Звягін микита, КМ-73

Тема: «Розробка програмного забезпечення для реалізації двошарового персептрону з сигмоїдальною функцією активації»

Теоретична частина

Нейронна мережа - це модель яку використовують для класифікації даних у системах із багатьма вимірами. Модель вчиться ставити у відповідність вхідним (у модель) масивам даних правильні вихідні масиви.

Помилка

Помилка нейронної мережі показує наскільки результат роботи моделі відрізняється від значення яке модель мала б повертати. Чим менша помилка моделі, тим правильніший результат моделі на цих вхідних даних.

Навчання

Навчання нейронних мереж відбувається шляхом корекції параметрів які входять у функцію що повертає результат. Ці параметри називаються вагами. Ваги змінюються шляхом мінімізації функції помилки моделі методом градієнтного спуску.

Тренувальні дані

Тренувальні дані (навчальні приклади) для моделей навчання із вчителем виглядають як «вхідні дані» - «правильний результат». Набір із багатьох навчальних прикладів називають датасетом.

Тренування

Під час тренування, усі приклади з датасету проходять через нейронну мережу, для кожного прикладу обчислюється помилка моделі, відбувається коригування вагів. Для отримання потрібної точності, датасет подають моделі не один раз. Кожне використання того ж датасету називають епохою.

Частина 1

Завдання: розробити програмне забезпечення для реалізації класичного нейрону. Передбачити режим навчання на одному навчальному прикладі та режим розпізнавання.

Сама нейронна мережа реалізована у вигляді классу, в якому передбачено:

-Ініціалізацію початкових вагових коефіцієнтів.

-Реалізацію сигмоїдальної функції та її похідної.

-Функцію train(), яка відповідає за навчання нашої нейронної мережі.

-Функцію активації activate().

Найбільш інтересною для нас є функція train():

```
def train(this, tr_in, tr_out, max_iter):
    #тренування мережі, tr_in матриця вхідних даних для тренування
    #tr_out матриця вихідних даних, max_iter якщо тренування буде невадлим для виходу з циклу
    iterations = []
    outputs = []
    errors = []

    output = 0
    error = 0
    i = 0

    print("Start weights:", str(this.net_weights))
    for iteration in range(max_iter):

        #обрахування похибки після проходу вектору через мережу
        output = this.activate(tr_in)
        error = abs((tr_out - output) / tr_out)

        #виведення частини ітерацій
        if (iteration + 1) % 10 == 0:
            iterations.append(iteration+1)
            outputs.append(numpy.round(output[0], 6))
            errors.append(numpy.round(error[0], 6))

        if error < e:
            break

        delta = this.dfsigm(output) * (tr_out - output)
        adjustment = dot(tr_in.T, delta)
        for j in range(len(this.net_weights)):
            this.net_weights[j] += adjustment[j][0]

        i += 1
    save_weights(this.net_weights)
    iterations.append(i + 1)
    outputs.append(numpy.round(output[0], 6))
    errors.append(numpy.round(error[0], 6))

    this.table.add_column(cols[0], iterations)
    this.table.add_column(cols[1], outputs)
    this.table.add_column(cols[2], errors)

    print(this.table)

    print("Weights:", str(this.net_weights))
```

Вона пропускає через функцію активації вхідні тренувальні дані, після чого обчислює помилку, та додає вихідне значення та помилку до масиву для їх збереження, після чого якщо помилка недостатньо мала, починається зворотній хід, та за допомогою методу градієнтного спуску корегуються ваги. Далі дані додаються в таблицю для виведення на екран, та починається наступна ітерація тренування.

Результат роботи першої частини:

```
[mykytazviahin@localhost lab1]$ python lab_1.py
Start weights: [ 0.19418938 -0.44067433  0.07675752 -0.02905309]
+-----------+----------+------------+
| Iteration |    Y     |   Error    |
+-----------+----------+------------+
|     50    | 0.92759  | [0.85518]  |
|    100    | 0.201247 | [0.597507] |
|    150    | 0.229223 | [0.541554] |
|    200    | 0.897403 | [0.794806] |
|    250    | 0.053877 | [0.892245] |
|    300    | 0.069038 | [0.861925] |
|    343    | 0.500453 | [0.000907] |
+-----------+----------+------------+
Weights: [ 0.22419576 -0.2306297   0.19678303  0.12097879]
Recognition regime:
Initial vector: [8 9 8 9]
Recognized value:  0.9153647791022786
[mykytazviahin@localhost lab1]$
```

