Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет ……ІРТЗІ……………………………………………………………………………………

Кафедра …МІРЕС………………………………………………………………………………………

**КУРСОВИЙ ПРОЕКТ (РОБОТА)**

1. **ПОЯСНЮВАЛЬНА ЗАПИСКА**

……………………………………………………..

(позначення документа в разі необхідності)

Програмна реалізація на Python згорткової нейронної мережі

для класифікації зображень рукописних цифр.

…………………………………………………………………………………………………

(тема проекту (роботи)

……………………….……………Цифрова обробка зображень……………………………….

( дисципліна)

Керівник …………………………………ст. викл. каф. МІРЕС Савченко І.В.………..

(підпис, дата, посада, прізвище, ініціали)

Студент групи ЕСТМу-18-1

Деркач А. Ю.

Харків 2020

Харківський національний університет радіоелектроніки

Факультет \_Інформаційних радіотехнологій та технічного захисту інформації\_\_\_

Кафедра \_Медіаінженерії та інформаційних радіоелектронних систем\_\_\_\_\_\_\_\_\_\_

Дисципліна \_Цифрова обробка зображень\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Спеціальність\_171 Електроніка, ОПП «Системи, технології та комп’ютерні\_\_\_\_\_

засоби мультимедіа»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Курс\_\_\_\_\_\_\_\_ група\_\_\_\_\_\_\_\_\_ семестр\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ЗАВДАННЯ**

на курсову роботу

студентові \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи \_\_\_\_\_Програмна реалізація на Python згорткової нейронної мережі\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_для класифікації зображень рукописних цифр\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Термін здачі студентом закінченої роботи \_\_27.05.2020\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. Вихідні дані до курсової роботи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_необхідно класифікувати зображень рукописних цифр з колекції MNIST,\_\_\_\_\_\_\_\_\_

\_згорткова нейронна мережа має містити convolutional layers, full-connected layers,\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_ MaxPooling layers, SoftMax перетворення на виході,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_ метод навчання нейронної мережі – стохастичний метод градієнтного пошуку\_\_\_\_\_\_

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

\_Вступ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_Огляд структур та алгоритмів роботи нейронних мереж для класифікації зображень\_\_

\_Вибір структури нейронної мережі та алгоритму навчання нейронної мережі\_\_\_\_\_\_\_\_

\_Розробка, навчання нейронної мережі на Python та аналіз її ефективності\_\_\_\_\_\_\_\_\_\_\_

\_Висновки\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_Перелік посилань\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Перелік графічного матеріалу (з точним визначенням обов’язкових креслень): \_\_\_\_\_\_\_\_\_\_

\_структура нейронної мережі з зазначенням кількості параметрів кожного шару (А4)\_\_\_

\_графік зміни точності класифікації зображень від кількості епох навчання мережі\_(А4)

1. Дата видачі завдання:\_\_\_22.04.2020\_\_\_\_

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Примітка |
| 1 | Огляд літератури з структур та алгоритмів роботи нейронних мереж для класифікації зображень | 23.04.2020 |  |
| 2 | Розробка структури нейронної мережі та вибір алгоритму навчання нейронної мережі | 25.04.2020 |  |
| 3 | Розробка програмного коду нейронної мережі на Python | 1.05.2020 |  |
| 4 | Аналіз ефективності нейронної мережі для задачі класифікації зображень та оптимізація її структури | 10.05.2020 |  |
| 5 | Оформлення пояснювальної записки та графічних матеріалів | 20.05.2020 |  |
| 6 | Захист курсової роботи | 27.05.2020 |  |
|  |  |  |  |

Студент \_\_\_\_\_\_\_\_\_Деркач А.Ю.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Савченко І.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище, ім’я, по батькові)

«\_\_\_\_\_»\_\_\_\_\_2020\_\_р.

