

## Лабораторна робота №5

**Тема:** Проектування та навчання штучної нейронної мережі для задач класифікації

**Математична модель.** Штучним лінійним нейроном із  $n$  входами називається функція  $y: R^n \rightarrow R$ , яка відображає вхідний вектор  $x = (x_1, x_2, \dots, x_n) \in R^n$  в число  $y(x)$  за правилом:

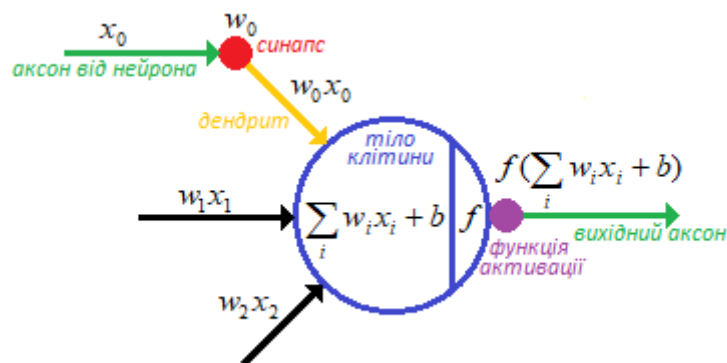
$$y(x) = f(x_1 w_1 + x_2 w_2 + \dots + x_n w_n + b) = f\left(\sum_{i=1}^n x_i w_i + b\right)$$

У векторній формі ця функція може бути записана так:

$$y(x) = f(Wx + b),$$

де  $Wx$  скалярний добуток векторів  $W$  і  $x$ .

Схематично, за аналогією із біологічним нейроном, штучний нейрон виглядає так:



Введемо спеціальну термінологію, яка використовується в штучних нейронних мережах:

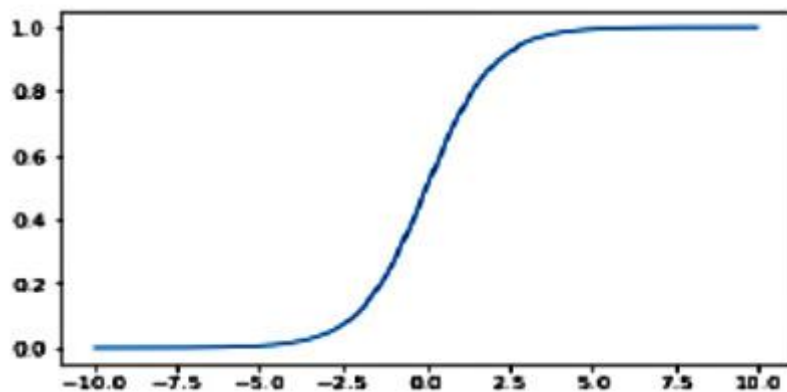
- Векторний простір  $R^n$  називається простором ознак
- Змінні  $x_i$  називають входами нейрона
- Параметри  $w_i$  називаються вагами нейрона
- Параметр  $b$  називається зсувом
- Значення функції  $y(x)$  називається виходом, відгуком нейрона
- Функція  $f: R \rightarrow R$  називається функцією активації (збудження) нейрона

- Гіперплощина  $x_1 w_1 + x_2 w_2 + \dots + x_n w_n + b = 0$  в просторі ознак  $R^n$  називається розділюючою гіперплощиною нейрона.

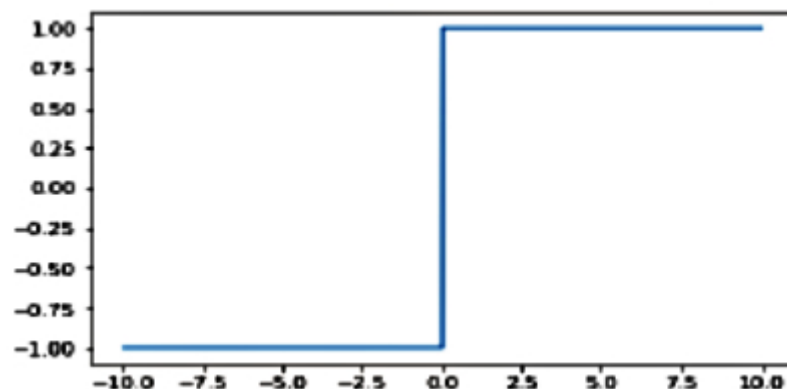
Як правило, функцію активації  $f$  підбирають так, щоб множина значень була інтервалом  $[0, 1]$ , або  $[-1, 1]$ ,  $[0, \infty]$ .

