну в контексті $\frac{\partial y_{pred}}{\partial w_1}$:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1}$$

Ці обчислення можливі завдяки диференціюванню складної функції.

Підрахувати $\frac{\partial L}{\partial y_{pred}}$ можна завдяки обчисленій вище `L = (1 - y_pred)^2`:

$$\frac{\partial L}{\partial y_{pred}} = \frac{\partial (1 - y_{pred})^2}{\partial y_{pred}} = \boxed{-2(1 - y_{pred})}$$

Тепер, давайте визначимо, що робити з обчисленням часткової похідної $\frac{\partial y_{pred}}{\partial w_1}$. Як і раніше, дозволимо h_1, h_2, o_1 стати результатами виходу нейронів, які вони представляють. Подальші обчислення:

$$y_{pred} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3)$$

Як було зазначено раніше, тут f є функцією активації сигмоїдою.

Оскільки w_1 впливає лише на h_1, а не на h_2, можна записати:

$$\frac{\partial y_{pred}}{\partial w_1} = \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial y_{pred}}{\partial h_1} = \boxed{w_5 * f'(w_5 h_1 + w_6 h_2 + b_3)}$$

(Використання диференціювання складної функції).

Ті самі дії проводяться для обчислення часткової похідної $\frac{\partial h_1}{\partial w_1}$:

$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1)$$

$$\frac{\partial h_1}{\partial w_1} = \boxed{x_1 * f'(w_1 x_1 + w_2 x_2 + b_1)}$$

(Ще одне використання диференціювання складної функції).

В даному випадку x_1 – вага, а x_2 – зріст. Тут $f'(x)$ як похідна функції сигмоїди зустрічається вдруге. Спробуємо вивести її:

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = f(x) * (1 - f(x))$$

Функція $f'(x)$ у такому вигляді буде використана дещо пізніше.

Тепер формули нейронної мережі розбиті на кілька частин, які будуть оптимальні для підрахунку:

$$\boxed{\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}}$$

Ця система підрахунку часткових похідних під час роботи у зворотному порядку відома як **метод зворотного поширення помилки, або backprop**.

Приклад підрахунку часткових похідних
У цьому прикладі також буде задіяна тільки Alice:

Ім'я/Name	Вага/Weight (мінус 135)	Зріст /Height (мінус 66)	Стать/Gender
Alice	-2	-1	1

Тут вага буде представлена як 1, а зсув як 0. Якщо виконаємо *пряме поширення (feedforward)* через мережу, отримаємо:

$$h_2 = f(w_3 x_1 + w_4 x_2 + b_2) = 0.0474$$

$$\begin{aligned} h_1 &= f(w_1 x_1 + w_2 x_2 + b_1) \\ &= f(-2 + -1 + 0) \\ &= 0.0474 \end{aligned} \qquad \begin{aligned} o_1 &= f(w_5 h_1 + w_6 h_2 + b_3) \\ &= f(0.0474 + 0.0474 + 0) \\ &= 0.524 \end{aligned}$$

Вихід нейронної мережі $y_{pred} = 0.524$. Це дає нам слабке уявлення про те, чи розглядається чоловік *Male (0)*, або жінка *Female (1)*. Давайте підрахуємо $\frac{\partial L}{\partial w_1}$:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\begin{aligned} \frac{\partial L}{\partial y_{pred}} &= -2(1 - y_{pred}) \\ &= -2(1 - 0.524) \\ &= -0.952 \end{aligned}$$

$$\begin{aligned} \frac{\partial y_{pred}}{\partial h_1} &= w_5 * f'(w_5 h_1 + w_6 h_2 + b_3) \\ &= 1 * f'(0.0474 + 0.0474 + 0) \\ &= f(0.0948) * (1 - f(0.0948)) \\ &= 0.249 \end{aligned}$$

$$\begin{aligned} \frac{\partial h_1}{\partial w_1} &= x_1 * f'(w_1 x_1 + w_2 x_2 + b_1) \\ &= -2 * f'(-2 + -1 + 0) \\ &= -2 * f(-3) * (1 - f(-3)) \\ &= -0.0904 \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial w_1} &= -0.952 * 0.249 * -0.0904 \\ &= \boxed{0.0214} \end{aligned}$$

Нагадування: ми вивели *f'(x) = f(x)*(1 - f(x))* раніше для нашої функції активації сигмоїди.

Результат говорить про те, що якщо ми збираємося збільшити w_1, то L трохи збільшується в результаті.

Тренування нейронної мережі: Стохастичний градієнтний спуск

У нас є всі необхідні інструменти для тренування нейронної мережі. Ми використовуємо алгоритм оптимізації під назвою стохастичний градієнтний

спуск (SGD), який говорить нам, як саме поміняти вагу та зміщення для мінімізації втрат. По суті це відображається в наступному рівнянні:

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

η є константою під назвою **оцінка навчання**, що контролює швидкість навчання. Все що ми робимо, так це віднімаємо $\frac{\partial L}{\partial w_1}$ з w_1:

- Якщо $\frac{\partial L}{\partial w_1}$ позитивні, w_1 зменшиться, що призведе до зменшення L.
- Якщо $\frac{\partial L}{\partial w_1}$ негативна, w_1 збільшиться, що призведе до зменшення L.

