ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «РОЗРОБКА НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ PYTHON»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Віталій ЗАХАРЧУК*

Виконав:
здобувач вищої освіти Віталій ЗАХАРЧУК
група ІСДМ-63

Керівник: Оксана ТКАЛЕНКО
науковий ступінь, вчене звання к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____Каміла СТОРЧАК
«____» _____20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ СТУДЕНТУ
Захарчуку Віталію Григорійовичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Розробка нейронної мережі для розпізнавання об'єктів з використанням мови програмування Python»

керівник кваліфікаційної роботи Оксана ТКАЛЕНКО, к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «19» жовтня 2023 року № 145.

2. Строк подання кваліфікаційної роботи: 29 грудня 2023 року.

3. Вихідні дані до кваліфікаційної роботи: Методи http GET, POST, PUT, PATCH, DELETE;
Протоколи HTTP, HTTPS, SSH;
Фреймворки TensorFlow, Keras, PyTorch, Theano.
Науково-технічна література з питань, пов'язаних з наукою про дані.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження особливостей науки про дані.

2. ML/DL – цілі та принципи роботи.
3. Розробка нейронної мережі для розпізнавання рукописних цифр.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Методологія по дослідженню даних CRISP-DM.
   2. Порівняльна характеристика ML та DL.
   3. Фреймворк TensorFlow.
   4. Реалізація нейронної мережі в середовищі Jupyter Notebook.

6. Дата видачі завдання: 19 жовтня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10 – 25.10.23	
2	Дослідження особливостей науки про дані	26.10 – 02.11.23	
3	Дослідження особливостей машинного та глибокого навчання	03.11 – 09.11.23	
4	Дослідження особливостей побудови нейронних мереж	10.11 – 16.11.23	
5	Вивчення особливостей фреймворків TensorFlow, Keras, PyTorch, Theano	17.11 – 21.11.23	
6	Практична реалізація нейронної мережі	22.11 – 11.12.23	
7	Оформлення роботи: вступ, висновки, реферат	12.12 – 20.12.23	
8	Розробка демонстраційних матеріалів	21.12 – 28.12.23	

Здобувач вищої освіти _____ Віталій ЗАХАРЧУК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник роботи
кваліфікаційної роботи _____ Оксана ТКАЛЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: ___стор., ___рис., ___табл., ___джерел.

Мета роботи – проведення комплексного аналізу методів і технологій обробки даних в інформаційних системах, а також розробка нейронної мережі для розпізнавання об’єктів.

Об'єкт дослідження – процес обробки даних в інформаційних системах.

Предмет дослідження – методи і технології обробки даних в інформаційних системах.

Короткий зміст роботи: Досліджені методи та технології обробки даних в інформаційних системах, математичні інструменти для аналізу та обробки даних. Досліджені особливості машинного та глибокого навчання, особливості побудови нейронних мереж, найпопулярніші фреймворки для роботи з технологіями ML/DL: TensorFlow, Keras, PyTorch, Theano, визначені їх позитивні сторони та недоліки, надані рекомендації по використанню зазначених фреймворків. Розроблено нейронну мережу для розпізнавання рукописних цифр в інтерактивному середовищі Jupyter Notebook, яке є популярним інструментом серед дослідників, програмістів та аналітиків даних. Визначено роль штучного інтелекту в аналізі та обробці даних. Здійснено програмування нейронної мережі.

КЛЮЧОВІ СЛОВА: ДАНІ, НЕЙРОННА МЕРЕЖА, МОВА ПРОГРАМУВАННЯ PYTHON, ВІЗУАЛІЗАЦІЯ, ТЕХНОЛОГІЯ, АЛГОРИТМ, СОКЕТ, ПРОТОКОЛ, ІНТЕРАКТИВНЕ СЕРЕДОВИЩЕ.

ABSTRACT

Text part of the master`s qualification work: ____ pages, ____ pictures, ____ table, ____ sources.

The purpose of the work is to carry out a comprehensive analysis of data processing methods and technologies in information systems, as well as the development of a neural network for object recognition.

Object of research is the process of data processing in information systems.

Subject of research is methods and technologies of data processing in information systems.

Summary of the work: Methods and technologies of data processing in information systems, mathematical tools for data analysis and processing were studied. The features of machine and deep learning, the features of building neural networks, the most popular frameworks for working with ML/DL technologies: TensorFlow, Keras, PyTorch, Theano were investigated, their positive sides and disadvantages were determined, and recommendations were given for the use of these frameworks. A neural network was developed to recognize handwritten digits in the interactive Jupyter Notebook environment, which is a popular tool among researchers, programmers, and data analysts. The role of artificial intelligence in data analysis and processing was determined. Neural network programming was carried out.

KEYWORDS: DATA, NEURAL NETWORK, PYTHON PROGRAMMING LANGUAGE, VISUALIZATION, TECHNOLOGY, ALGORITHM, SOCKET, PROTOCOL, INTERACTIVE ENVIRONMENT.

ЗМІСТ

ВСТУП

Актуальність теми: Останні роки принесли значний прогрес у глибинному навчанні, яке базується на використанні нейронних мереж з багатьма шарами (глибинними мережами). Це дозволяє досягати кращих результатів у розпізнаванні образів, класифікації даних та інших завданнях. Нейронні мережі можуть адаптуватися до змін у вхідних даних та оточуючому середовищі, що робить їх гнучкими та ефективними в умовах невизначеності. Нейронні мережі добре справляються з великими обсягами даних, такими як зображення, аудіо, текст, можуть виявити приховані патерни та взаємозв'язки у цих даних, що робить їх ефективними у різних сферах використання. Загалом, нейронні мережі стали потужним інструментом у сучасній інформаційній та технологічній епосі, допомагаючи вирішувати складні завдання та покращувати якість багатьох аспектів людського життя, тому дослідження в цій області є актуальними.

Метою роботи є проведення комплексного аналізу методів і технологій обробки даних в інформаційних системах, а також розробка нейронної мережі для розпізнавання об’єктів.

Для досягнення поставленої мети необхідно *виконати наступні завдання*:

- дослідити методи та технології ML/DL, особливості побудови нейронних мереж;
- дослідити інструменти для аналізу та обробки даних, фреймворки для роботи з технологіями ML/DL;
- виконати обробку даних, використовуючи бібліотеки мови програмування Python;
- розробити нейронну мережу для розпізнавання рукописних цифр в інтерактивному середовищі Jupyter Notebook.

Об'єкт дослідження – процес обробки даних в інформаційних системах.

Предметом дослідження є методи і технології обробки даних в інформаційних системах.

Наукова новизна отриманих результатів полягає у розробці нейронної мережі для розпізнавання об’єктів з використанням класу моделі Sequential, оптимізатору Adam.

Постійно розробляються нові методи та інструменти для обробки даних, такі як машинне навчання, штучний інтелект, обробка природної мови та інші. Дослідження їхнього застосування в інформаційних системах є актуальним завданням.

Апробація результатів магістерської роботи:

Захарчук В.Г. «Аналіз методів регуляризації у машинному навчанні». Тези доповіді на І Всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і Світу». – Київ, 28 листопада 2023 року.

1 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ НАУКИ ПРО ДАНІ

1.1 Дослідження особливостей штучного інтелекту, машинного та глибокого навчання

Коли до нас приходять з абстрактною задачею, то цю задачу потрібно перетворити в якийсь процес, тобто це потрібно якось ітерувати.

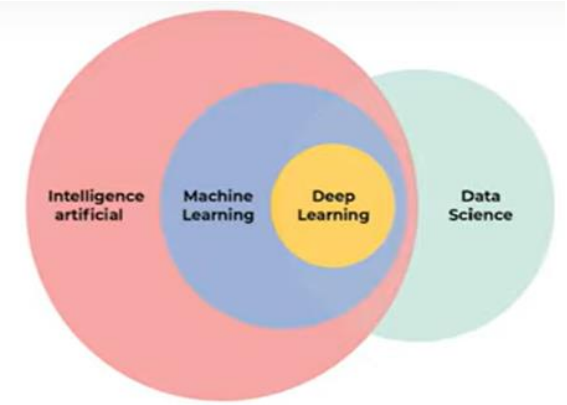


Рисунок 1.1 – Data Science

Штучний інтелект – величезний набір методів, інструментів, речей і в рамках штучного інтелекту є один із підходів побудови штучного інтелекту – це машинне навчання. Машинне навчання у нашому випадку – це набір визначених моделей (наприклад, лінійна регресія, логістична регресія, дерево рішень тощо), тобто коли ми на основі наявних у нас даних витягуємо якісь статистики з даних, якісь правила з даних і це все відбувається автоматично на основі тих даних, які ми маємо. Тобто машинне навчання, маючи якийсь набір даних, ми на ньому навчаємося передбачати. Коли ми говоримо про глибоке навчання DL – це нейронні мережі. DL – ми навчаємося на основі витягнутих нами автоматично даних. Будь-яка нейронна мережа розкладується до найпростіших алгоритмів. DL – специфічні алгоритми, які навчаються одночасно. Глибокий алгоритм складно пояснити. У машинному навчанні класичні алгоритми (лінійна регресія, логістична регресія, дерево рішень).

Тобто у нас є величезний набір методів штучного інтелекту. В ньому є кластер поменше – це машинне навчання – набір всіх алгоритмів, які у нас навчаються на

основі наших даних. І є глибоке навчання (DL) – коли в нас один алгоритм складається із блоків дуже схожих і наступний блок навчається на основі попереднього.

Якщо ми говоримо про Data Science – то в цілому це загальний підхід до роботи з даними, тобто це і візуалізація даних, це і аналітика поверх них, це і очистка даних, це і якісь висновки вже по навченим моделям, тобто моделювання і машинне навчання – дуже велика частина Data Science, але не більше половини. ML-спеціаліст дуже гарно розбирається у визначених алгоритмах машинного навчання, він вміє їх дуже гарно навчати. Data Scientist може взяти модель, яку вже навчив ML-інженер, і Data Scientist навколо неї зробить всю необхідну красу, яку потрібно зробити. Дата-інженер підводить дані до ML-інженера і в цьому випадку Data Scientist буде не одна визначена людина, а це буде ціла велика команда.

1.2 Методологія по дослідженню даних CRISP-DM

Коли ми говоримо про Data Science, то говоримо про CRISP-DM (рис.1.2). Будь-який Data Science проект, дослідження – має проходити всі ці етапи. Цей процес працює протягом всього проєкту. Спочатку беремо задачу, робимо BU (розуміємо, що хоче бізнес), тоді розуміємо, які дані є у бізнеса, які можуть бути, потім ми ці дані підготовлюємо і тільки після цього починається наш ML (робимо моделювання) і далі по метриці, яку ми заклали на етапі BU – ми вже робимо оцінку і якщо все добре, то по цій оцінці ми робимо Deployment (розгортання). А якщо ні – наша пісня добра, починай спочатку. Як бачимо в моделі (рис.1.2) стрілочок дуже багато, процес такий непрямий. Буває таке, що треба поговорити з бізнесом, подивитися на дані, тоді знову треба поговорити з бізнесом, тоді підготувати дані, промоделювати, повернутися знову назад і зрозуміти, що все ж ні, потрібно дані по іншому підготувати для того, щоб моделювати. Будь-який успішний проєкт можна розкласти по таким крокам.

Рисунок 1.2 – CRISP-DM

Король машинного навчання – Supervised learning (навчання під наглядом) – це, коли у нас є сирі дані та на додачу до них є якісь відповіді. Наприклад, коли ми говоримо про задачу класифікації котів та собак, то у нас картинки підписані, ми зараніш знаємо, що у нас на одній картинці кішка, а на іншій картинці – собака. І у нас є лейбли, тобто у нас є відповіді, не тільки самі картинки. На базі цих відповідей ми й будемо навчати нейронну мережу, яка й буде вирішувати задачу класифікації.