**Вступ**

**Актуальність роботи**

Розпізнавання візуальних образів є один з найважливіших

компонентів систем управління та обробки інформації, автоматизованих

систем і систем прийняття рішень. Завдання, пов'язані з класифікацією і

**іде**нтифікацією предметів, явищ і сигналів, що характеризуються кінцевим

набором деяких властивостей і ознак, виникають в таких галузях як

робототехніка, інформаційний пошук, моніторинг та аналіз візуальних даних,

дослідження штучного інтелекту. Алгоритмічна обробка і

класифікація зображень застосовуються в системах безпеки, контролю і

управління доступом, в системах відеоспостереження, системах віртуальної

реальності та інформаційних пошукових системах. На даний момент в

виробництві широко використовуються системи розпізнавання рукописного тексту, автомобільних номерів, відбитків пальців або людських осіб, що знаходять

застосування в інтерфейсах програмних продуктів, системах безпеки та

ідентифікації особистості, а також в інших прикладних цілях.

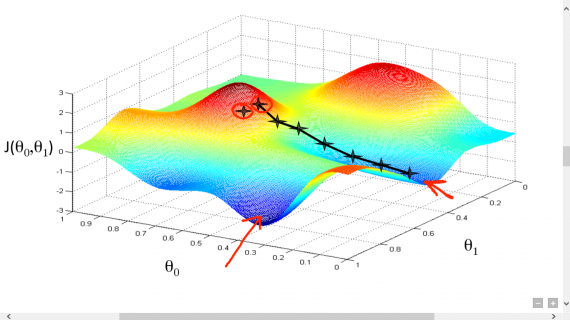
**Вибір структури нейронної мережі та алгоритму навчання нейронної мережі**

Метод найшвидшого спуск - найбільш використовуваний алгоритм навчання, він застосовується майже в кожної моделі машинного навчання. Метод найшвидшого спуск - це, по суті, і є те, як навчаються моделі. Без ГС машинне навчання не було б там, де зараз. Метод градієнтного спуску з деякою модифікацією широко використовується для навчання персептрона і глибоких нейронних мереж, і відомий як метод зворотного поширення помилки.

Метод найшвидшого спуск - метод знаходження мінімального значення функції втрат (існує безліч видів цієї функції). Мінімізація будь-якої функції означає пошук найглибшої западини в цій функції. Майте на увазі, що функція використовується, щоб контролювати помилку в прогнозах моделі машинного навчання. Пошук мінімуму означає отримання найменшій можливій помилки або підвищення точності моделі. Ми збільшуємо точність, перебираючи набір навчальних даних при налаштуванні параметрів нашої моделі (ваг і зміщень).

Отже, градієнтний спуск потрібен для мінімізації функції втрат.

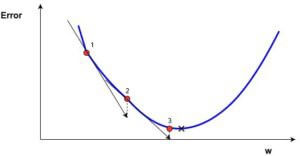
Суть алгоритму - процес отримання найменшого значення помилки. Аналогічно це можна розглядати як спуск в западину в спробі знайти золото на дні ущелини (найнижче значення помилки).



Пошук мінімуму функції

Надалі, щоб знайти найнижчу помилку (найглибшу западину) в функції втрат (по відношенню до одної ваги), потрібно налаштувати параметри моделі. Як ми їх налаштовуємо? У цьому допоможе математичний аналіз. Завдяки аналізу ми знаємо, що нахил графіка функції - похідна від функції по змінній. Це нахил завжди вказує на найближчу западину!

На малюнку ми бачимо графік функції втрат (названий «Помилка» з символом «J») з одним вагою. Тепер, якщо ми обчислимо нахил (позначимо це dJ / dw) функції втрат щодо одного ваги, то отримаємо напрям, в якому потрібно рухатися, щоб досягти локальних мінімумів. Давайте поки представимо, що наша модель має тільки один вага.



функція втрат

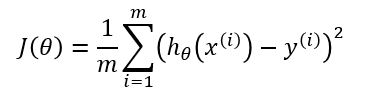
Функція втрат призначена для відстеження помилки з кожним прикладом навчанням, в той час як похідна функції відносного одного ваги - це те, куди потрібно змістити вагу, щоб мінімізувати її для цього прикладу навчання. Ви можете створювати моделі навіть без застосування функції втрат. Але вам доведеться використовувати похідну щодо кожного ваги (dJ / dw).