Найчастіше використовуються такі функції:

- Сигмоїдна функція активації  $\sigma(x) = \frac{1}{1 + e^{-x}}$ ,



- ReLU (rectified linear unit): Функція  $f(x) = \max(0, x)$
- Функція  $f(x) = \text{sign}(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0. \end{cases}$

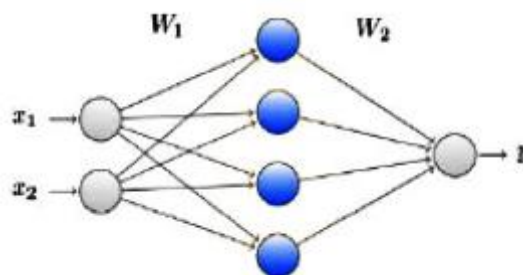


Основна обчислювальна властивість нейрона полягає в тому, що він визначає деяку гіперплощину в просторі ознак, яка розділяє цей простір на дві частини. Тобто, для всіх точок з одного боку гіперплощини нейрон, як функція, приймає одне значення, а для інших точок інше. Наприклад для функції  $\text{sign}(x)$  цими значеннями будуть -1 і 0. Ця обставина дозволяє

використати штучний нейрон для простих задач *бінарної класифікації*, коли дві множини ознак можна відокремити розділяючою гіперплощиною. Визначення коефіцієнтів цієї розділяючої гіперплощини називається *навчанням нейрона*. Штучний нейрон називається *навченим* для деякої бінарної задачі класифікації, якщо його розділяюча гіперплощина відокремлює два класи у просторі ознак даної задачі. Зрозуміло, що для даної задачі може існувати багато розділяючих гіперплощин, тому навчити нейрон можна, як правило, неоднозначно.

Проте, для багатьох реальних задач, одного нейрона недостатньо для проведення класифікації, тому потрібно задіяти більшу кількість нейронів, які об'єднують в шари, а шари в *нейронній мережі* різної *архітектури*.

Ось приклад нейронної мережі із *вхідним шаром* із двох нейронів, *прихованим шаром* із чотирьох нейронів і *вихідним шаром* із одного нейрона.



Вагові коефіцієнти чотирьох нейронів прихованого шару утворюють  $4 \times 2$  матрицю  $W_1$ :

$$W_1 = \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \\ w_{4,1} & w_{4,2} \end{pmatrix},$$

та вектор зсувів  $b_1$ :

$$b_1 = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

Як функція, на вході цей шар отримує двовимірний вектор-стовпчик  $x^T = (x_1, x_2) \in R^2$ , а на виході отримуємо чотиривимірний вектор стовпчик, який в матричному вигляді записується так:

$$f_1(W_1x + b_1) \in R^4$$

Тут  $f_1$  функція активації, яку ми будемо вважати однаковою для кожного нейрону цього шару, і яка діє по-елементно на кожен компоненту вектору. Далі отриманий стовпчик передається останньому шару з  $4 \times 1$  матрицею  $W_2$ , функцією активації  $f_2$  і вектором зсувів  $b_2$ , у нашому випадку, це не просто число, оскільки в останньому шарі тільки один нейрон. В результаті, числова функція  $y: R^4 \rightarrow R$ , яку реалізує ця нейронна мережа має вигляд:

$$y(x) = f_2(W_2 f_1(W_1x + b_1) + b_2),$$

і залежить від 17 параметрів.

Нейронна мережа з такою архітектурою називається нейронною мережею *прямого поширення* (Feedforward neural network). Багатoshарова нейронна мережа прямого поширення (Multilayer neural network) із  $n$  шарами, є послідовною композицією функцій шарів і реалізовує таку функцію:

$$y(x) = f_n(W_n(\dots f_2(W_2 f_1(W_1x + b_1) + b_2) \dots) + b_n),$$

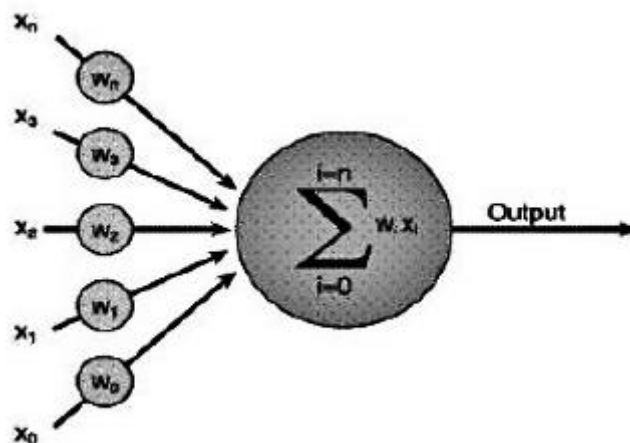
де  $W_i$ ,  $b_i$ ,  $f_i$  відповідні матриці вагових коефіцієнтів, вектори зсуву та функції активації для кожного шару.

Багатoshарові нейронні мережі дозволяють вирішувати задачі класифікації, особливо вони є ефективними для розпізнавання образів (Pattern recognition).