Якщо ми застосуємо це на кожну вагу та зміщення в мережі, **втрата буде поступово знижуватися, а показники мережі значно покращаться**.

Наш процес тренування буде виглядати так:

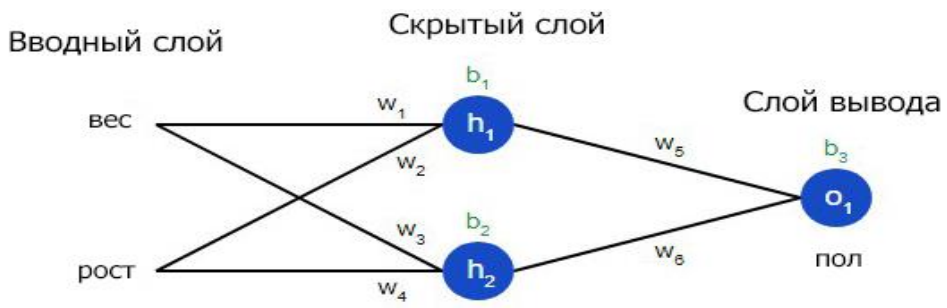
1. Вибираємо один пункт із нашого набору даних. Це те, що робить його стохастичним градієнтним спуском. Ми обробляємо лише один пункт за раз;
2. Підраховуємо всі часткові похідні втрати за вагою чи зміщенням. Це може бути $\frac{\partial L}{\partial w_1}$, $\frac{\partial L}{\partial w_2}$ тощо;
3. Використовуємо рівняння оновлення для оновлення кожної ваги та зміщення;
4. Повертаємось до першого пункту.

Погляньмо, як це працює на практиці.

Створення нейронної мережі з нуля на Python

Нарешті, ми реалізуємо готову нейронну мережу:

Ім'я/Name	Вага/Weight (мінус 135)	Зріст /Height (мінус 66)	Стать/Gender
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