Коли ми говоримо про Unsupervised learning – то в нас є тільки картинки без відповідей. В такому випадку у нас задача ставиться по іншому, ставиться задача кластеризації. Задача кластеризації має на увазі те, що ми повинні об’єднати схожі картинки і далі вже виконати поверх цього якусь аналітику. Задача кластеризації непередбачувана і її використовують зазвичай тоді, коли вже в нас немає необхідних відповідей і нам потрібно хоча б з чогось почати. Тобто задача кластеризації – це коли ми намагаємося знайти щось схоже.

Критерій схожості - це все задається індивідуально. Можна маніпулювати. 85% задач production – це Supervised learning. Тобто у нас зараніше були дані і до них є відповіді.

Є ще область – Reinforms learning – навчання з підкріпленням – ідея полягає в тому, що ми навчаємо агента, який вміє працювати у визначених умовах.

Наприклад, ми зараніше прописали правила гри у шахи і далі грубо кажучи комп'ютер грає сам із собою і він намагається сам себе обіграти, тобто він придумує нові ходи, запам'ятовує де й що стало краще, де й що стало гірше, тобто комп'ютер навчається за рахунок оточення. Коли ми говоримо про шахмати, то оточення – це правила, в яких ми знаходимося, а друга частина оточення – це суперник і популярний підхід до навчання з підкріпленням заключається в тому, що ми вибрали якогось агента, навчили його до якогось стану і далі наш агент стає вже оточенням і ми навчаємо другого агента для того, щоб він адаптувався і почав перемагати вже навченого агента.

Таблиця 1.1 – Машинне навчання

	Unsupervised learning	Supervised learning	Reinforms environment
Дані	Raw data	Raw data + labels	Raw data + environment
Задача	Кластеризація	Класифікація	Навчання агента
Приклад	Пошук схожих продуктів у магазині	Класифікації порід собак	Агент для гри в шахи

Навчання з підкріпленням добре використовується там, де ми можемо гарно визначити оточення. Наприклад, у математичному моделюванню Reinforms learning працює дуже гарно. Але таких проєктів дуже мало. У комерційних розробках майже завжди використовується Supervised learning.

First 5 rows

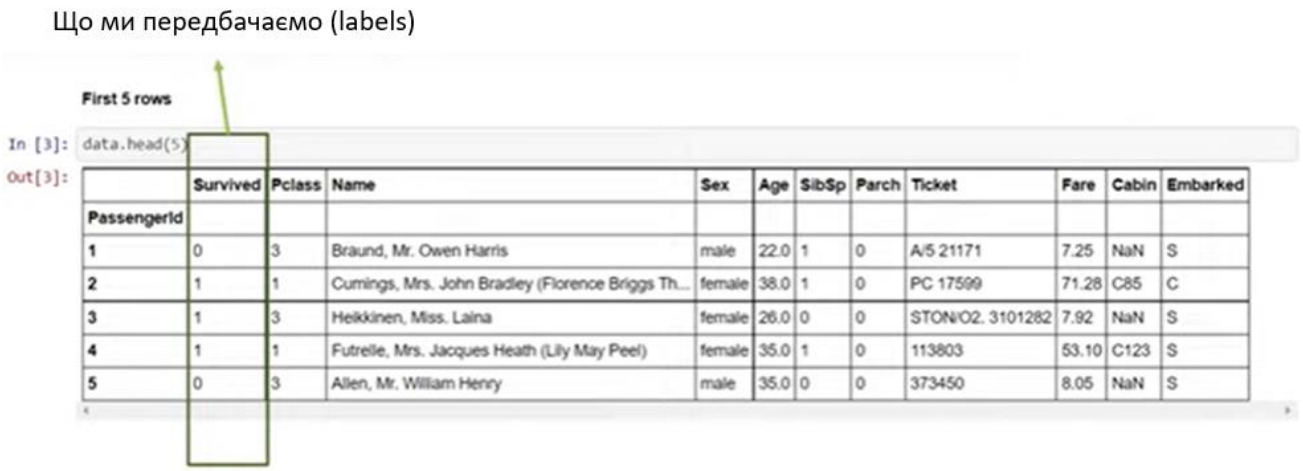
```
In [3]: data.head(5)
```

Out[3]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embar
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.28	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.92	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.10	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05	NaN	S

Рисунок 1.3 – Supervised learning

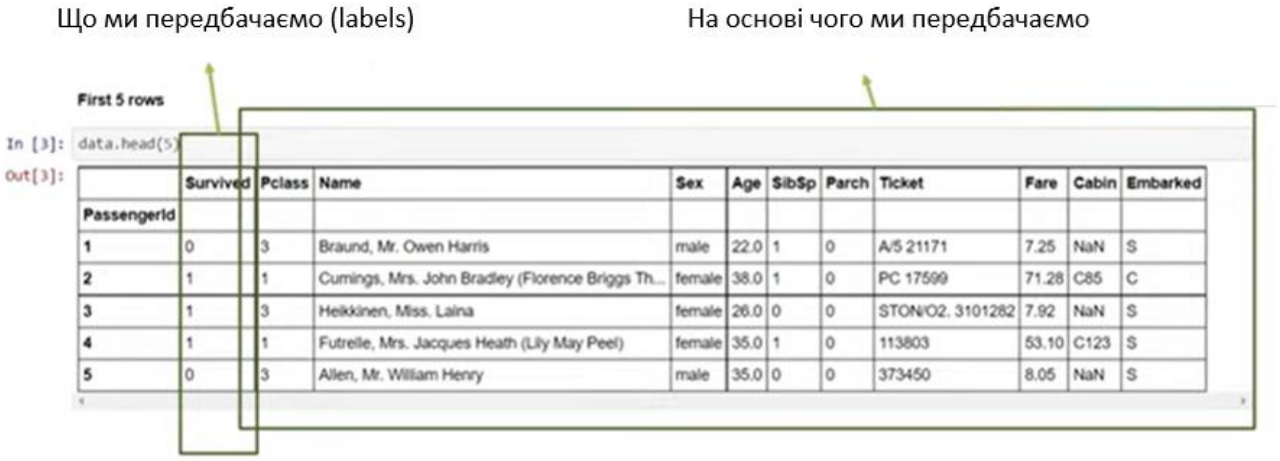
Всі поля перекладаємо на Excel.



	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId											
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.28	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.92	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.10	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05	NaN	S

Рисунок 1.4 – Data set

На рис.1.4 приведений популярний набір даних із змагань Kaggle, де потрібно передбачати, вижила людина на кораблі Титаник, чи померла. Тобто у нас є якась інформація про цю людину, в якому класі він знаходився, як його звали, яка в нього була стать, вік, номер його білету, скільки він заплатив, в якому порті він сів і що найголовніше у нас є інформація, про яку ми хочемо дізнатися, тобто це безпосередньо вижила людина або ні (рис.1.5).



	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId											
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.28	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.92	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.10	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05	NaN	S

Рисунок 1.5 - Supervised learning

У нас є те, на основі чого ми передбачаємо. Коли ми навчаємо який-небудь алгоритм машинного навчання, він намагається з наших якихось полів, із властивостей кожного пасажира, він намагається їх скомпонувати, щоб зрозуміти (відповідно ми говоримо й про дерево рішень), він подивиться на ті дані, які є в

нас, і побачить, що там люди молодше 22 років і старше 38 років – померли. Може використовуватися умовний оператор, якщо жінка – вижили, якщо чоловік – померли. Ця вибірка якраз і розділяє. Але у великих data set таких розділяючих властивостей зазвичай немає.

Що важливо розуміти. Коли ми говоримо про Supervised learning, у нас є колонка, яку ми передбачаємо, тобто вижила людина або ні і у нас є набір властивостей, на основі яких ми робимо передбачення. У цьому вся ідея Supervised learning. Далі деталі – які алгоритми ми використовуємо, як ми процесимо перед цим, яким препроцесінгом ми це повинні зробити, що є результатом роботи цього алгоритму, тобто всі ці деталі потрібно дивитися, коли ми вирішуємо визначену задачу. Тобто, якщо ми в цілому говоримо про задачу Supervised learning, то це на основі властивостей передбачити якусь характеристику.

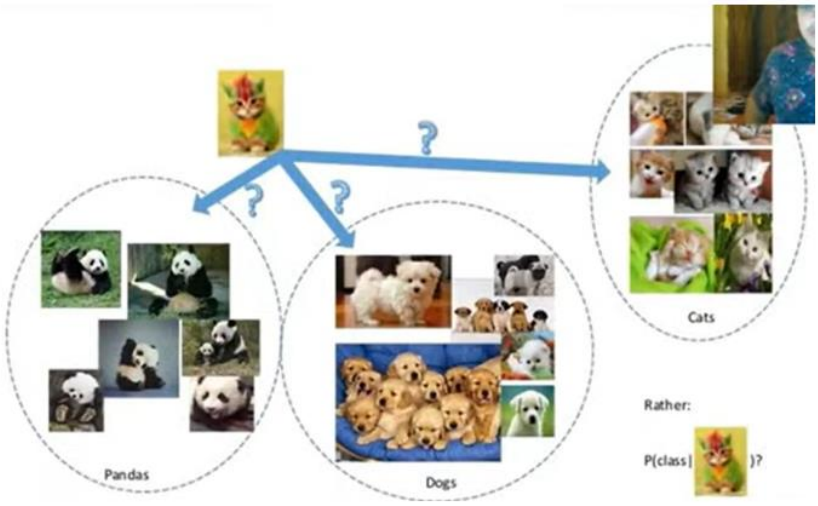


Рисунок 1.6 – Приклад Supervised learning

Якщо ми говоримо про картинки, те що в кружечках – те, на чому ми навчалися. І у нас є картинка, яку ми раніше не бачили (розмальована кішка). І ми повинні зрозуміти, до якого класу віднести цю розмальовану кішку для того, щоб її правильно класифікувати. Далі, у нас є алгоритм, який визначає в один із класів нашу картинку. Тобто, якщо в нашому випадку, це кішка і передбачення буде правильним, то воно відправить його у правий клас – клас котів. Відправити воно

може його і в клас панди, і в клас собаки, і тоді це буде неправильне передбачення. Попередня ідея така ж сама. Тут ми говорили про табличні дані і в нас прямо у колонках були ці відповіді. А у випадках з картинками (котики, панди, собачки) – кожний піксель – це наша характеристика, метадані – це назви наших файлів (коти – cats і якийсь номер, собаки – dogs і якийсь номер).

1.3 Дослідження навчання з підкріпленням

Поговоримо про навчання з підкріпленням (рис.1.7).

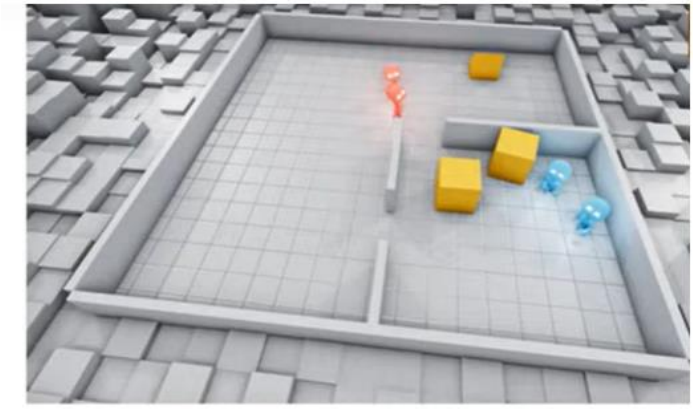


Рисунок 1.7 – Навчання з підкріпленням (Reinforcement learning)

Є агенти, які ховаються (сині) і є агенти, які намагаються знайти тих, хто заховався (червоні). І є декілька об’єктів, якими можна маніпулювати. Тобто одні агенти фіксуються, інші – навчаються під них адаптуватися. Тобто агенти, які ховаються, навчилися блокувати вхід у ту кімнату, в якій вони ховаються. Але через деякий проміжок часу червоні все рівно навчилися їх знаходити й навчилися використовувати трамплін для того, щоб пробиратися всередину їх кімнати. Але це відбувається не миттєво, а за великий проміжок часу. Це можуть бути дні комп’ютерних обчислень цих всіх кроків. Тобто, ідея полягає в тому, що у нас є певне оточення, у нас є умова цього оточення і наші алгоритми адаптуються до оточення, яке ми маємо. І за рахунок цього відбувається таке навчання. Тобто у нас немає яскраво виражених міток, які ми могли б використовувати як у випадку з задачею класифікації. Тобто тут ми використовуємо оточення, як інструмент для навчання.