Візуалізація: <https://www.youtube.com/watch?v=IHZwWFHWa-w>

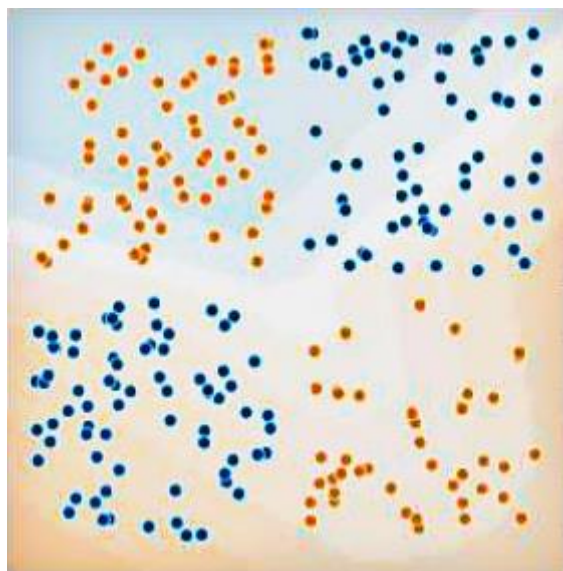
### **Навчання одного нейрона**

Алгоритм навчання нейрону відноситься до класичних алгоритмів *навчання з вчителем*. Розглянемо найпростішу мережу із трьома шарами:



Оскільки довільна мережа, локально побудована саме з таких частин, то для розуміння принципів навчання мереж важливо зрозуміти механізм навчання такої простої мережі.

Раніше (в курсі Обробка та аналіз зображень) ми вже розглядали детально процес навчання одного нейрона на двох лінійно роздільних класах. На практиці такі задачі зустрічаються рідко. На малюнку нижче зображено приклад двох лінійно нероздільних класів:



Хоча один нейрон не може розділити такі два класи, але ми можемо спробувати провести дискримінантну площину так, щоб мінімізувати помилку такого розділення.

Розглянемо *метод Відроу-Хоффа*, або *дельта-правило найменшого середнього квадрату* навчання одного нейрону, для задачі класифікації

двох типів класів  $w_1$  і  $w_2$ . На кожному кроці навчання мінімізується помилка між фактичною та бажаною реакцією нейрона. Розглянемо нейрон:

$$y(x) = f(Wx + b), x \in R^n$$

Для зручності розширимо ваговий вектор і вектор ознак поклавши:

$$W = (w_1, w_2, \dots, w_n, 1), x = (x_1, x_2, \dots, x_n, b)$$

Тоді нейрон перепишеться компактніше:

$$y(x) = f(Wx)$$

Розглянемо цільову функцію, або *функцію помилок* (loss function):

$$E(W) = \frac{1}{2}(r - Wx)^2,$$

тут  $r$  – бажана реакція нейрона, тобто  $r = 1$ , якщо вектор ознак належить класу  $w_1$  і  $r = 2$ , якщо вектор ознак належить до другого класу  $w_2$ .

Нам потрібно знайти мінімум цієї функції, як функції невідомих ваг. Це класична задача оптимізації для наближеного розв’язання якої використовують ітеративний алгоритм *градієнтного спуску* (gradient descent). Ідея алгоритму базується на простому факті – для диференційованої функції від  $n$ -змінних  $F(x)$  її градієнт:

$$\nabla F(x) = \left( \frac{\partial F(x)}{\partial x_1}, \frac{\partial F(x)}{\partial x_2}, \dots, \frac{\partial F(x)}{\partial x_n} \right),$$

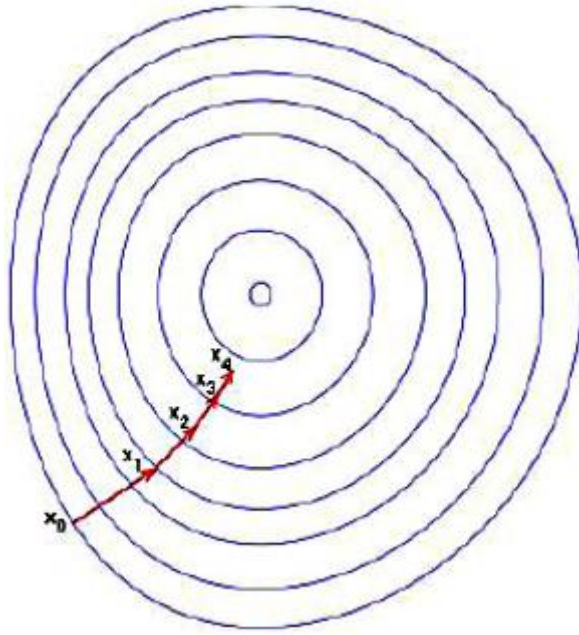
обчислений в деякій точці  $x_0$ , наближений в бік найбільшого зростання функції в даній точці. З цього випливає, що в точці:

$$x_1 = x - \lambda \nabla F(x),$$

при достатньо малому значенні  $\lambda$  значення функції менше ніж значення функції в точці  $x_1$  тобто  $F(x_0) > F(x_1)$ . Це означає, що віднявши  $\lambda \nabla F(x)$  ми здійснили рух проти градієнта, тобто рух у напрямку мінімуму функції. Тому ми будуємо таку послідовність точок  $x_0, x_1, x_2, \dots$ , що:

$$x_{k+1} = x_k - \lambda_k \nabla F(x_k),$$

і *сподіваємося*, що ці послідовність збігається до точки мінімуму.



Для багатьох *хороших* функцій  $F(x)$  і при правильному виборі  $\lambda_k$  ця послідовність справді збігається до точки мінімуму. Число  $\lambda_k$  називається *швидкістю навчання* (learning rates). На практиці вибирають швидкість навчання з інтервалу  $(0.1, 1)$ .