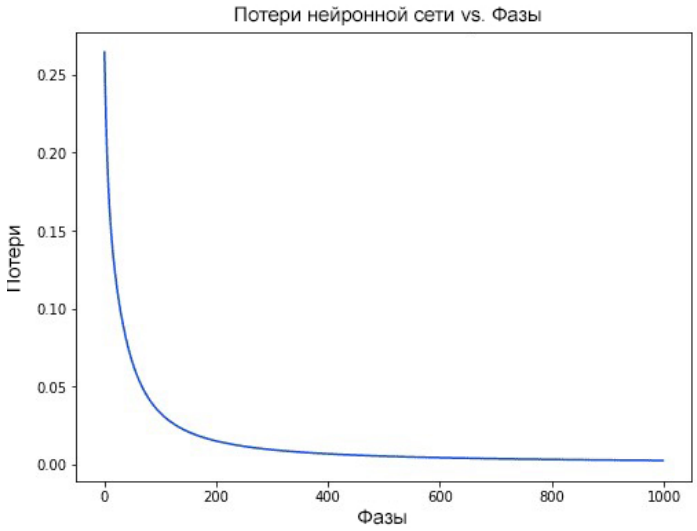


```
import numpy as np

def sigmoid(x):
  # Функція активації sigmoid:: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))

def deriv_sigmoid(x):
  # Похідна від sigmoid: f'(x) = f(x) * (1 - f(x))
  fx = sigmoid(x)
  return fx * (1 - fx)

def mse_loss(y_true, y_pred):
  # y_true и y_pred є масивами numpy з одинаковою довжиною
  return ((y_true - y_pred) ** 2).mean()

class OurNeuralNetwork:
  """
  Нейронна мережа, у якої:
    - 2 входи
    - прихований шар з двома нейронами (h1, h2)
    - шар виходу з одним нейроном (o1)

  *** ВАЖЛИВО ***:
  Код нижче написаний як простий, навчальний. НЕ оптимальний.
  Справжній код нейронної мережі виглядає не так. НЕ ВИКОРИСТОВУЙТЕ цей код у подальшому.
  Замість цього, прочитайте та запустіть його, щоб зрозуміти, як працює ця мережа.
  """
  def __init__(self):
    # Вага
    self.w1 = np.random.normal()
    self.w2 = np.random.normal()
    self.w3 = np.random.normal()
    self.w4 = np.random.normal()
    self.w5 = np.random.normal()
    self.w6 = np.random.normal()

    # Зміщення
    self.b1 = np.random.normal()
    self.b2 = np.random.normal()
    self.b3 = np.random.normal()

  def feedforward(self, x):
    # x є масивом numpy з двома елементами
    h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
    h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
    o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
    return o1

  def train(self, data, all_y_trues):
    """
```

```
- data is a (n x 2) numpy array, n = # of samples in the dataset.
- all_y_trues is a numpy array with n elements.
  Elements in all_y_trues correspond to those in data.
"""
learn_rate = 0.1
epochs = 1000 # кількість циклів у всьому наборі даних

for epoch in range(epochs):
  for x, y_true in zip(data, all_y_trues):
    # --- Виконуємо зворотній зв'язок (ці значання нам потрібні в подальшому )
    sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
    h1 = sigmoid(sum_h1)

    sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.b2
    h2 = sigmoid(sum_h2)

    sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
    o1 = sigmoid(sum_o1)
    y_pred = o1

    # --- Підрахунок часткових похідних
    # --- Найменування: d_L_d_w1 означає "частково L / частково w1"
    d_L_d_ypred = -2 * (y_true - y_pred)

    # Нейрон o1
    d_ypred_d_w5 = h1 * deriv_sigmoid(sum_o1)
    d_ypred_d_w6 = h2 * deriv_sigmoid(sum_o1)
    d_ypred_d_b3 = deriv_sigmoid(sum_o1)

    d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
    d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)

    # Нейрон h1
    d_h1_d_w1 = x[0] * deriv_sigmoid(sum_h1)
    d_h1_d_w2 = x[1] * deriv_sigmoid(sum_h1)
    d_h1_d_b1 = deriv_sigmoid(sum_h1)

    # Нейрон h2
    d_h2_d_w3 = x[0] * deriv_sigmoid(sum_h2)
    d_h2_d_w4 = x[1] * deriv_sigmoid(sum_h2)
    d_h2_d_b2 = deriv_sigmoid(sum_h2)

    # --- Оновлюємо вагу і зміщення
    # Нейрон h1
    self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w1
    self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w2
    self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_b1

    # Нейрон h2
    self.w3 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w3
    self.w4 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w4
```

```
            self.b2 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_b2

            # Нейрон o1
            self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_w5
            self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_w6
            self.b3 -= learn_rate * d_L_d_ypred * d_ypred_d_b3

        # --- Підраховуємо загальні втрати в кінці кожної фази
        if epoch % 10 == 0:
            y_preds = np.apply_along_axis(self.feedforward, 1, data)
            loss = mse_loss(all_y_trues, y_preds)
            print("Epoch %d loss: %.3f" % (epoch, loss))

# Задання набору даних
data = np.array([
    [-2, -1],    # Alice
    [25, 6],     # Bob
    [17, 4],     # Charlie
    [-15, -6], # Diana
])

all_y_trues = np.array([
    1, # Alice
    0, # Bob
    0, # Charlie
    1, # Diana
])

# Тренуємо вашу нейронну мережу!
network = OurNeuralNetwork()
network.train(data, all_y_trues)
```

Втрати постійно зменшуються в міру того, як навчається нейронна мережа:



Тепер ми можемо використовувати нейронну мережу для передбачення статі:

```
# Робимо передбачення
emily = np.array([-7, -3])  # 128 фунтов, 63 дюйма
frank = np.array([20, 2])  # 155 фунтов, 68 дюймов
print("Emily: %.3f" % network.feedforward(emily))  # 0.951 - F
print("Frank: %.3f" % network.feedforward(frank))  # 0.039 - M
```

Оцініть результати.

Текст програми занесіть у звіт.

Програмний код збережіть під назвою LR_5_task_2.py

Зробіть висновки у яких укажіть:
- призначення функції активації;
- про можливості нейронних мереж прямого поширення.

Завдання 2.3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

Розробіть класифікатор на основі перцептрону з використанням бібліотеки NeuroLab для файлу даних `data_perceptron.txt`.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі файли:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
```

Завантажте вхідні дані з файлу `data_perceptron.txt`. Кожен рядок містить числа розділені пробілом, із яких перші два числа – ознаки, а третє – мітка (маркер класу).

```
# Завантаження вхідних даних
text = np.loadtxt('data_perceptron.txt')
```

Розділимо текст на точки даних та мітки.

```
# Поділ точок даних та міток
data = text[:, :2]
labels = text[:, 2].reshape((text.shape[0], 1))
```

Побудуємо графік вхідних точок даних.

```
# Побудова графіка вхідних даних
plt.figure()
plt.scatter(data[:,0], data[:,1])
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')
```

Визначимо максимальне та мінімальне значення, які можуть досягатися у кожному вимірі.

```
# Визначення максимального та мінімального значень для кожного виміру
dim1_min, dim1_max, dim2_min, dim2_max = 0, 1, 0, 1
```

Оскільки дані розділені на два класи, для представлення вихідного результату потрібно лише один біт. В результаті вихідний шар міститиме лише один нейрон.

```
# Кількість нейронів у вихідному шарі
num_output = labels.shape[1]
```

У нашому наборі точки даних – двомірні. Визначимо перцептрон з двома вхідними нейронами, по одному для кожного виміру.