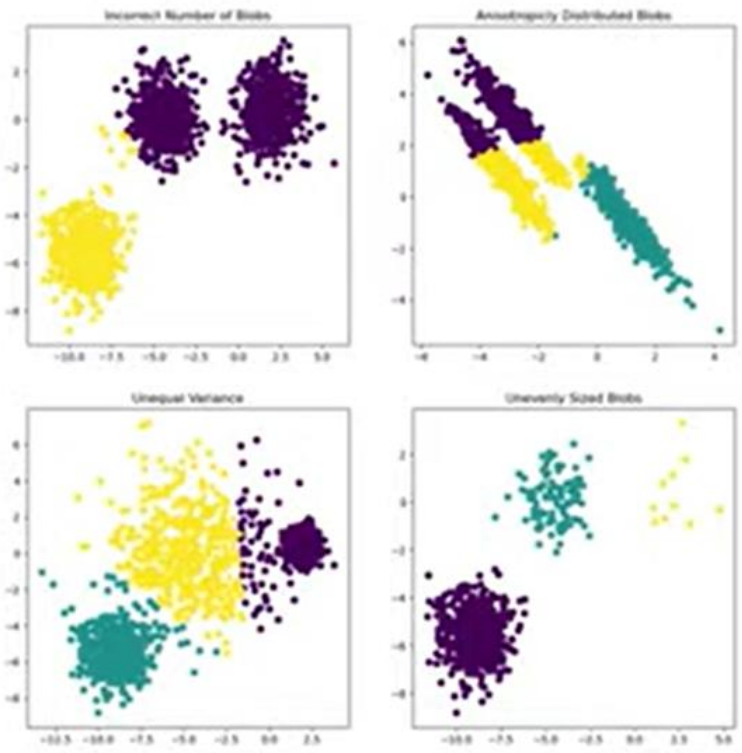


Рисунок 1.8 – Unsupervised learning (приклад задачі кластеризації)

На кожній картинці є три яскраво виражених кластери. Але алгоритми за своєю ідеєю, як вони об’єднують, схожі. У нас по суті немає ніяких відповідей. Але це вже добре, те що зображене на рис.1.8. Коли є таке розбиття на кластери, ми вже можемо робити якісь висновки, у нас вже є частина роботи, зроблена за нас. Ми вже можемо не з пустого починати аналітику, коли все у нас було б жовтим, а ми вже злегка його якось підрозділили. Зачасту алгоритми кластеризації працюють гірше, ніж це очікується. Але, як доводить практика, гарні проєкти в Data Science будуються на основі гарних даних. І алгоритми в Data Science – це вторинне. А первинне – це дані. І задача Supervised learning набагато простіше і точніше.

У результаті проведеного аналізу узагальнимо різницю між машинним навчанням і глибоким навчанням (рис.1.9).

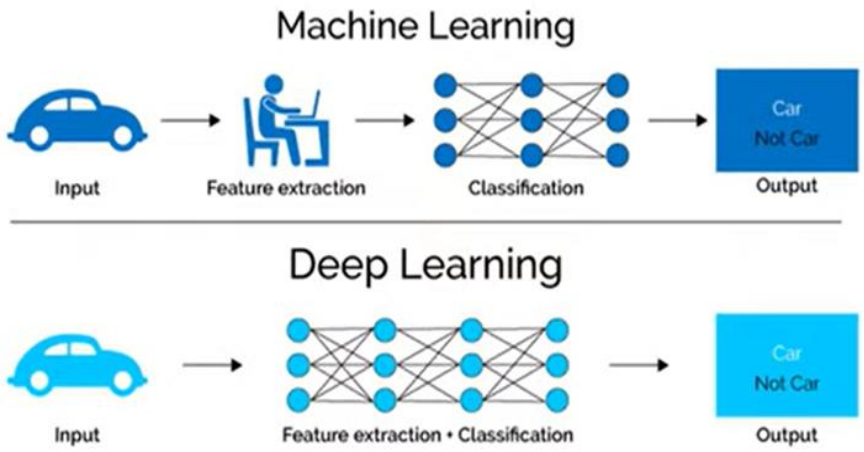


Рисунок 1.9 – Машинне навчання (ML) / глибоке навчання (DL)

Якщо до нас на вхід прийшли картинки, ми можемо взяти і почати застосовувати якісь детерміновані, сформовані алгоритми і намагатися знаходити якісь характеристики у цих об’єктів. Наприклад, стільки то зелених пікселів, стільки то синіх пікселів, стільки то червоних пікселів. У нас є якийсь препроцесінг, який ми застосовуємо до картинки, тобто ми його робимо автоматично, він передвизначений, для всіх картинок він однаковий, ми ці властивості вилучили і віддали це на класифікацію. І вже дізнаємося є в нас машина на картинці чи немає. На основі тих властивостей, які ми самі визначили.

Коли ми говоримо про глибоке навчання, то ми вже нічого далі з картинкою не робимо. Вся задача по вилученню якихось властивостей стає автоматичною. Тобто ми пропрацьовуємо всі ці пікселі, ми з них згортками вилучаємо якісь згиби і воно підбирається під наші дані. Тобто згорткова нейронна мережа, яка гарно підходить під цю задачу, вона буде автоматично шукати якісь властивості, які характерні для машини і казати, якщо ці властивості є присутніми (тут знайде властивості, тут знайде комбінацію цих властивостей і т.д.), то вона буде казати, якщо ці властивості є присутніми, то на цій картинці є машина, якщо ці властивості відсутні – то це буде означати, що машини немає.

Тобто у випадку машинного навчання ми самі готуємо властивості, на яких буде навчатися алгоритм, а у випадку глибокого навчання ми віддаємо сирі дані в хід, практично нічого для цього не роблячи.

Саме головне, що є в Data Science і в машинному навчанні – це дані. Дослідимо, які є дані (табл.1.2).

Таблиця 1.2 – Типи даних

Табличні дані	**Зображення**	**Текст**	**Звук**
Оцінка кредитоздатності	Виявлення дефектів на виробництві	Виявлення спаму	Ідентифікація по голосу
Ймовірність переходу на інший тарифний план	Самокеровані автомобілі	Перекладачі	Переведення голосу в текст
Контекстна реклама	Ідентифікація по обличчю	Автодоповнення	Видалення шумів

Зараз дані можна поділити на 4 типи. Самі зрозумілі дані, з якими ми часто маємо справу – це табличні дані. Якщо ми відкриваємо Excel або якийсь звіт, то ми там знаходимо табличні дані. Це якісь логи, списки транзакцій. Гарний приклад задачі, яка вирішується на основі табличних даних – це оцінка кредитоздатності. Тобто банк знає, скільки ми заробляємо грошей щомісяця, знає скільки у нас транзакцій, як часто ми витрачаємо ці кошти й на основі нашої активності збирається статистика по нам, робиться підготовка фічів, якихось характеристик нас як особистості і говорять, що значить ця людина, яка стільки то заробляє, стільки витрачає (в нього витрати такі то, покупки такі то), у таких то магазинах, то мабуть йому можна видати ось такий кредит. Наприклад, в Америці, якщо нам відмовили в кредиті, вони повинні назвати причину, по якій вони нам відмовили і коли вони називають цю причину, по якій вони відмовили, це могло бути зроблено якоюсь нейронною мережею. Коли це в нас дерево рішень з якого можна легко витягнути правила, по яким нас визначило кудись не туди, ми можемо дати таку відповідь.

Так само і мобільний оператор збирає інформацію: кому ми і скільки телефонуємо, скільки ми телефонуємо на інших операторів, скільки ми користуємося Інтернетом, яка тривалість дзвінка, як часто ми поповнюємо рахунок, на яку суму. Виходячи з цього він може в себе передбачати, які в нього будуть прибутки, збитки, коли люди перейдуть на більш дорогий тарифний план, на більш дешевий тарифний план і на основі такого табличного плану він може сформулювати новий табличний план. Він може спрогнозувати, яке буде використання мережі, яке буде навантаження на неї. На основі просто даних про те, як ми користуємося нашим мобільним зв'язком.

Так само і контекстна реклама визначає які сайти ми відвідуємо, той самий Google – які запити ми шукаємо, на основі цього ті ж самі cookies - ми всі даємо погодження на їх використання. Крім того, щоб все було зручно, вони потрібні для контекстної реклами, щоб Інтернет, який щось рекламує, розумів, хто ми такі.

Табличні дані в нашому випадку – якісь фіксовані транзакції. У нас є активність користувача за місяць і ми на основі цієї активності застосовуємо якісь фільтри, дивимось скільки разів він відвідав сайт такої то категорії і ми, виходячи з цієї активності вже самі формуємо табличні дані, на яких ми будемо навчати наш алгоритм.

Наступний вид даних – робота із зображеннями. Це у нас комп'ютерний зір, тут є багато застосувань. Наприклад, є у нас лінія виробництва, ми ставимо камеру, камера знаходить якісь об'єкти, які були виконані невірно, по ним очевидно, що на них якийсь дефект і вона їх відкидає, забраковує. Інше – це самокеровані автомобілі. Саме автопілот базується на зображеннях. На теслі знаходяться камери, вони знімають і далі по картинці ми розуміємо, що саме відбувається. У телефонах є функція ідентифікація по обличчю.

Третій тип даних – це текст. У нас на поштах є такий відділ, як спам і зараз на всіх поштових сервісах класифікатори спаму використовують засоби машинного навчання, засоби глибокого навчання. У них багато властивостей. Значна частина цього відбувається на основі контексту листів. Вони розуміють, які слова

використовуються у нас у спамі, а які слова в ньому не використовуються. На основі цього ми можемо зробити класифікацію тексту спам це чи не спам.

Четвертий тип даних, який найменш розповсюджений – це звук, але задачі, які вирішуються із звуками, до цих пір є складними. Це пов’язано з тим, що такі задачі на теперішній час є не дуже популярними і не отримали затребуваності на ринку. Але в нас є задачі ідентифікації по голосу. Тобто, якщо ми говоримо про всі розумні будинки, то ідентифікувати голос 5-6 людей цілком можна, нейронні мережі з цим справляються, цих 5-6 людей відрізняють, можна видати якісь права в залежності від того, який у людини голос, тобто дитині не дозволяти включати кондиціонер і змінювати температуру в будинку, а дружині дозволити.

Наступна задача – це переведення голосу в текст. Це також робота нейронних мереж, які розпізнають голос і перетворюють його в текст. Третій гарний приклад роботи із звуком – це видалення шумів. Дуже часто звукова доріжка чиститься за допомогою нейронних мереж. Звукову доріжку можна конвертувати у спектрограму і це вже стає зображенням. І далі працюємо як із зображенням. Тобто є проєкти, де люди працюють із звуком.

2 ML/DL – ЦІЛІ ТА ПРИНЦИПИ РОБОТИ

2.1 Дослідження особливостей машинного та глибокого навчання

Станом на теперішній час одним з популярним напрямків у сфері аналізу та обробки даних є нейронні мережі, їх побудова, використовуючи той чи інший фреймворк або написання з нуля, їх навчання, отримання предикції з метою розуміння можливостей нейронних мереж. І другий блок – блок, призначений програмуванню мультимедіа і комп'ютерної графіки.

Проведемо аналіз того, чим людський інтелект і взагалі людська нервова система відрізняється докорінно від машинної. Та й не тільки людська, а взагалі будь-якої живої істоти. Докорінна відмінність полягає в тому, що наша нервова система і наш інтелект може змінюватися, підлаштовуватися під обставини, під середовище оточення, тренуватися. Тобто ми здатні еволюціонувати як на рівні нашого виду, так і на рівні кожного індивідууму. Ми народжуємося з якимись базовими інстинктивними навичками, а потім ми починаємо вдосконалювати те, що у нас є і набувати те, чого у нас не було. Ми йдемо в дитячий садок – нас там навчають, ми йдемо в школу – нас там навчають, потім потрапляємо в університет і може навіть не в один, і з нами там теж роблять цікаві речі і в процесі цих цікавих речей нам потрібно наш інтелект підлаштовувати під оточуючі обставини. Коли ми говоримо про інтелект, то ми говоримо про здатність носія цього інтелекту змінюватися, змінювати своє відношення до того чи іншого процесу.