Повернемося тепер до цільової функції  $E(W)$  простої нейронної мережі. Нехай  $E(0)$  деяке початкове наближення значень вагових коефіцієнтів. Тоді, ми будемо ітеративний процес:

$$W(k+1) = W(k) - \lambda \left. \frac{\partial E(W)}{\partial W} \right|_{W=W(k)}, k = 1, 2, 3, \dots$$

і після деякої кількості ітерацій знаходимо оптимальне значення вагових коефіцієнтів, тобто таке значення, яке мінімізує цільову функцію. Тим самим процес навчання нейрона буде завершеним.

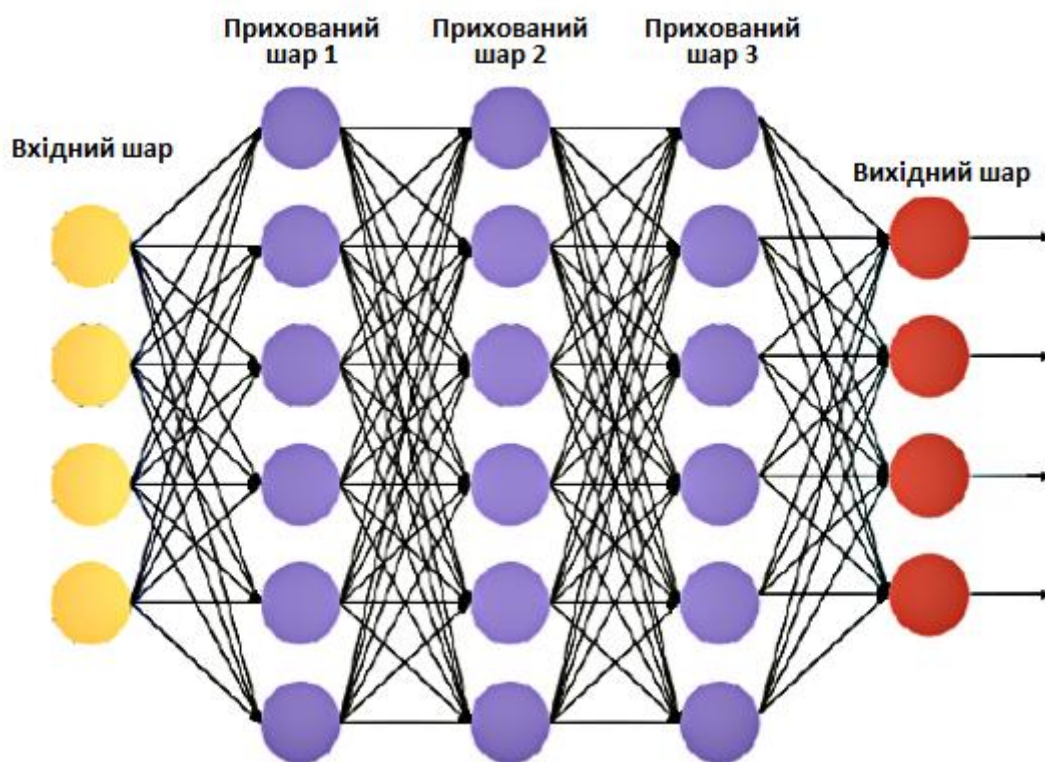
Відмітимо, що замість квадрату різниці для оптимізації можна застосувати інші цільові функції, які ми розглядаємо пізніше. Також в деяких випадках ітеративний процес може відбутися дуже повільно.

Зауважимо, що навіть коли класи  $w_1, w_2$  лінійно відокремлені, то цей алгоритм не обов'язково побудує дискримінантну гіперплощину.



## Навчання мережі методом зворотного поширення помилки (back propagation)

Розглянемо багатошарову нейронну мережу, в якій кожен нейрон одного шару з'єднаний з кожним нейроном наступного шару, і більше ніяких зв'язків між нейронами немає, наприклад нейрони одного шару ніяк не зв'язані. Такі нейронні мережі називаються *повнозв'язні мережі прямого поширення* (Fully Connected Neural Network).



Нейронна мережа із трьома прихованими шарами

Навчання методом зворотного поширення помилки є модифікацією методу градієнтного спуску і схоже на навчання одного нейрона. Навчання починають із нейронів останнього вихідного шару  $L$  за аналогією із навчанням одного нейрону. Цільова функція, за аналогією із одним нейроном може мати такий вигляд:

$$E(W_L) = \frac{1}{2} \sum_{n \in L} (r_n - L_n(x))^2,$$



де  $r_n$  бажане значення нейрона взятє із тестового прикладу, а  $L_n(x)$  – фактичний відгук  $n$ -го нейрона вихідного шару  $x$ . Знайшовши мінімум цільової функції, ми тим самим визначимо вагові коефіцієнти всіх нейронів вихідного шару. Знаючи як реагує вихідний шар, ми можемо за цим самим алгоритмом знайти вагові коефіцієнти нейронів останнього прихованого шару, потім наступного шару і так далі. Рухаючись до початку, тобто поширюючи помилку від шару до шару, ми поступово визначимо ваги всієї нейронної мережі.