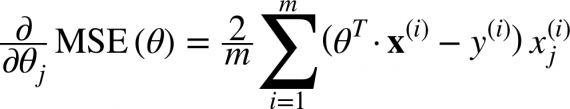
Тепер, коли ми визначили напрямок, в якому потрібно підштовхнути вага, нам потрібно зрозуміти, як це зробити. Тут ми використовуємо коефіцієнт швидкості навчання, його називають гіпер-параметром. Гіпер-параметр - значення, необхідне вашою моделлю, про який ми дійсно маємо дуже туманне уявлення. Зазвичай ці значення можуть бути вивчені методом проб і помилок. Тут не так: одне підходить для всіх гіпер-параметрів. Коефіцієнт швидкості навчання можна розглядати як «крок у правильному напрямку», де напрямок походить від dJ / dw.

Це була функція втрат побудована на один вага. У реальної моделі ми робимо все перераховане вище для всіх ваг, перебираючи все приклади навчання. Навіть у відносно невеликій моделі машинного навчання у вас буде більше ніж 1 або 2 ваги. Це ускладнює візуалізацію, оскільки графік буде мати розмірами, які розум не може собі уявити.

Детальніше про градієнти

Крім функції втрат градієнтний спуск також вимагає градієнт, який є dJ / dw (похідна функції втрат щодо одного ваги, виконана для всіх ваг). dJ / dw залежить від вашого вибору функції втрат. Найбільш поширена функція втрат середньоквадратичної помилки.

Похідна цієї функції щодо будь-якої ваги (ця формула показує обчислення градієнта для лінійної регресії):

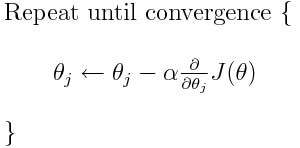


Це - вся математика в ГС. Дивлячись на це можна сказати, що по суті, ГС не містить багато математики. Єдина математика, яку він містить в собі - множення і ділення, до яких ми доберемося. Це означає, що ваш вибір функції вплине на обчислення градієнта кожного ваги.

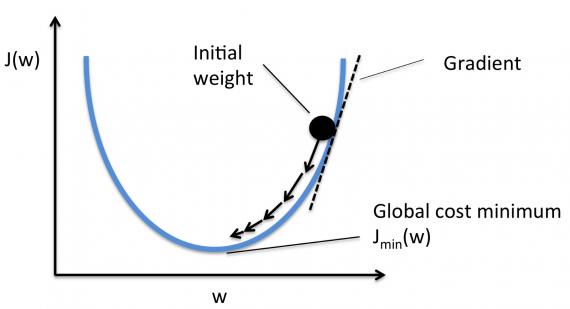
Коефіцієнт швидкості навчання

Все, про що написано вище, є в підручнику. Ви можете відкрити будь-яку книгу про градієнтному спуску, там буде написано те ж саме. Формули градієнтів для кожної функції втрат можна знайти в інтернеті, не знаючи як вивести їх самостійно.

Однак проблема у більшості моделей виникає з коефіцієнтом швидкості навчання. Давайте поглянемо на оновлене вираз для кожного ваги (j лежить в діапазоні від 0 до кількості ваг, а Theta-j це j-й вага в векторі ваг, k лежить в діапазоні від 0 до кількості зсувів, де Bk - це k-е зміщення в векторі зсувів). Тут alpha - коефіцієнт швидкості навчання. З цього можна сказати, що ми обчислюємо dJ / dTheta-j (градієнт ваги Theta-j), і потім крок розміру alpha в цьому напрямку. Отже, ми спускаємося по градієнту. Щоб оновити зміщення, замініть Theta-j на B-k.



Якщо цей розміру кроку alpha занадто великий, ми подолаємо мінімум: тобто промахнемося повз мінімуму. Якщо alpha занадто мала, ми використовуємо занадто багато ітерацій, щоб дістатися до мінімуму. Отже, alpha повинна бути відповідною.



Використання градієнтного спуску

Примітка: Ваги тут представлені в векторах. У більших моделях вони, напевно, будуть матрицями.