```
# Визначення перцептрону з двома вхідними нейронами (оскільки
# Вхідні дані - двовимірні)
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
perceptron = nl.net.newp([dim1, dim2], num_output)
```

Навчимо перцептрон за допомогою тренувальних даних.

```
# Тренування перцептрону з використанням наших даних
error_progress = perceptron.train(data, labels, epochs=100,
                                  show=20, lr=0.03)
```

Відобразимо графік процесу навчання, використовуючи метрику помилки.

```
# Побудова графіка процесу навчання
```

```
plt.figure()
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка обучения')
plt.title('Изменение ошибки обучения')
plt.grid()
plt.show()
```

У процесі виконання коду на екрані з'являться два графіки.

Занесіть їх у звіт. Підпишіть що міститься на кожному графіку.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_5_task_3.py
Зробіть висновок по другому графіку.

Завдання 2.4. Побудова одношарової нейронної мережі

Створіть одношарову нейронну мережу, що складається з незалежних нейронів, для вхідного файлу `data_simple_nn.txt`.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі файли.

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
```

Вхідні данні містяться у файлі `data_simple_nn.txt`. Кожен рядок цього файлу містить чотири числа. Перші два числа являють собою точку даних, а решта (два) є мітками. Чому нам потрібно використовувати два числа як мітки? Тому що в нашому наборі даних є чотири різних класи, для представлення яких потрібні два біта.

```
# Завантаження вхідних даних
text = np.loadtxt('data_simple_nn.txt')
```

Поділимо дані на точки даних та мітки.

```
# Поділ даних на точки даних та мітки
data = text[:, 0:2]
labels = text[:, 2:]
```

Побудуємо графік вхідних даних.

```
# Побудова графіка вхідних даних
plt.figure()
plt.scatter(data[:,0], data[:,1])
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')
```

Визначимо мінімальне та максимальне значення для кожного виміру. (Нам не потрібно вказувати їх у коді, як це робилося в попередньому завданні).

```
# Мінімальне та максимальне значення для кожного виміру
dim1_min, dim1_max = data[:,0].min(), data[:,0].max()
dim2_min, dim2_max = data[:,1].min(), data[:,1].max()
```

Визначимо кількість нейронів у вихідному шарі.

```
# Визначення кількості нейронів у вихідному шарі
num_output = labels.shape[1]
```

Визначимо одношарову нейронну мережу, використовуючи наведені вище параметри.

```
# Визначення одношарової нейронної мережі
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
nn = nl.net.newp([dim1, dim2], num_output)
```

Навчимо мережу на тренувальних даних

```
error_progress = nn.train(data, labels, epochs=100, show=20, lr=0.03)
```

Побудуйте графік просування процесу навчання.

```
# Побудова графіка просування процесу навчання
plt.figure()
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка обучения')
plt.title('Изменение ошибки обучения')
plt.grid()
```

```
plt.show()
```

Визначимо вибіркові тестові точки даних та запустимо для них нейронну мережу.

```
# Виконання класифікатора на тестових точках даних
print('\nTest results:')
data_test = [[0.4, 4.3], [4.4, 0.6], [4.7, 8.1]]
for item in data_test:
    print(item, '-->', nn.sim([item])[0])
```

У процесі виконання цього коду на екрані відобразяться два рафіка. ***Занесіть їх у звіт. Підпишіть що міститься на кожному графіку.***

Проаналізуйте значення, що виведені у вікні терміналу. Занесіть їх у звіт.

Зробіть висновок по другому графіку та по даних терміналу.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_5_task_4.py

Завдання 2.5. Побудова багатошарової нейронної мережі

Побудуйте багатошарову нейронну мережу, що виконує задачу регресії для тестових даних $y = 3x^2 + 5$.

Для отримання більш високої точності ми повинні надати більшу свободу нейронній мережі. Це означає, що нейронна мережа повинна мати більше одного шару для отримання базових закономірностей, що існують серед тестових даних.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі файли.

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
```

Згенеруємо вибірковий набір точок даних, використовуючи рівняння $y = 3x^2 + 5$, а потім нормалізуємо дані.

```
# Генерація тренувальних даних
min_val = -15
max_val = 15
num_points = 130
x = np.linspace(min_val, max_val, num_points)
y = 3 * np.square(x) + 5
y /= np.linalg.norm(y)
```

Переформуємо наведені вище змінні для створення тренувального набору даних.

```
# Створення даних та міток
data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)
```

Побудуємо графік вхідних даних.

```
# Побудова графіка вхідних даних
plt.figure()
plt.scatter(data, labels)
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')
```

Визначимо нейронну мережу з двома прихованим шарами, що містить 10 нейронів у першому шарі та 6 нейронів у другому шарі. Наше завдання полягає у передбаченні одного значення, тому вихідний шар міститиме лише один нейрон.