Крім штучних багатшарових нейронних мереж прямого поширення існують інші архітектури нейронних мереж, зокрема *рекурентна* та *згорткові*. Сукупність алгоритмів та методів навчання таких мереж називається *глибоким навчанням* (Deep learning) і є частиною ширшої дисципліни – *машинне навчання* (Machine learning).

Скільки потрібно прихованих шарів і скільки нейронів в них має бути?

Зі збільшенням розміру та кількості шарів в нейронній мережі, розділююча здатність мережі зростає, тобто збільшується кількість різних функцій, які визначаються такою мережею. Розглянемо три різні архітектури мереж для вирішення тієї самої задачі класифікації.



На наведеній діаграмі ми бачимо, що нейронні мережі з більшою кількістю нейронів можуть виражати складніші функції. Проте при збільшеній кількості шарів і нейронів може статися ефект *перенавчання*

(Overfitting), коли модель починає враховувати і другорядну інформацію про об'єкти класифікації, або просто запам'ятовує кожен потрібний відгук тестового набору, а на нових даних веде себе погано. Наприклад, модель з 20-ма прихованими нейронами відповідає всім навчальним даним, але ціною сегментації простору на безліч розділених червоних і зелених областей прийняття рішень. Тому може виявитися, що модель з трьома прихованими нейронами може ефективніше для класифікування даних тому, що не враховує випадкові данні (шуми), які не впливають на результат класифікації.

Візуалізація процесу навчання різних мереж можна побачити за посиланням:

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

## **Побудова та навчання нейронної мережі з Python і фреймворком Keras**

Keras – це потужна, проста у виконанні бібліотека Python, що надає високорівневі будівельні блоки для конструювання моделей глибокого навчання. Вона не реалізує низькорівневі операції, такі як маніпуляції з тензорами і диференціювання, для цього використовуються спеціалізовані низькорівневі бібліотеки. До Keras можна підключити декілька різних низькорівневих бібліотек. В даний час підтримуються три такі бібліотеки: TensorFlow, Theano і Microsoft Cognitive Toolkit (CNTK). За замовчуванням ми будемо використовувати бібліотеку TensorFlow як найбільш поширену і високоякісну.

Побудова нейронної мережі складається з наступних етапів:

- Підготовка і завантаження даних
- Визначення архітектури моделі
- Компіляція моделі
- Навчання моделі

- Оцінювання моделі Keras
- Збирання моделі
- Застосування моделі

## Встановлення Keras

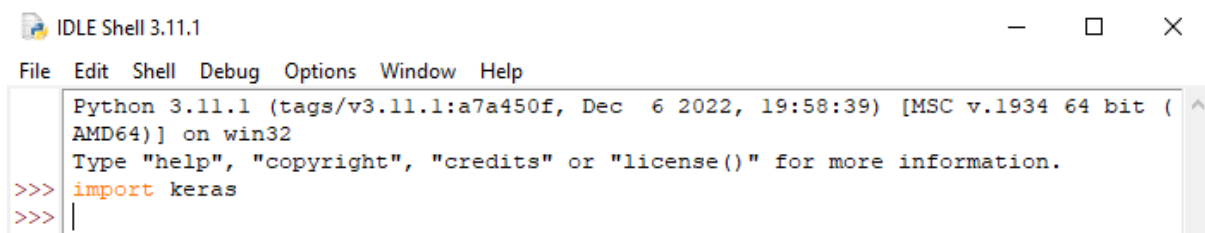
Встановлення пакета TensorFlow (даний пакет може встановлюватися досить довго):

```
pip install tensorflow
```

Далі встановлюємо пакет Keras:

```
pip install keras
```

Перевіряємо в IDLE успішність завершення установки:



```
IDLE Shell 3.11.1
File Edit Shell Debug Options Window Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import keras
>>> |
```

## Приклад виконання завдання

Основи роботи Keras покажемо на конкретному прикладі.

**Завдання** – Спроектувати та навчити нейронну мережу для розпізнавання рукописних чисел.

**Підготовка і завантаження даних.** Нагадаємо, що нейронна мережа визначається конкретними значеннями вагових коефіцієнтів кожного нейрона. Процес навчання нейронної мережі полягає у підготовці цих вагових коефіцієнтів на основі тестових даних так, щоб кожен образ на вході правильно класифікувався на виході. Ми будемо використовувати готовий набір даних чорно-білих зображень рукописних цифр із відомого набору даних MNIST, яка знаходиться на сайті <http://yann.lecun.com/exdb/mnist/> . Цей набір містить 60 000 навчальних зображень (28×28 пікселів) і 10 000 контрольних зображень такого вигляду:



Цей набір даних уже входить в Keras у вигляді чотирьох масивів і його можна завантажити наступним чином:

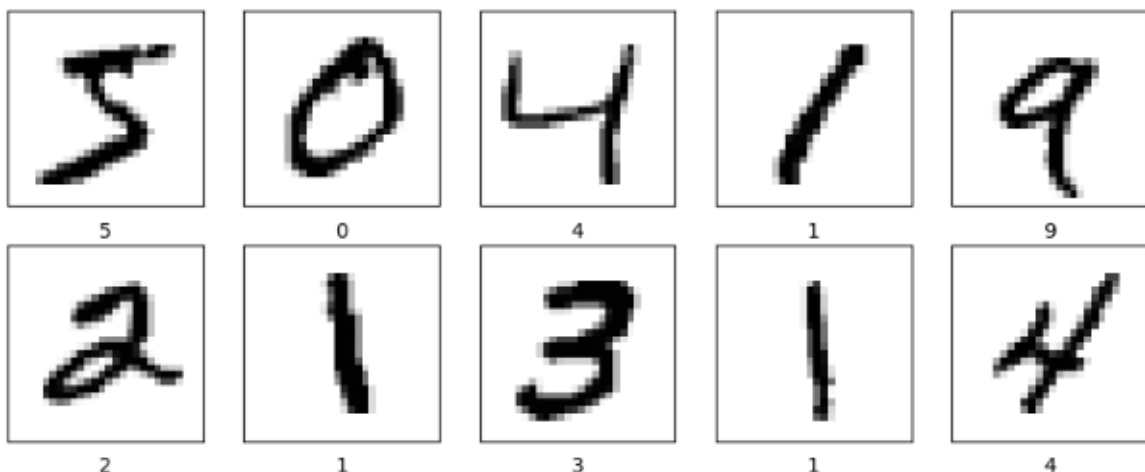
```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

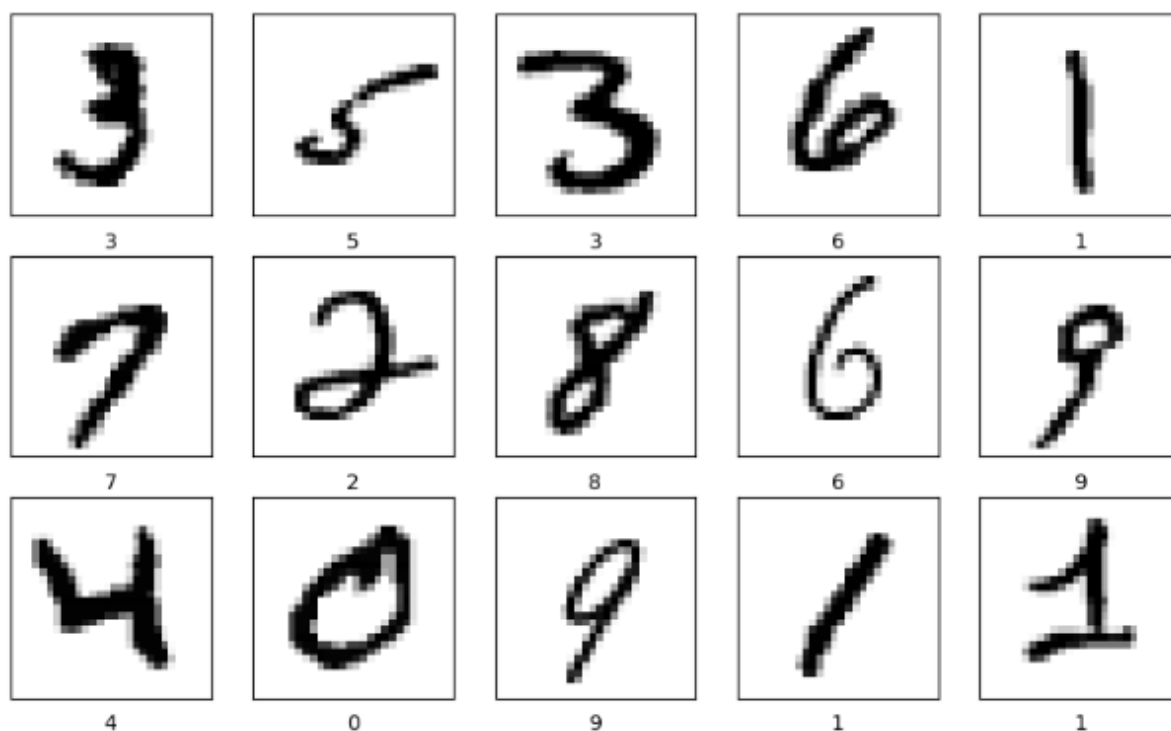
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset/mnist.npz
11490434/11490434 [=====] - 1s 0us/step
```

Вихідні дані зберігаються в тривимірному масиві (60000, 28, 28) типу uint8, значеннями в якому є числа в інтервалі [0, 255]. Такі масиви даних в бібліотеці TensorFlow називаються *тензорами*. Хоча, в математиці під тензорами розуміються дещо складніші об'єкти.

Переглядаємо, наприклад, перші 25 зображень та відповідні їм індекси:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(train_labels[i])
plt.show()
```





Бачимо, що в результаті зображені перші 25 зображень цифр та відповідні їм індекси, що відповідають зображеним цифрам.

Далі ми перетворимо тривимірний масив в двовимірний масив (60000, 28×28) типу float32, а потім нормалізуємо його так, щоб отримати значення в інтервалі [0, 1]:

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

Нормалізація необхідна, щоб наблизити числові значення пікселів до значень вагових коефіцієнтів, що пришвидшують збіжність розв'язку.

Отже, кожне зображення подається вектором довжини 784 і саме компоненти цього вектору будуть подаватися вхідним нейронам.

Тепер підготуємо мітки, тобто завантажимо масиви із відгуками на кожне тестове зображення:

```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Далі передамо нейронній мережі навчальні дані `train_images` і `train_labels`. В результаті цього мережа навчиться зіставляти зображення з мітками. Потім ми запропонуємо мережі класифікувати зображення в `test_images` і перевіримо точність класифікації по мітках з `test_labels`.