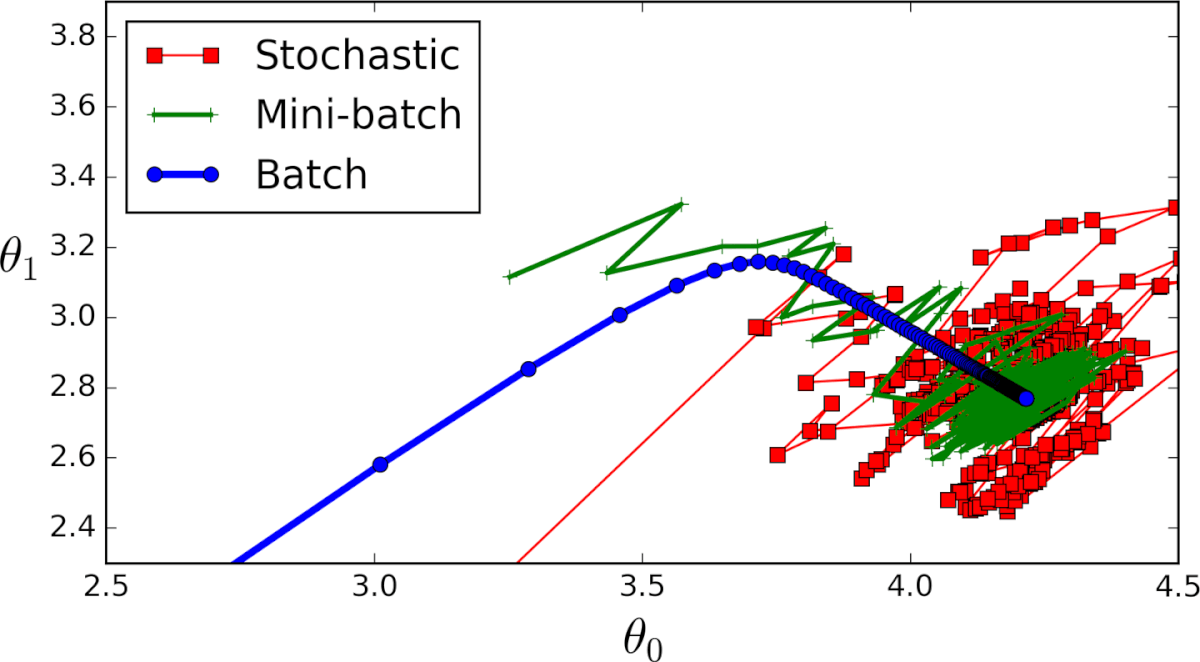
Різні типи градієнтного спуску

Існує 3 варіанти градієнтного спуску:

1. Мini-batch: тут замість перебирання всіх прикладів навчання і з кожної итерацией, яка виконує обчислення тільки на одному прикладі навчання, ми обробляємо n навчальних прикладів відразу. Цей вибір хороший для дуже великих наборів даних.

2. Стохастичний градієнтний спуск

Серія ГС

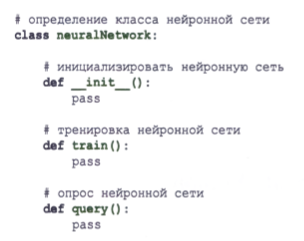


Зображення, що порівнює 3 попадання в локальні мінімуми

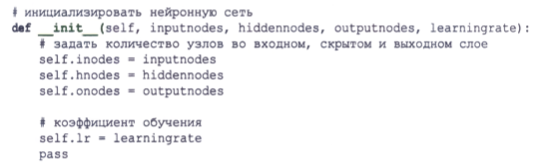
**Розробка, навчання нейронної мережі на Python та аналіз її ефективності**

**Скелет коду**

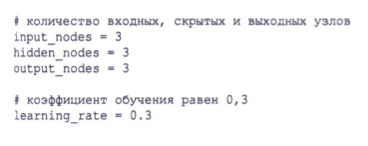
Клас нейронної мережі повинен містити принаймні три функції:

* Ініціалізація - завдання кількості вхідних, прихованих і вихідних вузлів;
* Тренування - уточнення вагових коефіцієнтів в процесі опрацювання наданих для навчання мережі тренувальних прикладів;
* Опитування - отримання значень сигналів з вихідних вузлів після надання значень вхідних сигналів.   
     
  Мій початковий код має такий вигляд:  
      
     
  **Ініціалізація мережі**

Почнемо з ініціалізації. Мені необхідно задати кількість вузлів вхідного, прихованого і вихідного шарів. Ці дані визначають конфігурацію і розмір нейронної мережі. Замість того щоб жорстко ставити їх в коді, я передбачила установку відповідних значень у вигляді параметрів під час створення об'єкта нейронної мережі. Завдяки цьому можна буде створювати нові нейронні мережі різного розміру. Далі потрібно встановити коефіцієнт навчання. Так виглядає функція ініціалізації \_\_init \_\_ () в подібному випадку:



Додаю цей код в визначення класу нейронної мережі і створюю об'єкт невеликої мережі з трьома вузлами в кожному шарі і коефіцієнтом навчання, рівним 0,3.