```
# Визначення багатошарової нейронної мережі з двома прихованими
# шарами. Перший прихований шар складається із десяти нейронів.
# Другий прихований шар складається з шести нейронів.
# Вихідний шар складається з одного нейрона.
nn = nl.net.newff([[min_val, max_val]], [10, 6, 1])
```

Встановимо метод градієнтного спуску як навчальний алгоритм.

```
# Завдання градієнтного спуску як навчального алгоритму
nn.trainf = nl.train.train_gd
```

Навчимо нейронну мережу, використовуючи згенерований раніше тренувальний набір даних.

```
# Тренування нейронної мережі
error_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)
```

Запустимо нейронну мережу для тренувальних точок даних.

```
# Виконання нейронної мережі на тренувальних даних
output = nn.sim(data)
y_pred = output.reshape(num_points)
```

Побудуємо графік просування процесу навчання.

```
# Побудова графіка помилки навчання
plt.figure()
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка обучения')
plt.title('Изменение ошибки обучения')
```

Побудуємо графік передбачуваних результатів.

```
# Побудова графіка результатів
x_dense = np.linspace(min_val, max_val, num_points * 2)
y_dense_pred =
nn.sim(x_dense.reshape(x_dense.size,1)).reshape(x_dense.size)
plt.figure()
plt.plot(x_dense, y_dense_pred, '-', x, y, '.', x, y_pred, 'p')
plt.title('Фактические и прогнозные значения')
```

У процесі виконання цього коду на екрані з'являться три графіки.

На першому знімку екрана будуть представлені вхідні дані.

На другому знімку екрану представлений графік просування процесу навчання.

На третьому знімку екрана представлений графік передбачуваних результатів, що накладений на графік вхідних даних.

Занесіть їх у звіт..

Як ви побачите, передбачені результати наслідують загальну тенденцію. Якщо ви продовжите навчання мережі та зменшите помилку, то побачите, що відповідність передбачуваних результатів кривої вихідних даних буде характеризуватися ще вищою точністю.

У вікні терміналу з'явиться інформація.

Занесіть її у звіт. Проаналізуйте значення, що виведені у вікні термінал та зробіть по них висновок.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_5_task_5.py

Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту

По аналогії з попереднім завданням, побудуйте багатошарову нейронну мережу, що виконує задачу регресії для тестових даних вашого варіанту. Варіант обирається згідно номеру за списком групи. Варіанти тестових даних указані в таблиці 1. Параметри багатошарової мережі указані в таблиці 2.

Таблиця 1

№ варіанта	Тестові дані	№ варіанта	Тестові дані
Варіант 1	$y = 2x^2+5$	Варіант 16	$y = 5x^2+7$
Варіант 2	$y = 2x^2+6$	Варіант 17	$y = 5x^2+8$
Варіант 3	$y = 2x^2+7$	Варіант 18	$y = 5x^2+9$
Варіант 4	$y = 2x^2+8$	Варіант 19	$y = 3x^2+2x+1$
Варіант 5	$y = 2x^2+9$	Варіант 20	$y = 4x^2+2x+1$
Варіант 6	$y = 2x^2+10$	Варіант 21	$y = 5x^2+2x+1$
Варіант 7	$y = 3x^2+7$	Варіант 22	$y = 6x^2+2x+2$
Варіант 8	$y = 3x^2+8$	Варіант 23	$y = 2x^2+2x+1$
Варіант 9	$y = 3x^2+9$	Варіант 24	$y = 2x^2+2x+4$
Варіант 10	$y = 5x^2+1$	Варіант 25	$y = 3x^2+3x+5$
Варіант 11	$y = 5x^2+2$	Варіант 26	$y = 4x^2+3x+1$
Варіант 12	$y = 5x^2+3$	Варіант 27	$y = 4x^2+3x+2$
Варіант 13	$y = 5x^2+4$	Варіант 28	$y = 4x^2+3x+3$
Варіант 14	$y = 5x^2+5$	Варіант 29	$y = 5x^2+3x+4$
Варіант 15	$y = 5x^2+6$	Варіант 30	$y = 5x^2+3x+5$

Таблиця 2

Номер варіанта	Багатошаровий персептрон		Номер варіанта	Багатошаровий персептрон	
	Кількість шарів	Кількості нейронів у шарах		Кількість шарів	Кількості нейронів у шарах
1	2	3-1	16	2	7-1
2	2	2-1	17	3	7-7-1
3	3	3-3-1	18	3	7-4-1
4	2	5-1	19	2	8-1
5	3	2-2-1	20	3	8-8-1
6	2	10-1	21	2	3-1
7	2	5-1	22	2	2-1
8	3	5-5-1	23	3	3-3-1
9	3	3-5-1	24	2	5-1
10	2	4-1	25	3	2-2-1
11	3	4-4-1	26	2	10-1
12	3	3-4-1	27	2	5-1
13	2	6-1	28	3	5-5-1
14	3	6-3-1	29	3	3-5-1
15	3	8-3-1	30	2	4-1

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

У процесі виконання коду виведіть на екран три графіки.

На першому знімку екрана будуть представлені вхідні дані.

На другому знімку екрану представлений графік просування процесу навчання.

На третьому знімку екрана представлений графік передбачуваних результатів, що накладений на графік вхідних даних.

Занесіть їх у звіт.

У вікні терміналу з'явиться інформація.

Занесіть її у звіт. Проаналізуйте значення, що виведені у вікні термінал та зробіть по них висновок.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_5_task_6.py

Завдання 2.7. Побудова нейронної мережі на основі карти Кохонена, що самоорганізується

По аналогії з попереднім завданням

ДОДАТКОВІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Нейронна мережа Кохонена (SOM)

Карта ознак самоорганізації Кохонена (Kohonen Self-Organizing Map – SOM) відноситься до кластеризаторів, для навчання яких використовуються вибірки образів із заздалегідь не заданою класифікацією.