Якщо ми будемо порівнювати штучний інтелект з машинним навчанням: машинне навчання базується на статичних коефіцієнтах. Тобто, ми з вами беремо модель, натреновуємо її на якомусь дата сеті, у неї випрацьовується певний набір коефіцієнтів реакції на ті предиктори, які ми можемо до неї передавати і ми передаємо на неї дані, вона зводить їх до коефіцієнтів, порівнює з тим, що в неї є з тією сіткою і на підставі цієї сітки дає нам відповідь і все. А з самою моделлю при цьому нічого не відбувається. Вона як була в тому стані, як ми її навчили, так вона у нас і залишилася. Ось чому моделі машинного навчання, у тому числі і моделі роботи з часовими рядами – їх дуже стрьомно використовувати для

прогнозу таких речей, які пов’язані з грошима. Гроші знаходяться у прямому зв’язку з діяльністю самої людини. Людина може поміняти, наприклад, економічні умови. А модель машинного навчання працює по тим навченим матеріалам, які вже були. Ось тому ми для того, щоб отримувати більш менш правдоподібні прогнози, маємо постійно перенавчати наші моделі машинного навчання по оновленим даним. Тільки в цьому випадку ми можемо розраховувати на те, що вони нам дадуть прийнятний результат. При інших обставинах вони нас будуть обманювати.

А от що стосується нейронних мереж, систем штучного інтелекту, ситуація зовсім інша. Їх розробили таким чином, що завдяки певним механізмам, які в них закладені, дана модель може донавчатися в процесі своєї експлуатації самостійно. Ми створили якусь певну модель, навчили її на якихось певних навчальних наборах, от на певних картинках ми її навчили відрізняти породи собак.

Знаю одну людину, яка працює у волонтерській організації, яка надає зараз допомогу безпритульним тваринам. У них дійсно була реальна проблема. Приходить собака, а собак бездомних приходить дуже багато і їх потрібно якимось чином класифікувати: чи то німецька овчарка, чи то кавказька овчарка, чи то спанієль. А не всі ветеринари, які сидять на сортуванні, розбираються у цих породах собак. І він зробив модель: наводимо фотокамеру на собаку – вона приймає її зображення і з певною ймовірністю дає заключення, що це, наприклад, така порода як чіхуахуа – на стільки то відсотків. А потім приносять ще одну чіхуахуа – чорного кольору і такої чіахуахуа у навчальному наборі не було. От модель починає вже сама автоматично навчатися. З якогось певного періоду вона починає розпізнавати нові зображення, відносити їх до якихось певних існуючих порід. Ось це приклад роботи нейронної мережі моделі штучного інтелекту.

Моделі машинного навчання займають певне місце в Data Science. У деякому випадку вони відпрацьовують як кінцевий програмний продукт. Таких прикладів зараз дуже багато. Але з появою нейронних мереж, дані моделі почали досить часто використовуватися як допоміжні – ті, які використовуються на початковому етапі обробки даних. Перед тим, як передати якусь інформацію в нейронну

мережу, ми спочатку пропускаємо її, наприклад, через дерево ухвалення рішень, отримуємо якийсь певний клас і вже потім із цим показником певного класу і передаємо на нейронну мережу, яка робить з цими даними щось.

У табл.2.1 приведені сфери застосування машинного навчання (ML), глибокого навчання (DL). Зліва – сфери застосування ML, справа – сфери застосування DL.

Таблиця 2.1 - Практичні сфери застосування ML, DL

№ п/п	Сфери застосування ML	Сфери застосування DL
1	Розпізнавання мови	Прогнозування часових рядів
2	Розпізнавання жестів	Ранжування в пошуковиках
3	Розпізнавання текстів	Виявлення спаму
4	Розпізнавання образів	Категоризація документів
5	Біоінформатичні розрахунки	Біржовий технічний аналіз
6	Медична діагностика	Кредитний скоринг
7	Виявлення шахрайства	Прогнозування втрати клієнтів

Сфера застосування моделей ML і моделей DL знаходяться трохи в інших нішах. Дослідимо, у чому полягають відмінності між машинним і глибоким навчанням. Такі відмінності краще всього прослідкувати у порівнянні. У табл.2.2 приведемо порівняльну характеристику ML та DL.

Таблиця 2.2 – Порівняльна характеристика ML та DL

	Machine Learning	**Deep Learning**
Об’єм даних	Можна використати невеликі обсяги даних	Необхідні великі обсяги даних
Потреба в ресурсах	Не вимагає значних обчислювальних ресурсів	Потребує значних обчислювальних ресурсів
Конструювання ознак	Працює з точно визначеними ознаками (features)	Може самостійно розпізнавати та створювати ознаки
Підхід до навчання	Дискретний (навчання розбивається на окремі кроки)	Задача вирішується «наскрізним методом»
Час навчання	Відносно коротке навчання	Довге навчання

В ML ми як правило використовуємо відносно невеликі дата сети порівняно з дата сетами тими, які використовуються в DL. У зв'язку з цим моделі ML ми можемо створювати на відносно малопотужних комп'ютерах і в цьому їх велика перевага. Беремо звичайний нетбук, навчили швиденько і ось у нас модель є, ми її потім зберегли і можна деплоїти. У моделях DL дуже часто щось толкове зробити не вдасться на малопотужних машинах. Пов'язано це з тим, що робота даних моделей базується на обчисленні коефіцієнтів, на проведенні математичних операцій з натуральними числами. Це накладає додаткові вимоги на процесорний апарат, на процесор. А справа в тому, що процесори з числами з плаваючою точкою працюють не дуже добре. Набагато краще з такими цифрами працюють відеокарти – процесори, які знаходяться на відеокартах. Тому на дуже багатьох сучасних фреймворках реалізована підтримка використання для моделей DL процесорів відеокарт. Звідси витікає у нас і потреба в ресурсах. Якщо ML – машини у нас малопотужні, то DL – до машин виникають додаткові вимоги.

Що стосується роботи з предикторами. Для того, щоб нам навчити модель ML, нам потрібно чітко окреслити предиктори, які ми на неї передаємо. Дуже рідко трапляються моделі, які можуть працювати з пропущеними даними, самостійно ці дані додавати. Нам потрібно перед тим як передати навчальну вибірку на модель ML, нам потрібно спочатку повністю підготувати наш набір даних.

Що стосується моделі DL – модель DL може самостійно розпізнавати та створювати нові ознаки, тобто вона може підлаштовуватися під характер будь-яких предикторів. Ми можемо на неї передати в один момент дуже якісну фотографію, в інший момент – не якісну, вона проковтне і одне і друге, все буде лише залежати від якості роботи самої моделі. Звичайно, якщо ми хочемо отримати найкращу якість роботи нейронної мережі, то і навчальний матеріал для неї також повинен бути якісний.

Звернемо ще увагу на швидкість навчання. Моделі ML навчаються відносно швидко. Бувають випадки, коли комп'ютер навчає модель більше години. Але там був дуже великий дата сет. Там був курс біткоїна до долара протягом декількох останніх років і там були похвилинні показники. Більше мільйону записів було.

Що стосується моделей DL – то там навчання протягом години – це не межа. Ми можемо навчати нейронну протягом декількох діб – таке теж трапляється. І до цього теж потрібно також бути готовими. А звідси витікають вимоги до нашого робочого місця. Якщо від вас вимагають створювати моделі глибокого навчання, то ми повинні вимагати від нашого замовника (роботодавця), щоб він нас забезпечував додатковими обчислювальними можливостями. Це повинні бути сервери, обчислювальні кластери, куди ми будемо заходити під час нашої роботи, задавати алгоритм навчання, запускати його і забувати про цей обчислювальний кластер на декілька діб, поки нам він не пришле або СМС-ку, або електронне повідомлення, що навчання закінчено. І тоді ми можемо дивитися на результат.

То знову ж повертаючись до вимог роботи з даними типами моделей – ресурси, ресурси і ще раз ресурси.

2.2 Дослідження особливостей побудови нейронних мереж

Нейронна мережа – це одна із реалізацій моделей глибокого навчання (рис.2.1). На сьогоднішній день вона провідна. З іншого боку, коли ми говоримо про DL ми в першу чергу маємо на увазі навчання нейронних мереж.

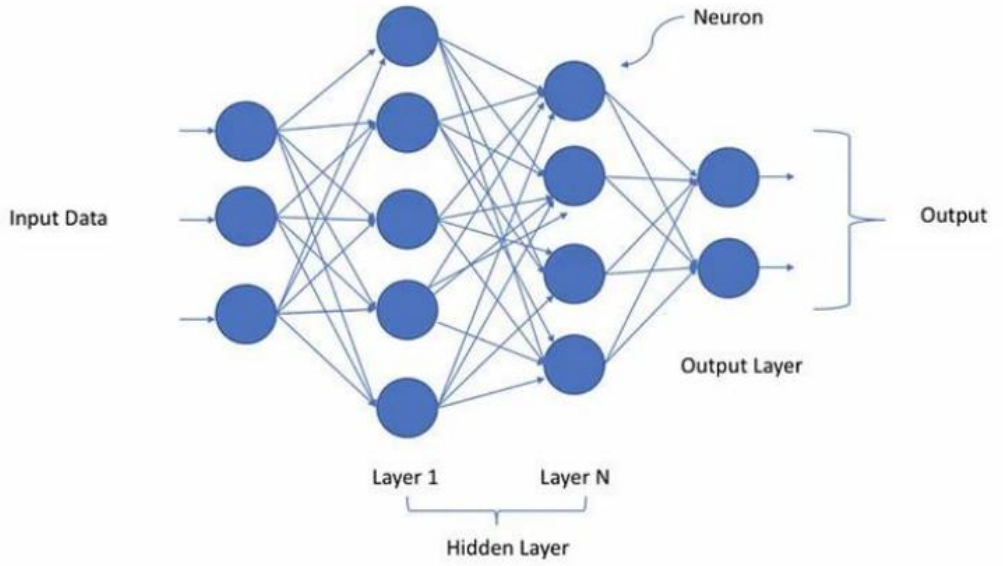


Рисунок 2.1 – Глибокі нейромережі в Deep Learning

Нейронна мережа – це по суті своїй граф. Нейронна мережа представляє собою самий що ні на є стандартний граф, з яким можна працювати в принципі тими ж

самими методами, з якими ми працювали з графами. Вона складається з декількох елементів. До таких елементів відносяться в першу чергу – ноди, тобто вузли – їх ще називають нейрони – це певна комірка у віртуальній пам'яті або на жорсткому диску, куди записується певний цифровий коефіцієнт, який обраховується в результаті навчання і подальшого донавчання моделі. Цей коефіцієнт постійно змінюється, він не статичний, якщо він перестає змінюватися – це дуже погано, тому що це говорить про те, що наша модель перенавчилася і вона не буде далі адаптуватися. Крім того, другою складовою нейронної мережі є так званий зв'язок (connect). З connect-ами трошки складніше. Власне кажучи по показникам цих конектів і визначаються окремі шари нейронної мережі. Тому що конекти в нейронах можуть зв'язувати їх із нейронами сусідніх рівнів або ж з нейронами свого власного рівня. Третя складова – це рівні нейронної мережі. Рівні нейронної мережі – це групи нейронів, які виконують якусь певну функціональну частину. Зокрема, у будь-якій нейронній мережі є три основних типи рівнів. Це input-рівень або вхідний рівень. За вхідним рівнем у нас знаходяться один або декілька (їх може бути дуже багато) прихованих рівнів. Саме на цих прихованих рівнях і відбувається вся магія нейронної мережі. І output-рівень або вихідний рівень.