Зазначу, що до цього вже було зроблено 1000 ітерацій, але для компактності я залишив другу спробу навчання.
Також приклад з використання навчального вектора, як тестового:

```
[mykytazviahin@localhost lab1]$ python lab_1.py
Start weights: [ 0.22419576 -0.2306297   0.19678303  0.12097879]
+-----------+----------+------------+
| Iteration |    Y     |   Error    |
+-----------+----------+------------+
|     1     | 0.500453 | [0.000907] |
+-----------+----------+------------+
Weights: [ 0.22419576 -0.2306297   0.19678303  0.12097879]
Recognition regime:
Initial vector: [1 7 4 5]
Recognized value:  0.5004534889760589
[mykytazviahin@localhost lab1]$
```

Частина 2.

Завдання: розробити програмне забезпечення для реалізації елементарного двошарового персептрону із структурою 1-1-1. Передбачити режим навчання на одному навчальному прикладі та режим розпізнавання.

Структура програми схожа на першу частину, для функції активації також використовується сигмоїдальна функція. На вхід подається вектор довжиною 1.

Після цього в функції train() реалізовано три слоя нейронної мережі. Коли дані проходять через всі слої обчислюється похибка, та частина значень буде виводитись на екран. Вихідні значення проходять через функцію активації, Після цього корегуються ваги.

```
def train(this, tr_in, tr_out, max_iter):
    #тренування мережі, tr_in матриця вхідних даних для тренування
    #tr_out матриця вихідних даних, max_iter якщо тренування буде невадлим для виходу з циклу
    iterations = []
    outputs = []
    errors = []

    y_third_layout = 0
    error = 0
    i = 0
    print("Start weights:", str(this.net_weights))
    for iteration in range(max_iter):
        #прохід даних через другий та третій слой
        x_second_layout = this.net_weights[0] * tr_in
        y_second_layout = this.sigm(x_second_layout)

        x_third_layout = this.net_weights[1] * y_second_layout
        y_third_layout = this.sigm(x_third_layout)

        error = abs((tr_out - y_third_layout) / tr_out)
        #виведення результатів частини ітерацій
        if (iteration + 1) % 50 == 0:
            iterations.append(iteration+1)
            outputs.append(round(y_third_layout[0][0], 6))
            errors.append(round(error[0][0], 6))

        if error <= e:
            print(this.net_weights)
            break
        #зворотній хід, метод розповсюдження помилок
        q_third_layout = this.dfsigm(y_third_layout) * (tr_out - y_third_layout)
        q_second_layout = this.dfsigm(y_second_layout) * (q_third_layout * this.net_weights[1])

        delta_weights_third_layout = q_third_layout * y_second_layout
        delta_weights_second_layout = q_second_layout * tr_in

        this.net_weights[0] += float(delta_weights_second_layout)
        this.net_weights[1] += float(delta_weights_third_layout)
        i += 1

    save_weights(this.net_weights)
    iterations.append(i + 1)
    outputs.append(round(y_third_layout[0][0], 6))
    errors.append(round(error[0][0], 6))
```

Результат роботи програми:

```
[mykytazviahin@localhost lab1]$ python lab_12.py
Start weights: [0.2 0.4]
[ 0.63860194 -1.44307036]
+-----------+----------+----------+
| Iteration |    Y     |  Error   |
+-----------+----------+----------+
|     50    | 0.256763 | 0.283813 |
|    100    | 0.21188  | 0.059401 |
|    150    | 0.20306  | 0.015298 |
|    200    | 0.200826 | 0.004128 |
|    250    | 0.200226 | 0.001128 |
|    300    | 0.200062 | 0.000309 |
|    344    | 0.20002  | 9.9e-05  |
+-----------+----------+----------+
Weights: [ 0.63860194 -1.44307036]
Recognition regime:
Initial vector: [[5]]
Recognized value:  [[0.19241374]]
y = 0.2
[mykytazviahin@localhost lab1]$
```

Як бачимо, розпізнаний образ дорівнює 0.192, що досить близко до 0.2.

Третя частина.

Завдання: розробити програмне забезпечення для реалізації двошарового персептрону із структурою 2-3-1. Передбачити режим навчання «ON-LINE» та режим розпізнавання.