Задачею мережі є визначення приналежності вхідного вектора ознак s-го екземпляра вибірки $x^s = \{x^s_1, x^s_2, ..., x^s_N\}^T$ до одного з L можливих кластерів, представлених векторними центрами $w_j = \{w_{j1}, w_{j2}, ..., w_{jN}\}^T$, $j = 1, 2, ..., L$, де ^{T} – символ транспонування.

Позначимо i-у компоненту вхідного вектора x^s у момент часу t як $x^s_i(t)$, а вагу i-го входу j-го вузла в момент часу t як $w_{ij}(t)$.

Якщо вузли SOM є лінійними, а вага i-го входу j-го вузла дорівнює w_{ij}, $i = 1, 2, ..., N$, $j = 1, 2, ..., L$, тоді при відповідних значеннях порогів кожен i-та вихід мережі з точністю до несуттєвих постійних буде дорівнює евклідовій відстані d_j між пред'явленим вхідним вектором x^s_i і j-м центром кластера.

Вважається, що вектор x^s належить до j-го кластера, якщо відстань d_j для j-го центра кластера w_j мінімальна, тобто якщо $d_j \le d_k$ для кожного $k \ne j$.

При навчанні НМ пред'являються вхідні вектори без указівки бажаних виходів і корегуються ваги відповідно до алгоритму, що запропонував видатний фінський вчений, академік Теуво Кохонен. Алгоритм Кохонена, що формує SOM, вимагає, щоб біля кожного вузла було визначене поле NE, розмір якого з часом постійно зменшується.

Крок 1. Ініціюються ваги входів вузлів малими випадковими значеннями. Встановлюється початковий розмір поля NE.

Крок 2. Пред'являється новий вхідний вектор x^s.

Крок 3. Обчислюється відстань (метрика) d_j між вхідним вектором і кожним вихідним

вузлом j:

$$d_j = \sum_{i=1}^{N} (x_i^s(t) - w_{ji}(t))^2$$

Крок 4. Визначається вузол j^* з мінімальною відстанню d_j.

Крок 5. Корегуються ваги входів вузлів, що знаходяться в полі $NE_j(t)$ вузла j^*, таким чином, щоб нові значення ваг були рівні:

$$w_{ji}(t+1) = w_{ji}(t) + \eta(t)(x_i^s - w_{ji}(t)),\ j \in NE_j(t),\ i = 1,2,...,N.$$

При цьому коригувальний приріст $\eta(t)$ ($0 < \eta(t) < 1$) повинний спадати зі зростом t.

Крок 6. Якщо збіжність не досягнута, то перейти до кроку 2.

Збіжність вважається досягнутою, якщо ваги стабілізувалися і коригувальний приріст η у кроці 5 знизився до нуля.

Якщо число вхідних векторів у навчальній множині є великим стосовно обраного числа кластерів, то після навчання ваги мережі будуть визначати центри кластерів, розподілені в просторі входів таким чином, що функція щільності цих центрів буде апроксимувати функцію щільності ймовірності вхідних векторів. Крім того, ваги будуть організовані таким чином, що топологічно близькі вузли будуть відповідати фізично близьким (у смислі евклідової відстані) вхідним векторам.

Інтерпретація результатів класифікації SOM

Важливо відзначити, що при класифікації за допомогою SOM, номер вузла, до якого віднесений екземпляр, і фактичний номер його класу в загальному випадку не збігаються – розділяючи екземпляри, SOM робить суб'єктивну класифікацію, що не має того реального фактичного змісту, яким ми наділяємо класи.

Результати класифікації SOM можуть бути наділені фактичним смислом шляхом постановки у відповідність номеру кожного вузла SOM номера того фактичного класу, до якого відноситься більша частина екземплярів навчальної вибірки, віднесених SOM до даного вузла. Для здійснення такої постановки можна запропонувати використовувати простий спосіб, заснований на використанні асоціативного запам'ятовуючого пристрою (АЗП).

Алгоритм навчання системи SOM-АЗП має вигляд:

Крок 1. Реалізується навчальний експеримент і визначаються фактичні класи екземплярів. Виробляється навчання SOM для всіх екземплярів навчальної вибірки.

Крок 2. Для кожного вузла SOM підраховується число екземплярів, що відносяться до кожного з фактичних класів.

Крок 3. Кожному вузлу SOM ставиться у відповідність той фактичний клас, до якого відноситься більша частина екземплярів, віднесених SOM до даного вузла. Постановка відповідності виробляється шляхом запису пари (кортежу) <номер вузла SOM, номер класу> в АЗП. У якості АЗП може бути використаний як блок лінійної або динамічної пам'яті, що обслуговується відповідною процедурою, так і нейромережна асоціативна пам'ять:

а) для системи з двома класами – одношаровий дискретний персептрон;

б) для системи з великим числом класів – багатошарова нейронна мережа або комбінація асоціативної пам'яті на основі НМ Хопфілда з нейромережним селектором максимуму. При цьому на відповідні входи НМ Хопфілда подаються сигнали від кожного з вузлів SOM, а на виході одержують 0, якщо номер вузла SOM не зіставлений даному класу і 1 – якщо зіставлений. Нейромережний селектор максимуму визначає номер вузла НМ Хопфілда (тобто номер фактичного класу), для якого вихід дорівнює 1, для всіх інших вузлів SOM вихід НМ Хопфілда буде дорівнювати 0.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі файли.