Вхідний рівень – він приймає деяку вхідну інформацію. Тобто, якщо ми, наприклад, передаємо на нейронну мережу якусь картинку, то нам її потрібно примітизувати до розміру цього самого вхідного рівня. Як це розраховується. Якщо ми передаємо на вхідний рівень цифрову картинку розміром 128 на 128 пікселів, розраховуємо: 128 * 128 = 16384. От у нас на вхідному рівні повинно бути 16384 приймаючих нейрони. Кожен із цих нейронів буде приймати по одному пікселю нашого цифрового зображення. Далі, основна задача вхідного рівня – це прийом зовнішніх даних, які ми передаємо на нейронну мережу.

Наступний рівень – це прихований рівень. Тут у нас відбувається обчислення вхідної інформації. Ця інформація може стискатися або навпаки розтискатися і за рахунок вже існуючих там коефіцієнтів набувати певних змін або проходити певну обробку. І коли у нас вхідна інформація проходить через систему прихованих рівнів вона у нас якраз і набуває тих властивостей, які нам потрібні. В

процесі переходу вона поступово зводиться до вихідного рівня. І цей вихідний рівень – його роль заключається в тому, щоб він нам видав оброблену інформацію. Яким чином він видає інформацію. Знову ж таки залежить від того, що ми хочемо отримати від моделі. Якщо у нас наша модель нейронної мережі займається бінарною класифікацією, де нам потрібно всього навсього передати на неї зображення і отримати відповідь, до якого класу воно відноситься (до класу 0 або до класу 1), то відповідно у нас на виході може бути тільки 2 нейрони. Тобто тут у нас може бути нейронів дуже багато – тисячі, а на вихід у нас буде тільки 2 нейрони, який буде нам передавати 1 або 0, або другий буде передавати одиницю.

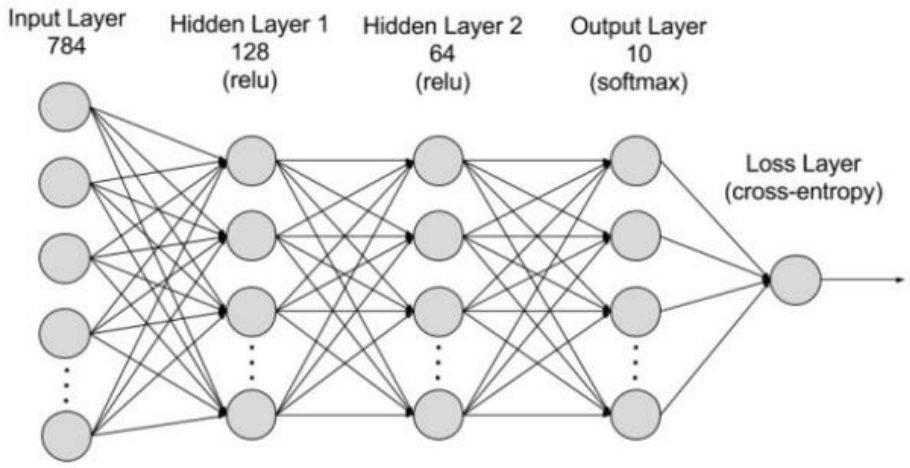


Рисунок 2.2 – Базова структура штучної нейронної мережі

Зовсім інша справа, якщо ми використовуємо нейронну мережу для розпізнавання якоїсь певної класифікації нашого зображення. Тоді у нас повинно бути на виході нейронів принаймні не менше, ніж кількість класів. Тобто, якщо ми класифікуємо всі зображення, що передаємо на нейронну мережу, в межах 10-ти класів, то у нас на вихідному рівні повинно бути мінімум 10 нейронів. Кожен відповідає за відповідь по своєму окремому класу.

І якщо ми використовуємо нашу мережу для обробки перетворення нашого зображення, наприклад, генерації якогось зображення на основі вхідного шуму. У генеративно-змагальних мережах таке робиться. То для цього нам потрібно мати вихідний рівень такого рівня, щоб він нам видав це зображення на вихід. Якщо,

наприклад, ми хочемо отримати зображення на виході 128 на 128 пікселів, вихідний рівень повинен мати 16384 нейронів. Ось така математика попередньо.

Яким чином ми можемо запрограмувати нейронну мережу. Коли це тільки починалося. Коли зв'язки, взаємодії між зв'язками описувалися вручну, це було щось страшне і були кілометри коду, і дуже багато помилок, і дуже не зручно, тому ця тема просувалася досить повільно. Але людині притаманно все зводити до якоїсь певної системи і все автоматизувати. Тому досить швидко для створення нейронних мереж були створені спеціальні фреймворки, спеціальні додаткові класи, пекетджі, за допомогою яких нейронні мережі можна було створювати вже у більш комфортний спосіб.

Таких фреймворків, які використовуються зараз в системах глибокого навчання, дуже багато. Вони є як спеціалізовані, так і широкого змісту, які мають в собі не тільки класи для створення шарів нейронних мереж, але і спеціальні функції підготовки дата сетів, спеціальні функції обробки елементів даних дата сетів, зокрема обробки зображень і т.д. Тобто, якщо беремо один фреймворк, там наприклад, TensorFlow. То ми там отримуємо такий набір інструментарію, що по-перше, там потрібно окремий університет закінчити, щоб цей інструментарій освоїти, а по друге, іноді виникає запитання, а навіщо він там взагалі потрібен, якщо в інших фреймворках він теж є і реалізований теж не погано, а нам розробники відповідають: а навіщо вам до вашого проекту підключати ще додаткові якісь фреймворки, якщо ми вам даємо цілий набір інструментів. От ваші дані, а ми вам даємо повний швейцарський ніж з усіма фішками і ви можете з його допомогою обробити ваші дані, підготувати їх, побудувати нейронну мережу і все, задеплоїти її ще навіть десь.

Виконаємо аналіз найбільш популярних фреймворків.

2.3 Фреймворк TensorFlow

На сьогоднішній день з іншим фреймворком він поділяє перше місце. І інший фреймворк називається Pytorch. TensorFlow завойовує серця програмістів і датасаєнтистів тим, що це розробка не якогось там лівого колективчику, а це

розробка фахівців від корпорації Google. Є у них такий підрозділ, називається Google Brain і він займається розробкою в тому числі і фреймворку, який потім надає для надання вільного використання. Давайте з вами подивимось, де у нас знаходиться сайт цього фреймворка. *TensorFlow.org* – це сайт нашого фреймворку. А ще давайте перейдемо на TensorFlow.PyPI.

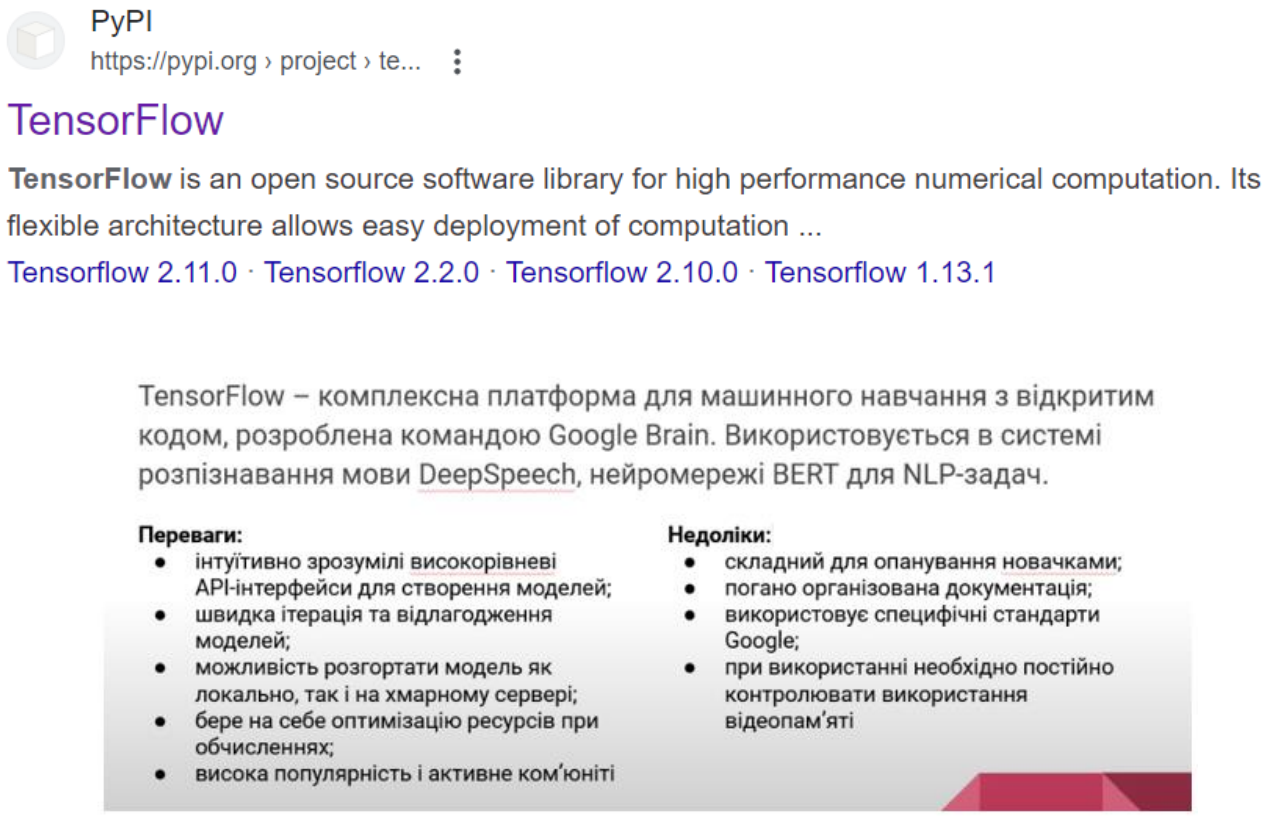


Рисунок 2.3 – Фреймворки для машинного навчання: TensorFlow

Ми можемо проінсталювати даний фреймворк, використовуючи ось цю команду: *pip install tensorflow*

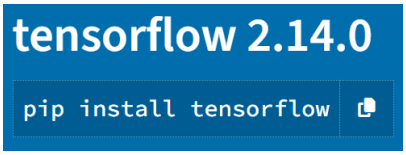


Рисунок 2.4 – Інсталяція фреймворку TensorFlow

На сьогоднішній день у нас актуальна версія 2.13. Тобто версія лінійки 2. Але до цього у нас була версія лінійки 1. І дана версія була актуальна на момент

6.01.21, тобто ну рахуємо десь 2 роки назад. Ось чому, коли ми будемо працювати з даним фреймворком і шукати якісь приклади в Інтернеті ми дуже часто будемо натикатися на розробки, які були розроблені на першій версії. А проблема полягає в тому, що версія 1 і версія 2 мають між собою досить суттєві розбіжності і вони не підтримують зв'язок зверху вниз. Тобто, якщо ми напишемо код на версії 2, а потім спробуємо його запустити у віртуальному середовищі, де у нас стоїть версія 1.15, то з великою ймовірністю він у нас там або не виконається, або буде працювати з помилками. І це потрібно враховувати. Подвійність реалізації даного фреймворку. Тобто у нас є лінійка першої версії і лінійка другої версії.

Хоч і закидають даному фреймворку, що тут не дуже добре пророблена документація, але я особисто з цим погодитися не можу. По перше документація тут багатомовна (рис.2.5).