Даний код дозволяє отримати об'єкт мережі, але такий об'єкт поки що не особливо корисний, тому що не містить жодної функції, здатної виконувати корисну роботу. Далі я створюю мережу, що складається з вузлів і зв'язків. Найбільш важлива частина цієї мережі - **вагові коефіцієнти зв'язків** (ваги). Вони використовуються для розрахунку поширення сигналів в прямому напрямку, а також зворотного поширення помилок.

Тепер я створюю такі матриці:

• матрицю ваг для зв'язків між вхідним і прихованим шарами, Wвходной\_скритий> розмірністю hidden\_nodes х input\_nodes;

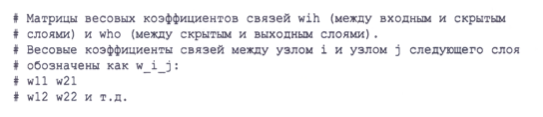
• іншу матрицю для зв'язків між прихованим і вихідним сло ями, Wскритий\_виходной, розмірністю output\_nodes х hidden\_nodes. Початкові значення вагових коефі цієнт повинні бути невеликими і вибиратися випадковим чином. Наступна функція з пакета numpy генерує масив випадок них чисел в діапазоні від 0 до 1, де розмірність масиву дорівнює rows х х columns:



Я використовую розширення пакета numpy і імпортую цю бібліотеку в самому початку коду.

Даний підхід можна поліпшити, оскільки вагові коефіцієнти можуть мати не тільки позитивні, але і негативні значення і змінюватися в межах від -1,0 до +1,0. Для простоти віднімемо 0,5 з цих граничних значень, перейшовши до діапазону значень від -0,5 до +0,5. деякі вагові коефіцієнти тепер мають негативні значення. Далі створюю матрицю початкових ваг в програмі на Python. Ці вагові коефіцієнти складають невід'ємну частину нейронної мережі і служать її характеристиками а не тимчасовим набором даних, які зникають відразу ж після того, як функція відпрацювала. Це означає, що вони будуть частиною процесу ініціалізації і будуть доступними для інших методів, таких як функції тренування і опитування мережі.

Нижче наведено код, включаючи коментарі, який створює дві матриці вагових коефіцієнтів, використовуючи значення змінних **self .inodes,** **self .hnodes** і **self .onodes** для завдання відповідних розмірів кожної з них.





Так я створила матриці зв'язків між її вузлами.

**Опитування мережі**

Функція **q u e r y ()** приймає в якості аргументу вхідні дані нейронної мережі і повертає її вихідні дані.

Отримуємо вхідні сигнали для вузлів прихованого шару шляхом поєднання матриці вагових коефіцієнтів зв'язків між вхідним і прихованим шарами з матрицею вхідних сигналів:



Далі застосовую функцію скалярного твори бібліотеки numpy до матриць ваг та вхідних сигналів:

hidden\_inputs = numpy.dot(self.wih, inputs)

Цей рядок коду виконує всю роботу по об'єднанню всіх вхідних сигналів з відповідними вагами для отримання матриці згладжених комбінованих сигналів в кожному вузлі прихованого шару.

Для отримання вихідних сигналів прихованого шару просто застосовуємо до кожного з них сигмоид:

**Оскритий** = сигмоїда (**Xскритий**)

Бібліотека scipy в Python містить набір спеціальних функцій, в тому числі сигмоид, яка називається expit ().



Нижче наведено код, що визначає функцію активації, який використовується в розділі ініціалізації нейронної мережі.



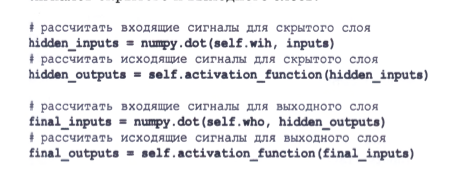
Створила функцію lambda, яке дозволяє створювати функції швидким і зручним способом. В даному випадку функція приймає аргумент х і повертає scipy.special.expit (), а це сигмоїда. даної функції я привласнила ім'я self .activation\_function (). Це означає, що всякий раз, коли потрібно використовувати функцію активації, її потрібно буде викликати як self.activation\_function ().