```
import numpy as np
import neurolab as nl
import numpy.random as rand
```

Проведіть моделювання вхідних даних. Для цього створіть 4 центри (точки з двома координатами) та випадкове значення, що буде моделювати розкид даних навколо центру за нормальним законом розподілу. Величина розкиду регулюється коефіцієнтом `skv`. Додайте точки та випадкові значення і перемішайте результати.

```
skv = 0.05
centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5]])
rand_norm = skv * rand.randn(100, 4, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 4, 2)
rand.shuffle(inp)
```

Створіть нейронну мережу Кохонена з 2 входами (бо 2 координати точки) та 4 нейронами (бо ми будемо шукати 4 центри і формувати навколо них кластери).

Натренуйте мережу за правилом: алгоритм «Переможець отримує все» (CWTA) на 200 ітерацій (епох) та виводьте помилку кожних 20 епох (щоб можна було б оцінити чи зменшується помилка).

```
# Create net with 2 inputs and 4 neurons
net = nl.net.newc([[0.0, 1.0],[0.0, 1.0]], 4)
# train with rule: Conscience Winner Take All algoritm (CWTA)
error = net.train(inp, epochs=200,  show=20)
```

Виведіть два графіка з результатами.

На першому відобразіть зміну помилки від кількості епох.

На другому вхідні данні за якими здійснювалося тренування, реальні центри кластеру та центри кластеру отримані після тренування.

```
# Plot results:
import pylab as pl
pl.title('Classification Problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('error (default MAE)')
w = net.layers[0].np['w']