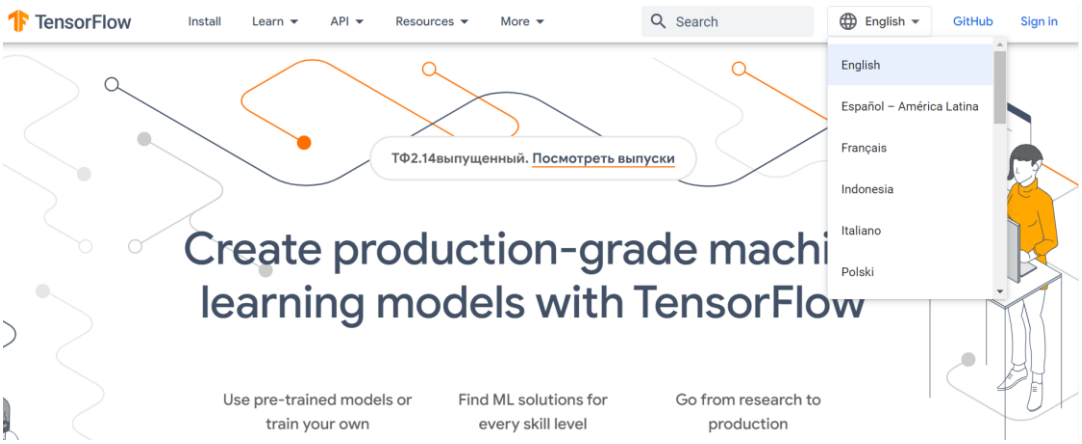


Рисунок 2.5 – Документація TensorFlow

Вже навіть ведеться лінійка документації українською мовою (скоро з'явиться). І що тут у нас знаходиться безпосередньо на нашому сайті, що тут ми можемо знайти. По-перше, ми зразу можемо перейти в установку і в установці надаються самі різні варіанти установки нашого фреймворку TensorFlow (рис.2.6).

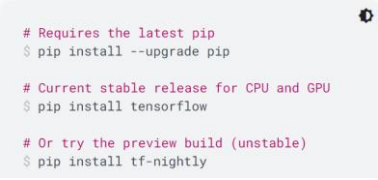


Рисунок 2.6 - Варіанти установки фреймворку TensorFlow

Якщо раніше в попередній версії у нас була окрема гілка для роботи з CPU і окрема гілка для роботи з графічним підпроцесором, то тут зараз все зведено в один варіант **pip install tensorflow** і все залежить від налаштувань нашої системи: чи буде у вас працювати GPU, чи не буде. Це не завжди буває зручно, але тим не менше.

Далі, крім Python, TensorFlow підтримує і інші мови програмування, зокрема Java, C, а також мову програмування Go. Тобто те, що написано за допомогою даного фреймворку, ми можемо зустріти в різних варіантах виконання коду.

У плані вивчення даного фреймворку (рис.2.7).

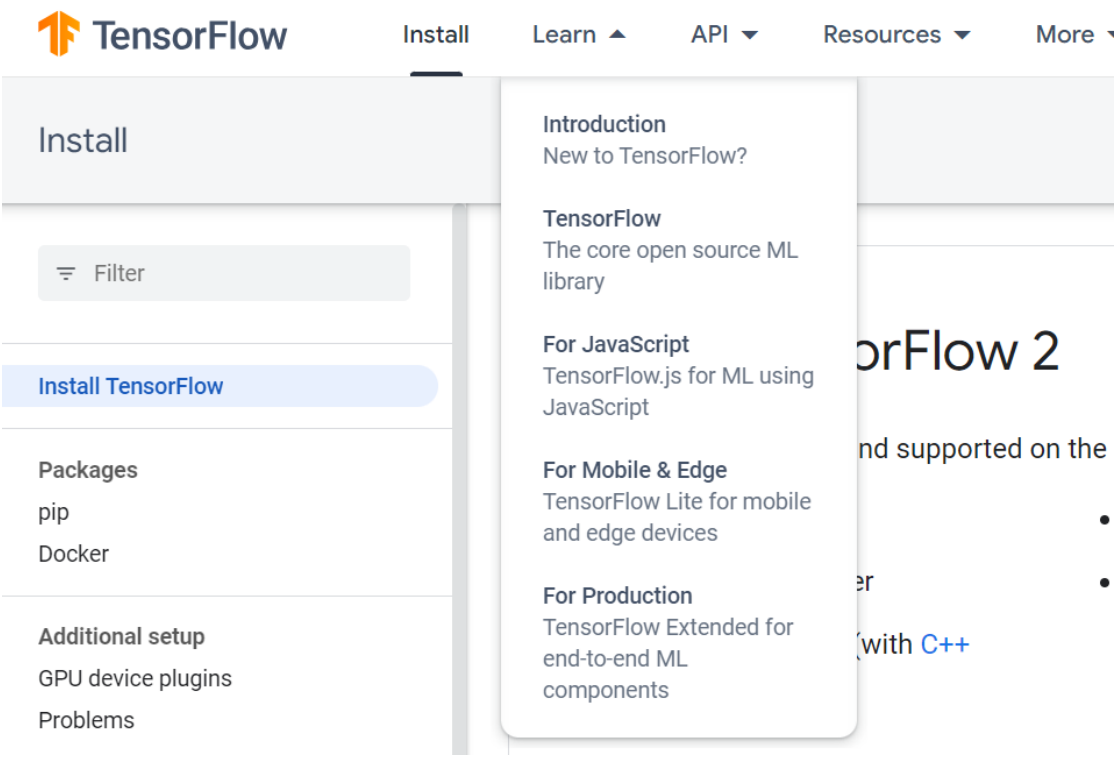
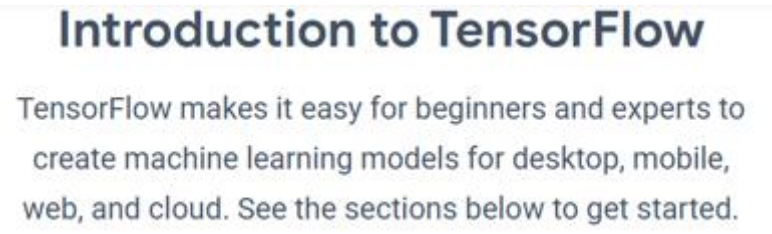


Рисунок 2.7 – Вивчення фреймворку TensorFlow

У нас тут є розділ Введення (рис.2.8).



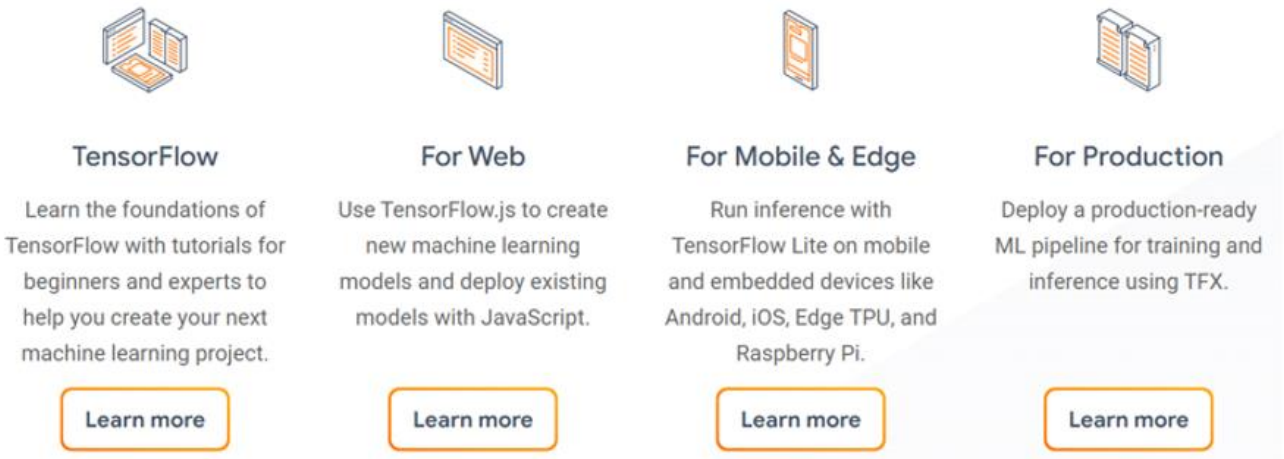


Рисунок 2.8 – Введення в TensorFlow

Дуже поступовий і дуже продуктивний вхід з прикладами (рис.2.9).

Рисунок 2.9 – Вхід з прикладами

Переходимо:

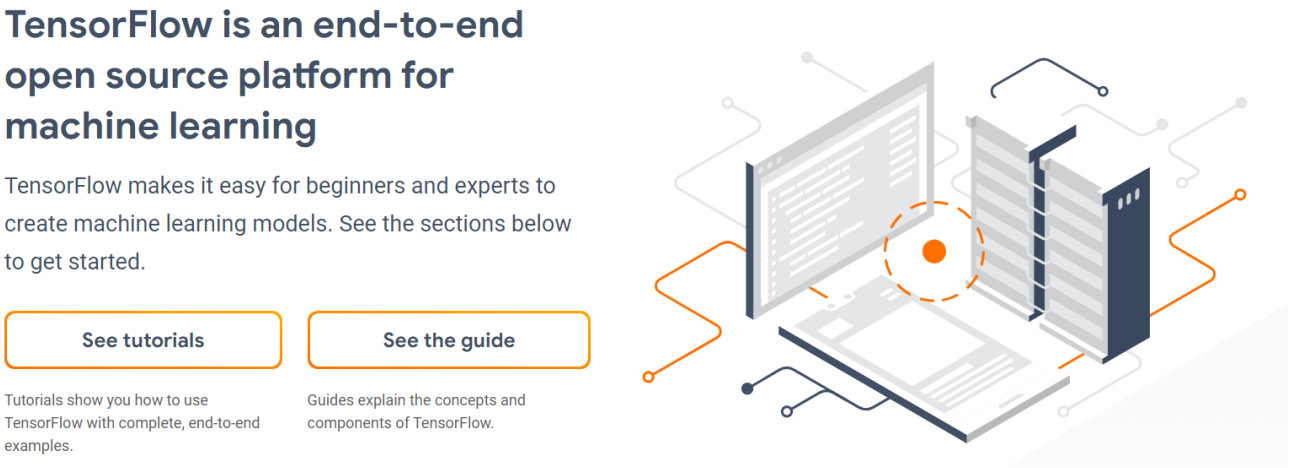


Рисунок 2.10 - Посилання

І зразу у нас посилання: дивитися керівництво, дивитися навчальний посібник. Переходимо до навчального посібнику (рис.2.11).

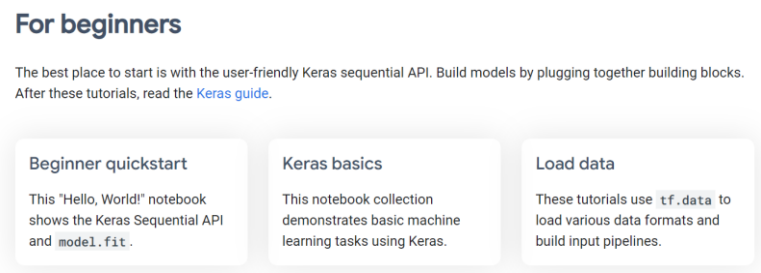


Рисунок 2.11 – Перегляд навчальних посібників з TensorFlow

У нас тут йде відеолекція. І по окремим елементам у нас тут йде ще код. При чому код на декількох мовах програмування, зокрема на Python і крім усього цього ми можемо зразу цей код запустити в Google Colab. Або завантажити собі безпосередньо на комп'ютер (рис.2.12).

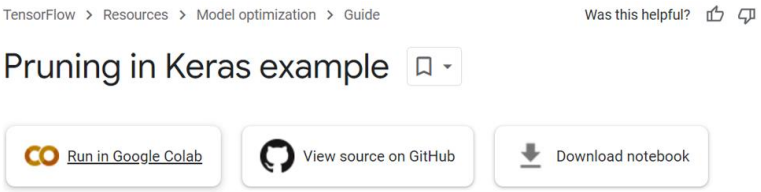


Рисунок 2.12 – Запуск коду в середовищі Google Colab

Підрозділи для різних мов (рис.2.13): для JavaScript, мобільні пристрої і т.д.