**Архітектура моделі.** У Keras є два основні типи моделей: послідовна модель (`Sequential class`) та клас `Model`, що використовується з функціональним API. Ми будемо використовувати клас `Sequential` тому, що наша модель є простою послідовністю шарів – вихідного шару, який складається із  $28 \times 28 = 784$  нейронів, тобто кожному пікселю зображення відповідає один нейрон. Інформація про вхідний шар міститься в описі прихованого шару, який створюється методом `.add()`. Шар фактично є функцією, на вхід якої подається тензор і на виході також отримується тензор, можливо іншого розміру. Ми будемо використовувати *щільні, повнозв'язні* (`Dense`) шари в яких кожен нейрон одного шару з'єднаний з кожним нейроном наступного шару. Прихований шар містить 512 нейронів, а вихідний шар містить 10 класифікаційних нейронів.

За *функції активації* нейронів ми будемо використовувати функції `relu` у прихованому шарі і `softmax` в останньому шарі.

```
from keras import models
from keras import layers
# Визначення типу моделі
network = models.Sequential()
# Визначення прихованого шару
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Опцію `input_shape` можна опустити, тоді розмір вхідних даних визначиться із форми навчальних даних.

**Компіляція мережі.** Після того, як модель визначена, її потрібно підготувати до навчання і компілювати, тобто привести її до вигляду, сумісного із базовою бібліотекою `TensorFlow`. Для цього налаштуємо ще три додаткові параметри:

- *оптимізатор* (optimizer) – конкретний алгоритм, який буде оновлювати ваги в процесі навчання моделі (оптимізатори, які доступні в Keras: RMSProp, sgd, Adagrad, Adadelta, Arguments)
- *функція втрат* (цільова функція) (loss), яку оптимізатор використовує для оцінювання якості своєї роботи на навчальних даних і, відповідно, як коригувати навчання в правильному напрямку (цільові функції, які доступні в Keras: categorical\_crossentropy, logcosh, mean\_squared\_error, hinge, binary\_crossentropy, poisson)
- *метрики* – для оцінки якості проведеного навчання. Метрики оцінюють результати навчання за такими параметрами як точність, коректність, повнота (метрики, які доступні в Keras: accuracy, binary\_accuracy, categorical\_accuracy, sparse\_categorical\_accuracy, cosine\_proximity)

Показники якості схожі з цільовими функціями, але вони використовуються не для навчання моделі, а для оцінки вже навченої моделі.

Рекомендується використовувати такі поєднання цільових функцій, оптимізаторів та метрик:

- для задач класифікації із багатьма класами:

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

- для бінарних задач класифікації із двома класами:

```
model.compile(optimizer = 'rmsprop',
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])
```

Після визначення цих параметрів процес компіляції виконується методом .compile():



```
network.compile(optimizer = 'rmsprop',  
                loss = 'categorical_crossentropy',  
                metrics = ['accuracy'])
```

Детальна інформація про ці параметри компіляції знаходяться на сайті Keras.

**Навчання моделі.** Для навчання викликаємо метод `fit`, який намагається адаптувати модель під навчальні дані. Процес адаптації відбувається у такий спосіб – всі тестові завдання *пакетами* (batch) фіксованого розміру `batch_size` подаються модель, яка ітеративно підганяє вагові коефіцієнти відповідно до виставлених міток з файлу міток. Кожна така ітерація називається *епоху*. Епох повинно бути кілька тому, що кожна нова зміна вагових коефіцієнтів може порушити відгук вже налаштованого раніше тестового завдання, для виправлення потрібна нова ітерація.

```
network.fit(train_images, train_labels, epochs = 5, batch_size = 128)
```

```
Epoch 1/5  
469/469 [=====] - 2s 4ms/step - loss: 0.2533 - accurac  
y: 0.9268  
Epoch 2/5  
469/469 [=====] - 2s 4ms/step - loss: 0.1032 - accurac  
y: 0.9687  
Epoch 3/5  
469/469 [=====] - 2s 4ms/step - loss: 0.0684 - accurac  
y: 0.9795  
Epoch 4/5  
469/469 [=====] - 2s 4ms/step - loss: 0.0502 - accurac  
y: 0.9848  
Epoch 5/5  
469/469 [=====] - 2s 4ms/step - loss: 0.0372 - accurac  
y: 0.9890
```

На навчальних даних ми досягнули точності на навчальних даних 0.9890.

Якщо у прихованому шарі задати, наприклад 128 нейрони, то точність буде 0.9742, але обчислення пройдуть набагато швидше.

Збільшення числа епох до 6 змінює точність до 0.9918. З оптимізатором `sgd` точність 0.9058, з оптимізатором `Adagrad` – 0.9831, `Adadelat` – 0.9852.

**Перевірка моделі.** Після того, як модель навчена, її потрібно перевірити на *контрольному наборі даних*, які ще не пред'являлися моделі. Перевіримо як модель розпізнає контрольний набір:

```
test_loss, test_acc = network.evaluate(test_images, test_labels)

313/313 [=====] - 10s 32ms/step - loss: 0.0717 - accuracy: 0.9774
```

**Збереження моделі.** Команда `model.save(filepath)` зберігає модель Keras в одному файлі HDF5, який буде містити:

- архітектуру моделі, що дозволяє відновити модель
- ваги моделі
- конфігурація тренувань (функцію втрат, оптимізатор)
- стан оптимізатора, що дозволяє відновити навчання саме там, де ви зупинилися

У нашому випадку зберігається модель з іменем `network` у файлі `my_model.h5`:

```
network.save('my_model.h5')
```

Після виконання цієї команди в поточному каталозі має з'явитися файл `my_model.h5`:

☐  my\_model.h5 4 minutes ago 3.28 MB

При потребі можна зберегти лише архітектуру моделі, наприклад у `.json` форматі:

```
json_string = network.to_json()
```

Або можна зберегти лише значення вагових коефіцієнтів:

```
network.save_weights('my_model_weights.h5')
```

Ці коефіцієнти пізніше можна завантажити в моделі з іншими архітектурами.

**Робота із моделлю.** Завантажуємо попередньо збережену модель:

```
from keras.models import load_model
model = load_model('my_model.h5')
```

Для класифікації зображення із тестового файлу (наприклад test.png) із цифрою зберігаємо його в каталозі зі виконуваним файлом:



Завантажуємо його в OpenCV як сіре зображення і отримуємо данні у форматі ndarray. Потім цей масив переформатовується до того розміру на якому відбувалося тренування нашої моделі:

```
import cv2
tst = cv2.imread('C:/Users/Andrii Nesteruk/Desktop/test.png', 0)
tst = cv2.resize(tst, (28, 28))
tst = tst.reshape((1, 28*28))
tst = tst.astype('float32') / 255
```

Підготовлені данні передаємо функції model.predict(tst). На виході отримаємо список списків із одним елементом – списком довжини 10, у якому на  $i$ -ій позиції знаходяться ймовірність того, що на вхідному зображенні є число  $i$ . Нам потрібна позиція з максимальною ймовірністю.

```
pred=list(model.predict(tst)[0])
print(pred.index(max(pred)))
```

```
1/1 [=====] - 0s 13ms/step
3
```

Бачимо, що модель коректно визначила зображену цифру. Повторимо тест з ще трьома цифрами (4, 5 та 9):



І відповідні їм результати роботи програми:

```
pred=list(model.predict(tst)[0])  
print(pred.index(max(pred)))
```

```
1/1 [=====] - 0s 14ms/step  
4
```

```
pred=list(model.predict(tst)[0])  
print(pred.index(max(pred)))
```

```
1/1 [=====] - 0s 13ms/step  
5
```

```
pred=list(model.predict(tst)[0])  
print(pred.index(max(pred)))
```

```
1/1 [=====] - 0s 15ms/step  
9
```

### Завдання:

1. Виконати завдання із прикладу і отримати файл із навченою моделлю для розпізнавання рукописних цифр. В будь якому графічному редакторі створити файл із рукописною цифрою і розпізнати її. Пояснити результат.

2. Спроекувати і розробити нейронну мережу на основі таких наборів даних імплементованих в Keras:

#### 1) Cifar10

```
from keras.datasets import cifar10  
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Формат даних 2 кортежі:

- x\_train , x\_test : uint8 array of RGB image data with shape (num\_samples, 3, 32, 32) or (num\_samples, 32, 32, 3) based on the image\_data\_format backend setting of either channels\_first or channels\_last respectively
- y\_train , y\_test : uint8 array of category labels with shape (num\_samples, 1)

Набір містить фото таких 10 категорій об'єктів: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

#### 2) FMNIST

```
from keras.datasets import fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

Формат даних 2 кортежі:

- x\_train , x\_test : uint8 array of grayscale image data with shape (num\_samples, 28, 28)
- y\_train , y\_test : uint8 array of labels (integers in range 0-9) with shape (num\_samples)

Набір містить фото таких 10 категорій об'єктів: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.

**Потрібно розробити архітектуру нейронної мережі, навчити її на тестових прикладах, і продемонструвати її роботу на кількох зображеннях.**

**3. Зробити звіт про роботу, який включає:**

**а. Титульна сторінка з інформацією про виконавця, темою та номером лабораторної роботи**

**б. Постановку завдання**

**в. Скріни коду та скріни результату виконання з коментарями**

**г. Висновок**

**4. Надіслати звіт(.docx або .doc) на @zeit\_13 (Telegram) або aonesterukr@gmail.com (e-mail). В повідомленні обов'язково вказати Ваше ПІБ, групу, назву предмету (скорочено)**

**5. Записати захист роботи на відео та надіслати його разом зі звітом на вище вказані адреси. Під час захисту роботи показати роботу програмного коду! (не звіт) та пояснити свої дії.**