Далі застосовую функцію активації до згладженим комбінованим входять сигналів, що надходять на приховані вузли.



Таким чином, сигнали, що йдуть від прихованого шару, описуються матрицею hidden\_outputs.

Фрагмент коду, який об'єднує розрахунки сигналів прихованого і вихідного шарів:



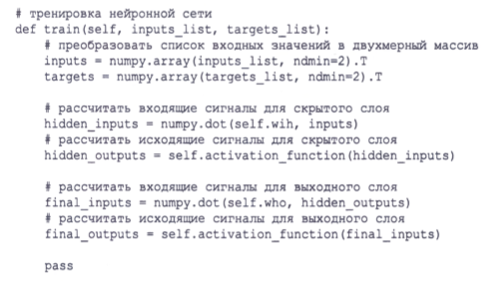
далі імпортую модулі **numpy** та **scipy.special.**

**import numpy**

**# библиотека scipy.special с сигмоидой expit ()**

**import scipy.special**

**Тренування мережі**

* Перша частина - розрахунок вихідних сигналів для заданого тренувального прикладу.
* Друга частина - порівняння розрахованих вихідних сигналів з бажаним відповіддю і оновлення вагових коефіцієнтів зв'язків між вузлами на основі знайдених відмінностей. 

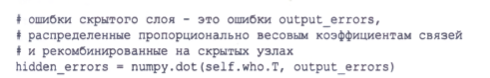
Список targets\_list перетворюється в масив точно так, як список input\_list:



уточнення ваг на основі розбіжності між розрахунковими і цільовими значеннями. Спочатку потрібно обчислити помилку, яка є різницею між бажаним цільовим вихідним значенням, наданий ним тренувальним прикладом, і фактичним вихідним значенням. Вона являє собою різницю між матрицями (targets - final\_ outputs), що розраховується поелементно:



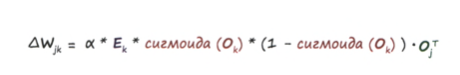
Далі потрібно розрахувати зворотне поширення помилок для вузлів прихованого шару.

Для ваг зв'язків між прихованим і вихідним шарами використовую змінну output\_errors.

Для ваг зв'язків між вхідним і прихованим шарами ми використовуємо тільки що розраховану змінну hidden errors.

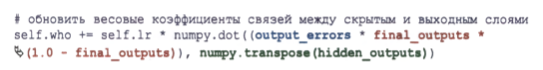
Вираз для поновлення ваги зв'язку між вузлом j і вузлом k наступного шару в матричної формі:

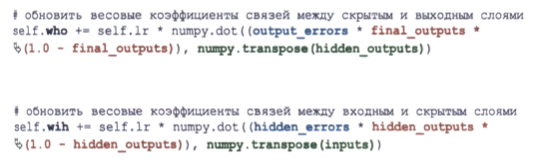


Величина а - це коефіцієнт навчання, сигмоїда - це функція активації.

Останній член виразу - перетворення стовпчика вихідних сигналів в рядок.

Код для поновлення ваг зв'язків між прихованим і вихідним шарами:



Код для двох наборів вагових коефіцієнтів: 

**Набір рукописних цифр MNIST**

**all\_values = data\_list[**0**].split(',')**

image\_array = numpy.asfarray(all\_yalues[l:]).reshape((28,28)) matplotlib.pyplot.imshow(image\_array, cmap='Greys', ^interpolation»'None')

Запис списку у вигляді all\_values [1:] беремо всі елементи списку за винятком першого, ігноруємо перше значення, що грає роль маркера і беремо лише решту 784 елемента.

**Подготовка тренировочных данных MNIST**

Спочатку потрібно перевести значення колірних кодів з більшого діапазону значень 0-255 в набагато менший, що охоплює значення від 0,01 до 1,0.