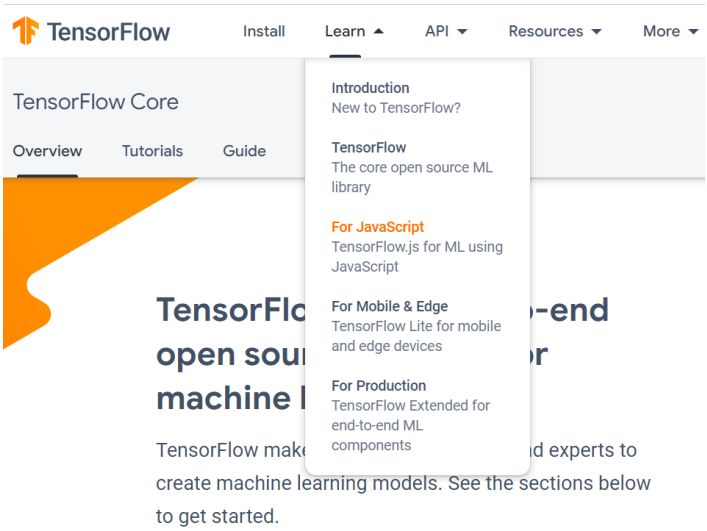


Рисунок 2.13 – Підрозділи для різних мов

Перейдемо далі на ресурси (рис.2.14).

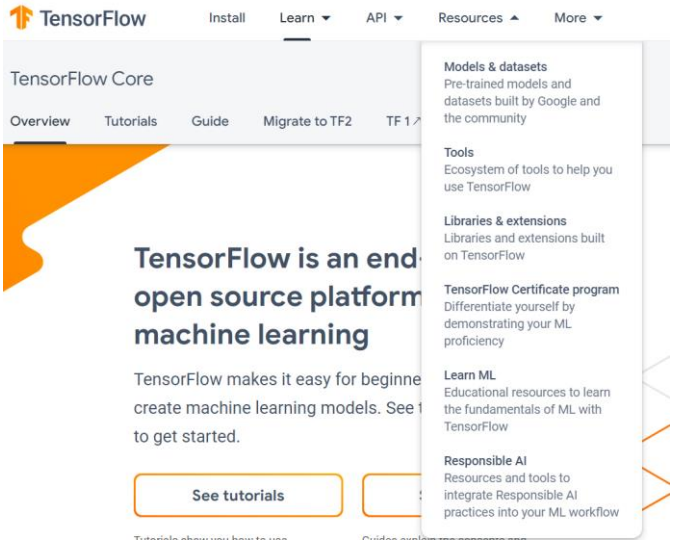


Рисунок 2.14 – Ресурси TensorFlow

В Ресурсах у нас виходи на статті форуму, де ми можемо в принципі отримати відповідь на будь-яке наше запитання. Як бачимо, в плані підтримки даний фреймворк досить непогано організований. Визначимо його особливості.

По-перше, він досить простий у використанні. Він інтуїтивно зрозумілий. Працює по принципу конструктора лєго, тобто тут вже є якісь певні заготовки, просто беремо ці заготовки і конструюємо з них нашу нейронну мережу. Працювати з ним дуже приємно, дуже легко. Ми можемо за допомогою даного фреймворку розгортати наші моделі як у себе на комп'ютері так і в хмарних сервісах. Даний фреймворк дуже добре масштабує наші комп'ютерні ресурси, тобто якщо у нашої машини повна сумісність з вимогами даного фреймворку по апаратній частині, то він буде знати, коли підключати процесор, коли підключати відеокарту, тобто дуже полегшує роботу при створенні нейронних мереж. Звичайно є у нього й певні недоліки. Якщо ми захочемо більш тонко налаштувати, наприклад, таку функцію як fit() – тобто навчання нашої нейронної мережі, то нам у TensorFlow буде це дуже важко зробити, тому що там функція fit() прив'язана до класу нейронної мережі, яку ми будемо використовувати. Там декілька класів шаблонів і кожна функція fit() в принципі написана під цей шаблон. Теж саме стосується функції predict, тобто ось цей підхід конструктора

лєго - він має як свої переваги так і дає свої певні недоліки. Крім того, як на мене, є негативом у даному фреймворку те, що він дуже жорстко прив'язаний до стандартів Google. Тобто те, що реалізується в Google на рівні його стандартів в усіх його сервісах – ми це перескочити ніяк не зможемо. Тому, маючи дуже багато переваг, тим не менше у даного фреймворку є свої недоліки, як і у всіх в принципі, і їх потрібно враховувати при роботі з ним.

2.4 Фреймворк Keras

На сьогоднішній день є надбудовою над фреймворком TensorFlow. Він ще більш простий. Рекомендується для початківців і саме головне він існує в цілих двох іпостасях.

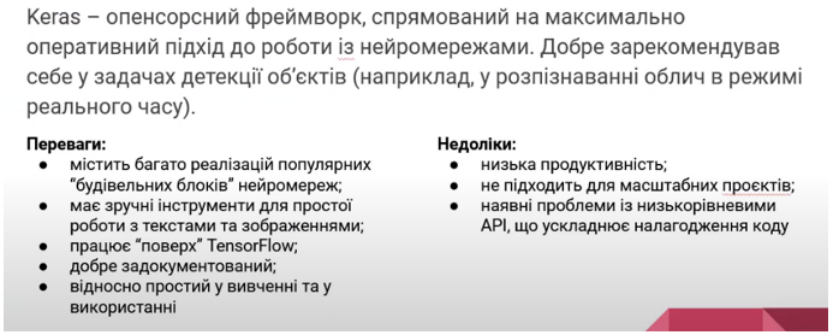
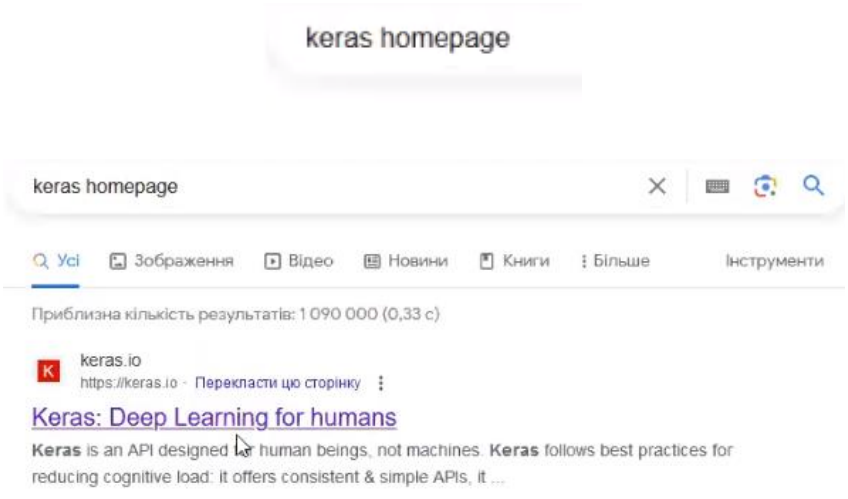


Рисунок 2.15 – Фреймворки для машинного навчання: Keras

Спочатку з'явилася група програмістів, які побачили TensorFlow, сказали так, це круто. Давайте ми візьмемо за основу TensorFlow і розробимо на його основі свій фреймворк і назвемо його Keras. І почали розробляти.



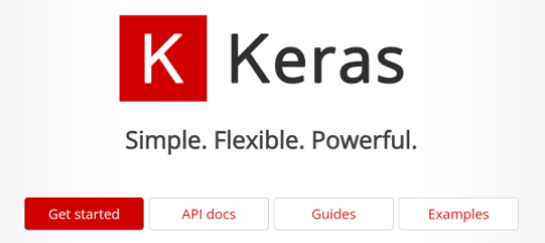


Рисунок 2.16 – Вивчення фреймворку Keras

Google на цю справу подивився і каже: добре, якщо ви розробляєте на основі нашого фреймворку свій фреймворк, то ми тоді зробимо наступне. Ваш фреймворк поширюється по ліцензії GNU GPL (загальна публічна ліцензія на вільне ПЗ), тобто абсолютно безкоштовно в тому числі і вихідний код. От ми візьмемо цей вихідний код і інтегруємо його в наш TensorFlow. Так вони і зробили. Вони (програмісти Google Brain) взяли цей фреймворк, інтегрували його в якості пекетжа в TensorFlow. Тому дуже часто Keras не використовується як фреймворк самостійний, а використовується як пекетж фреймворку TensorFlow, тобто ми імпортуємо TensorFlow.keras, а потім з нього - шаблони, моделі, шаблони шарів і код пишеться практично один в один з тим кодом, який би ми писали, використовуючи тільки фреймворк Keras. Можемо зайти на сайт *keras.io* (у даному випадку натиснути *Examples*). Тут досить непогано організована документація, можна подивитися приклади, зокрема по роботі з відео, по обробці зображень і т.д. В основному ним користуються в рамках TensorFlow.

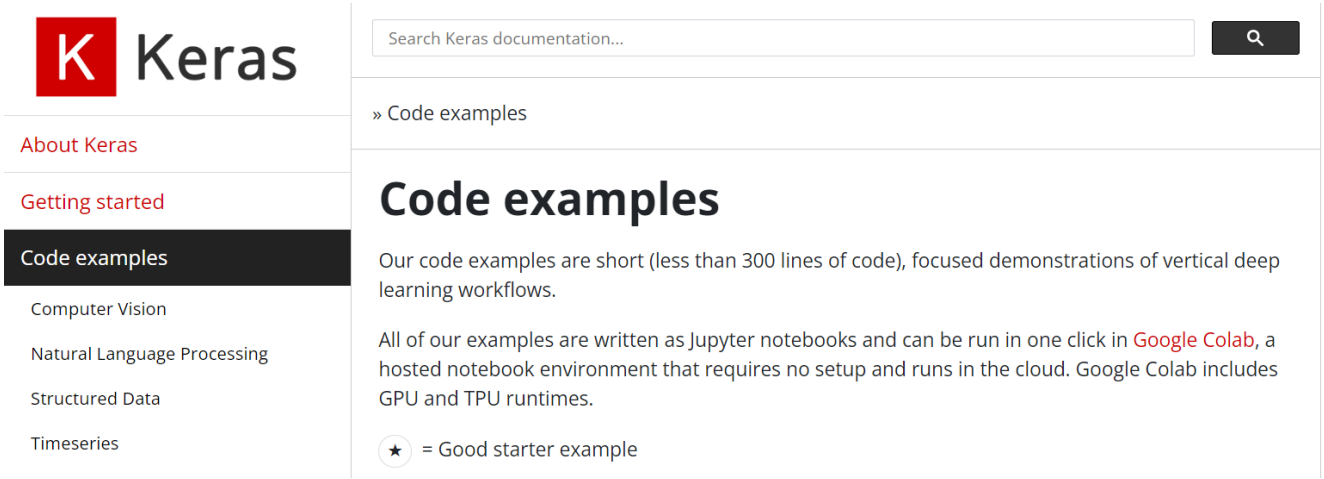


Рисунок 2.17 – Code examples

У загалому сам по собі фреймворк Keras як окремий фреймворк має продуктивність меншу, ніж фреймворк TensorFlow. І це зрозуміло, тому що це надбудова. Він не підходить для великих проектів. Велику нейронну мережу, потужну нейронну мережу, наприклад, ChatGPT ми на ньому не зробимо. Але як навчальний варіант він в принципі досить зручний, але оскільки він використовується як пекетж TensorFlow, то вивчати його окремо самостійно в принципі сенсу особливого немає. Але такий проєкт існує. Ми маємо про це знати, щоб бути грамотними фахівцями.

TensorFlow ділить перше місце з фреймворком Pytorch.

2.5 Фреймворк Pytorch

Фреймворк Pytorch розроблений фахівцями від Facebook. А це вже багато про що говорить. І ось це вже дійсно самостійна цікава річ, яку використовує друга половина датасаєнтистів, які займаються побудовою нейронних мереж. Тобто можна всіх датасаєнтистів поділити на дві групи: 40% використовують TensorFlow, 40% використовують Pytorch, всі інші експериментують.