Піддослідна функція x1+x2=y

Частина

Реалізуємо двошаровий персептрон, використаємо функції з попередніх роботи, такі як:
-sigm
-dfsigm
-train()
-activate
-save_result

В функції train() буде реалізовано персептрон структури 2-3-1:

Прохід даних через скриті шари нейронної мережі в наступному коді це q_hidden_layout_1, q_hidden_layout_2, q_hidden_layout_3.

lab_1_3.py

```python
            #Виведення зі скритого слою
            output_hidden = this.sigm(input_hidden)

            #Введення для вихідного слою
            input_op = np.dot(output_hidden, this.out_w)

            #Виведення для вихідного слою
            output_op = this.sigm(input_op)

            #Підрахування похибки та виведення частини даних у консоль
            error = abs((tr_outs - output_op) / tr_outs)

            if (iteration + 1) % 10 == 0:
                iterations.append(iteration+1)
                outputs.append(round(output_op[0], 6))
                errors.append(round(error[0][0], 6))

            if error <= e:
                break

            #зворотнє розповсюдження помилки
            q_out_lay = this.dfsigm(output_op) * (tr_outs - output_op)
            del_out_lay = q_out_lay * output_hidden

            del_out_lay = np.reshape(del_out_lay, (3, 1))
            q_hid_1 = this.dfsigm(
                output_hidden[0][0]) * (q_out_lay * this.out_w[0])
            del_hid_1 = q_hid_1 * tr_inputs

            q_hidden_layout_2 = this.dfsigm(
                output_hidden[0][1]) * (q_out_lay * this.out_w[1])
            del_hid_2 = q_hidden_layout_2 * tr_inputs

            q_hidden_layout_3 = this.dfsigm(
                output_hidden[0][2]) * (q_out_lay * this.out_w[2])
            del_hid_3 = q_hidden_layout_3 * tr_inputs

            delta_hidden_layout = np.array(
                [[del_hid_1[0][0], del_hid_2[0][0], del_hid_3[0][0]],
                 [del_hid_1[0][1], del_hid_2[0][1], del_hid_3[0][1]]])

            for i in range(3):
                this.out_w[i] += del_out_lay[i][0]
            this.hidden_w += delta_hidden_layout

            i += 1
```

Тренування та результат роботи програми:

```
mykytazviahin@localhost:~/univer/neuralNetworks/lab1
 [-0.00279647]
 [-0.00283748]]
Епоха:  17
Похибка:  [[0.02032197]]
Розпізнаний образ: 0.5103717571353796
[-0.07798364  0.0098608   0.10699109]
[[-0.00222219]
 [-0.00237155]
 [-0.00240629]]
Епоха:  18
Похибка:  [[0.01728362]]
Розпізнаний образ: 0.5087937991943694
[-0.08020584  0.00748925  0.1045848 ]
[[-0.00188442]
 [-0.00201098]
 [-0.00204041]]
Епоха:  19
Похибка:  [[0.01469207]]
Розпізнаний образ: 0.5074555734931562
[-0.08209026  0.00547826  0.10254439]
[[-0.00159786]
 [-0.0017051 ]
 [-0.00173003]]
Епоха:  20
Похибка:  [[0.01248375]]
Розпізнаний образ: 0.5063207841112645
[-0.08368812  0.00377316  0.10081436]
[[-0.00135479]
 [-0.00144566]
 [-0.00146678]]
Епоха:  21
Похибка:  [[0.01060353]]
Розпізнаний образ: 0.5053585831604243
[-0.0850429   0.0023275   0.09934757]
[[-0.00114863]
 [-0.00122565]
 [-0.00124354]]
Епоха:  22
Похибка:  [[0.00900374]]
Розпізнаний образ: 0.5045427710249837
Скриті вагові коеф-ти: [[0.39884263 0.59455075 0.79117621]
 [0.29922842 0.29636716 0.09411747]]
Вагові коеф-ти на виході: [-0.08619154  0.00110185  0.09810403]
[mykytazviahin@localhost lab1]$
```

Висновок:

Було реалізовано нейронні мережі наступних видів:

Класичний нейрон структури 4-1.

Бажаний результат: 0.9

Отриманий результат: 0.915

Персептрон структури 1-1-1.

Бажаний результат: 0.2

Отриманий результат: 0.192

Персептрон структури 2-3-1.

Бажаний результат: 0.5

Отриманий результат: 0.505

Результат було отримано за 22 епохи. (тренування на одному тренувальному наборі даних).