Розподіл вихідних вхідних значень, що змінюються в діапазоні 0-255, на 255 приведе їх до діапазону 0-1,0. Подальше множення цих значень на коефіцієнт 0,99 приведе їх до діапазону 0,0-0,99. Далі інкрементуємо їх на 0,01, щоб вмістити їх в бажаний діапазон 0,01-1,0.

scaled\_input = (numpy.asfarray(all\_values**[1:]) / 255.0 \* 0.99)** + **0.01** print(scaled\_input)

будемо використовувати значення 0,01 і 0,99, і тому цільовим масивом для маркера "5" повинен бути масив [0.01,0.01,0.01,0.01,0.01,0.99,0.01,0.01,0.01,0.01].

А ось як виглядає код на мові Python, що створює цільову матрицю:

# кількість вихідних вузлів - 10

onodes = 10

targets = numpy.zeros(onodes) + 0.01

Підводячи підсумки, На самому початку цього коду імпортуємо графічну бібліотеку, додаємо код для завдання розмірів вхідного, прихованого і вихідного шарів, зчитуємо малий тренувальний набір даних MNIST, а потім тренуємо нейронну мережу з використанням цих записів.

Я використовую 784 вхідних вузла бо це число дорівнює добутку 28x28, який представляє кількість пікселів, з яких складається зображення рукописного цифри. Тому, вибираючи кількість вузлів меншим, ніж кількість вхідних значень, ми змушуємо мережу намагатися знаходити ключові особливості шляхом узагальнення інформації. У той же час, якщо вибрати кількість прихованих вузлів занадто малим, будуть обмежені можливості мережі щодо визначення достатньої кількості характерних ознак або шаблонів в зображенні. Тим самим ми позбавили б мережу можливості виносити власні судження щодо даних MNIST. Вихідний шар повинен забезпечувати висновок 10 маркерів, а значить, повинен мати десять вузлів, вибір проміжного значення 100 для кількості вузлів прихованого шару.

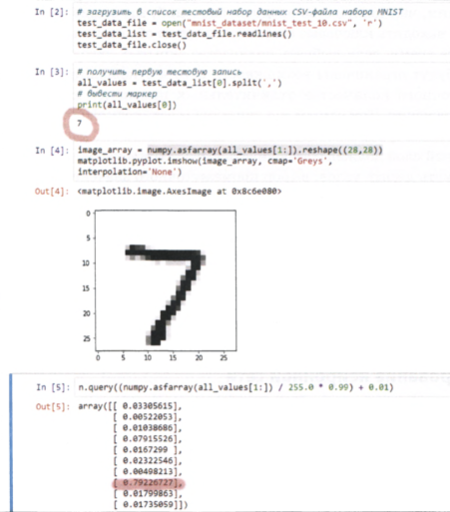
**Тестування нейронної мережі**

Впоравшись з тренуванням мережі, потрібно перевірити, як вона працює використовуючи тестовий набір даних.

# загрузить в список тестовый набор данных CSV-файла набора MNIST

test\_data\_file = open("mnist\_dataset/mnist\_test\_10.csv", 'г')

test\_data\_list = test\_data\_file.readlines()



Результати опитування вже навченої нейронної мережі, виконаного з використанням першого запису тестового набору даних.

**Висновки**

Я навчила нейронну мережу способом мінімізації помилок мережі методом градієнтного спуску і добилися того, що вона змогла визначити цифру, надану їй у вигляді зображення. До цього мережа не стикалася з даними зображенням, оскільки воно не входило в тренувальний набір даних. Отже, нейронна мережа виявилася в стані коректно класифікувати незнайомий їй цифровий символ.

За допомогою кількох рядків коду на мові Python створила нейронну мережу, здатну робити те, що багато людей вважали б проявом штучного інтелекту, - розпізнавати зображення цифр, написаних рукою людини.

Для навчання мережі була використана мала частина повного набору тренувальних даних. цей набір включає 60 тисяч записів, а я використовувала тільки 100 з них.

Для розпізнавання рукописних цифр була виконана розробка матричного

В якості маркера першого запису тестового набору мережу визначила символ "7". Саме цієї відповіді ми очікували, опитуючи її.

**Перелік посилань**

* habr.com
* “Створюємо нейронну мережу”, Тарик Рашид
* “Глибоке навчання з точки зору практика”, Паттерсон, Гібсон.
* “Грокаем глибоке навчання”, Ендрю Траск
* “Прикладне глибоке навчання. Підхід до розуміння глибоких нейронних мереж на основі методу кейсів”, Умберто Мікелуччи