pl.subplot(212)
pl.plot(inp[:,0], inp[:,1], '.', \
        centr[:,0], centr[:, 1] , 'yv', \
        w[:,0], w[:,1], 'p')
pl.legend(['train samples', 'real centers', 'train centers'])
pl.show()
```

Занесіть обидва графіка у звіт.

У вікні терміналу також відобразиться інформація.

Занесіть її у звіт. Напишіть, що таке помилка MAE. Проаналізуйте значення, що виведені у вікні термінал та зробіть по них висновок.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_5_task_7.py

Завдання 2.8. Дослідження нейронної мережі на основі карти Кохонена, що самоорганізується

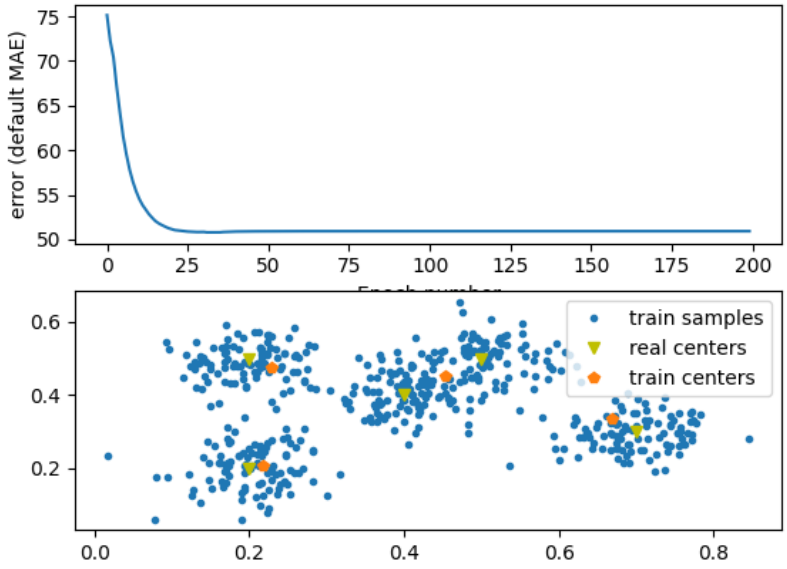
Проведіть дослідження по аналогії з попереднім завданням. Використовуючи готовий код внесіть зміни у вхідні данні згідно вашого варіанту у таблиці 3

Таблиця 3

№ варіанту	Центри кластера	skv
Варіант 1	[0.2, 0.2], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,03
Варіант 2	[0.1, 0.2], [0.4, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,03
Варіант 3	[0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.4, 0.5]	0,03
Варіант 4	[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7]	0,03
Варіант 5	[0.2, 0.1], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,03
Варіант 6	[0.3, 0.3], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,04
Варіант 7	[0.2, 0.1], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.3, 0.5]	0,04
Варіант 8	[0.1, 0.2], [0.4, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.3]	0,04
Варіант 9	[0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.1, 0.5], [0.4, 0.5]	0,04
Варіант 10	[0.2, 0.2], [0.3, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7]	0,04
Варіант 11	[0.2, 0.1], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,05
Варіант 12	[0.3, 0.3], [0.5, 0.4], [0.7, 0.3], [0.2, 0.4], [0.6, 0.5]	0,05
Варіант 13	[0.2, 0.2], [0.3, 0.4], [0.6, 0.3], [0.2, 0.5], [0.5, 0.5]	0,05
Варіант 14	[0.1, 0.2], [0.3, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,05
Варіант 15	[0.2, 0.3], [0.4, 0.4], [0.1, 0.3], [0.2, 0.6], [0.4, 0.5]	0,05
Варіант 16	[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.3, 0.6], [0.5, 0.7]	0,05
Варіант 17	[0.2, 0.1], [0.5, 0.4], [0.5, 0.3], [0.2, 0.5], [0.5, 0.5]	0,06
Варіант 18	[0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,06
Варіант 19	[0.2, 0.1], [0.5, 0.6], [0.5, 0.4], [0.6, 0.5], [0.5, 0.7]	0,06
Варіант 20	[0.1, 0.3], [0.4, 0.4], [0.5, 0.3], [0.2, 0.5], [0.7, 0.5]	0,06
Варіант 21	[0.3, 0.3], [0.4, 0.4], [0.5, 0.3], [0.2, 0.6], [0.4, 0.5]	0,07
Варіант 22	[0.5, 0.2], [0.4, 0.4], [0.3, 0.3], [0.3, 0.6], [0.5, 0.7]	0,07
Варіант 23	[0.1, 0.1], [0.2, 0.4], [0.5, 0.3], [0.2, 0.7], [0.6, 0.5]	0,07
Варіант 24	[0.2, 0.1], [0.3, 0.3], [0.7, 0.3], [0.2, 0.5], [0.6, 0.5]	0,07
Варіант 25	[0.1, 0.1], [0.5, 0.6], [0.2, 0.4], [0.3, 0.5], [0.5, 0.7]	0,07
Варіант 26	[0.1, 0.3], [0.3, 0.4], [0.5, 0.3], [0.4, 0.5], [0.7, 0.5]	0,04
Варіант 27	[0.2, 0.2], [0.3, 0.4], [0.4, 0.3], [0.2, 0.5], [0.5, 0.5]	0,04
Варіант 28	[0.1, 0.2], [0.3, 0.3], [0.4, 0.3], [0.2, 0.5], [0.5, 0.6]	0,04
Варіант 29	[0.2, 0.1], [0.4, 0.4], [0.1, 0.3], [0.2, 0.6], [0.4, 0.5]	0,04
Варіант 30	[0.1, 0.2], [0.4, 0.3], [0.3, 0.3], [0.3, 0.6], [0.5, 0.5]	0,04

Створіть нейронну мережу Кохонена з 2 входами та 4 нейронами

Занесіть обидва графіка результатів у звіт. Щось подібне до рисунку



У вікні терміналу також відобразиться інформація. ***Занесіть її у звіт.***

Створіть нейронну мережу Кохонена з 2 входами та 5 нейронами

Занесіть обидва графіка результатів у звіт.

У вікні терміналу також відобразиться інформація. ***Занесіть її у звіт. Проаналізуйте значення, що виведені у вікнах термінала та на графіках.***

Порівняйте їх між собою та зробіть по них висновок де обов'язково вкажіть: як впливає невірний вибір кількості нейронів числу кластерів на величину помилки?

Порівняйте графіки з попереднього завдання та зробіть висновок як впливає розкид вхідних даних на точність кластеризації.

Висновки занесіть у звіт.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Виконується по аналогії з попереднім завданням.

У коді попереднього завдання необхідно вказати данні свого варіанту та врахувати кількість вхідних центрів.

Збережіть код останньої робочої програми з обов'язковими коментарям під назвою LR_5_task_8.py

Коди комітити на GitHab. У кожному звіті повинно бути посилання на GitHab.

Назвіть бланк звіту СШІ-ЛР-5-NNN-XXXXX.doc
де NNN – позначення групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в СШІ-ЛР-5-NNN-XXXXX.pdf