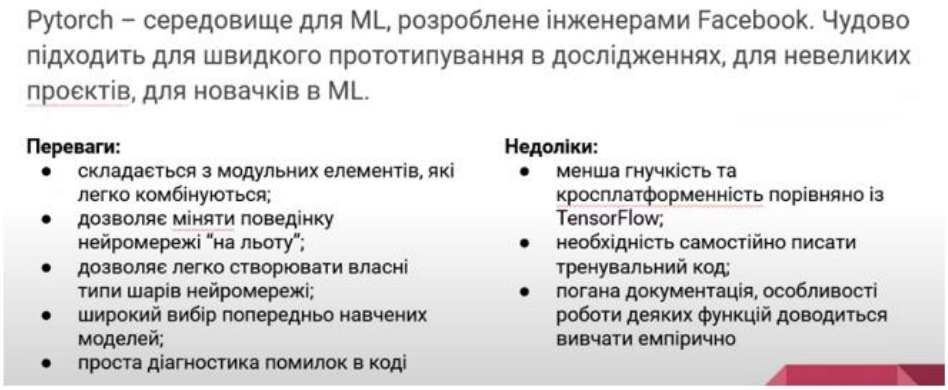


Рисунок 2.18 – Фреймворки для машинного навчання: Pytorch

Визначимо особливості фреймворку Pytorch та його специфіку. Він дуже гнучкий. Це не конструктор лєго. Так, там є шаблони функцій активації, там є шаблони нейронних мереж і шарів нейронних мереж, але в цілому даний фреймворк передбачає написання більшого об’єму коду для реалізації однієї і тієї ж задачі, яку ми б реалізовували з меншим розміром коду, використовуючи TensorFlow.

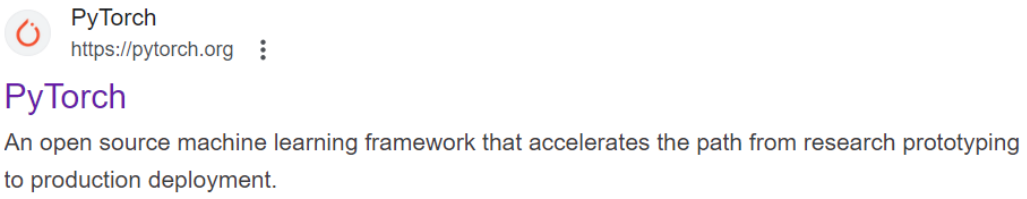


Рисунок 2.19 – Сайт Pytorch

У даного фреймворку сайт кращий, ніж у досліджених попередніх фреймворках.

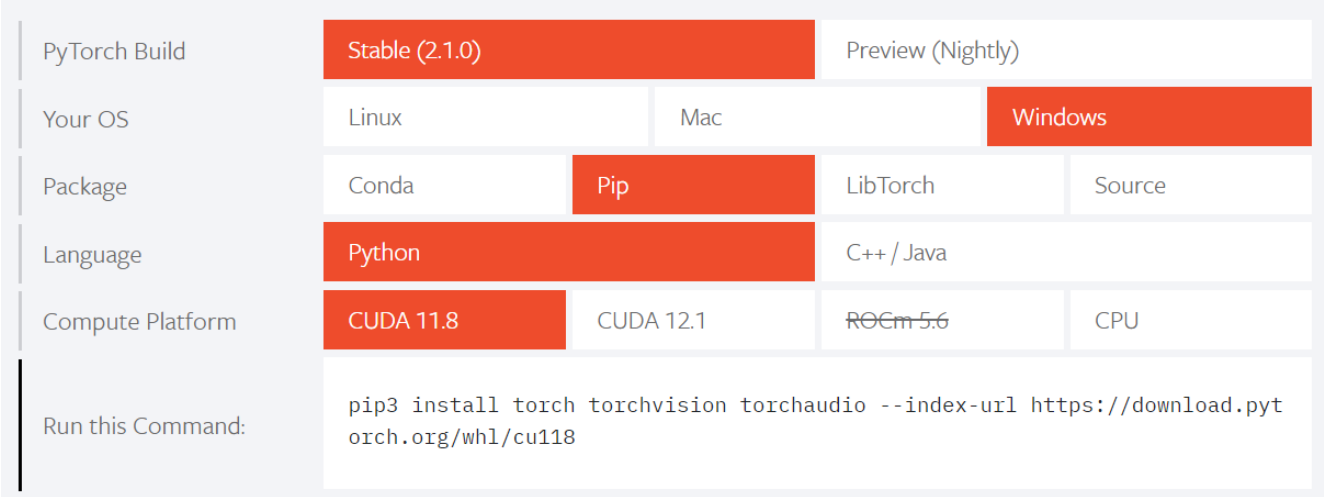


Рисунок 2.20 – Сітка Pytorch

Перше, що ми обираємо – це версію даного фреймворку (або стабільно, або експериментально). Потім ми вибираємо операційну систему: Linux, Mac або Windows. Потім вибираємо менеджер пакетів: Source – вихідні пакети, тобто там де вам потрібно його збирати. Вибираємо pip. Вибираєте мову програмування. Pytorch може встановлюватися і на Python, і на C++/Java з бібліотеки LibTorch. Ми вибираємо Python і далі нам потрібно вибрати – це те, що ми будемо використовувати в якості обчислювальної системи. Якщо ми будемо використовувати тільки процесор, ми вибираємо CPU і внизу інструкція по встановленню даного фреймворку: *pip install torch totchvision torchaudio*. Якщо ж ми будемо використовувати графічний сопроцесор, то нам потрібно буде попередньо встановити підсистему, яка називається nvideo cuda. І все що нам потрібно зробити на етапі встановлення фреймворку – це вибрати версію CUDA 11.7 або CUDA 11.8. Натиснули і отримали в самому низу рядок, який ми

копіюємо, встановлюємо в командний рядок у нашому віртуальному оточенні, встановлюємо необхідну версію. І якщо ми це поставимо, то гарантовано наш фреймворк Pytorch буде використовувати ресурси нашої відеокарти. Прекрасна річ у всіх відносинах.

Тепер документація і ресурси (рис.2.21).

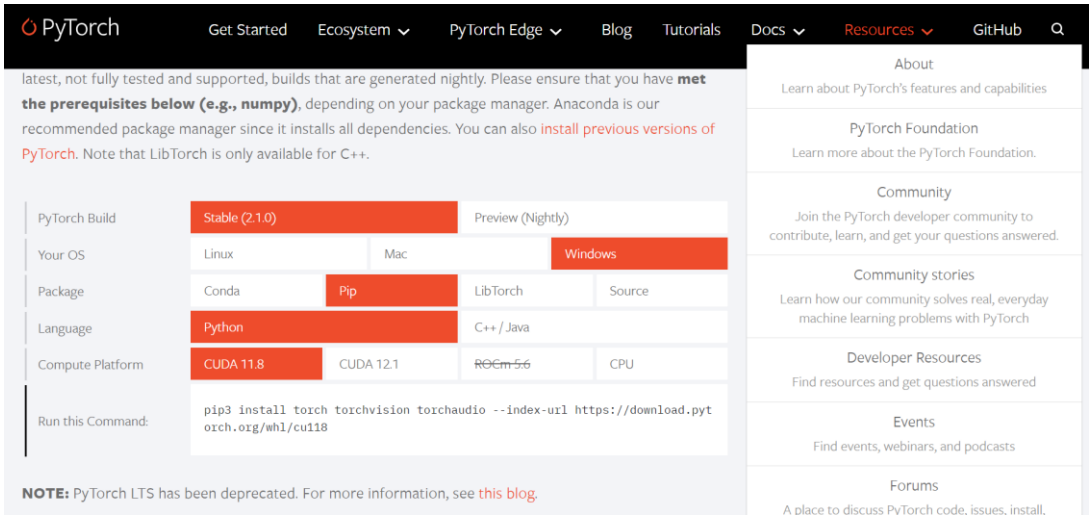


Рисунок 2.21 - Документація і ресурси

Tutorials тут взагалі чудові (рис.2.22).

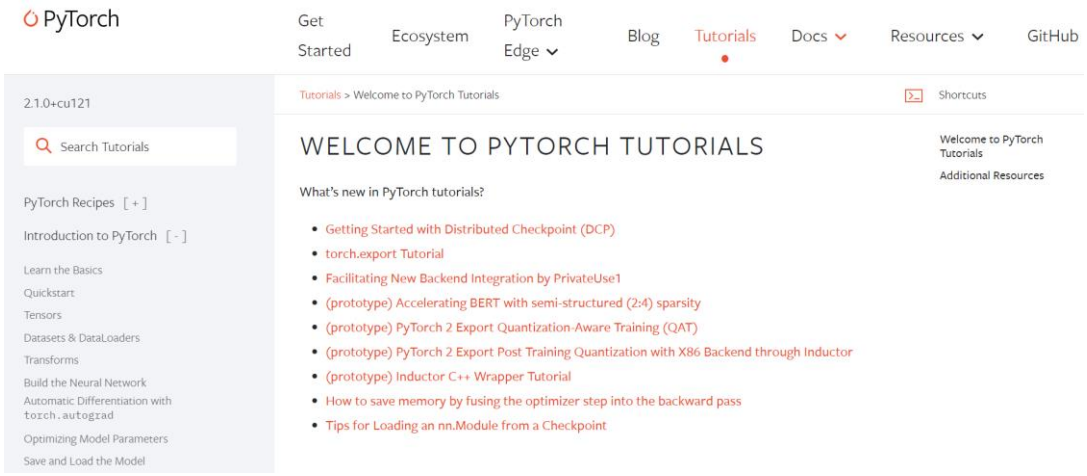


Рисунок 2.22 - Tutorials

Можна вибрати All (рис.2.23) - всі підручники по фреймворку Pytorch. Можна вибрати Image/Video – обробка відео і обробка картинок. Можна вибирати. Маємо автоматичне сортування всіх навчальних посібників по даному напрямку.

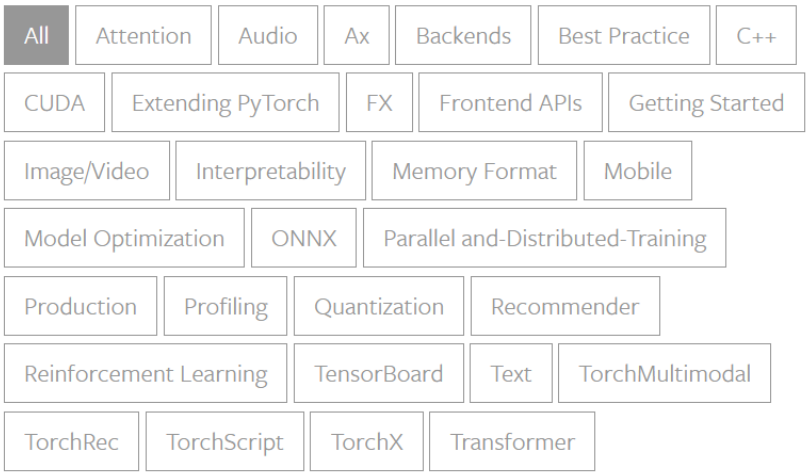
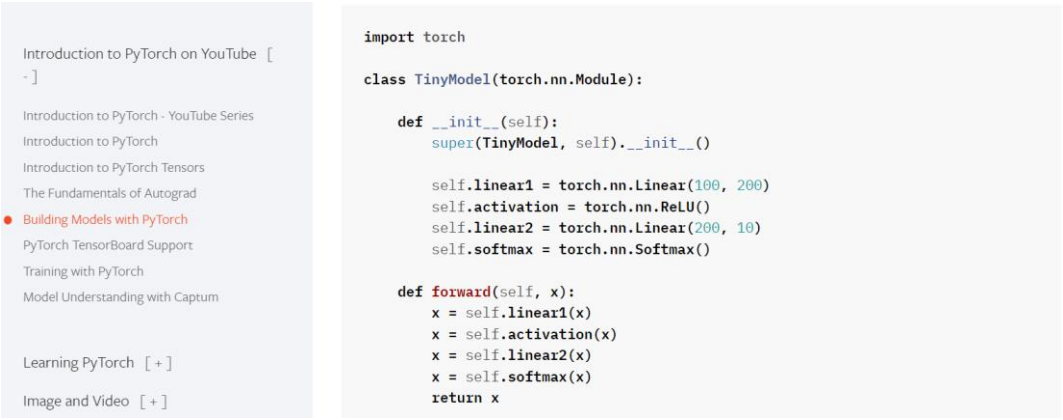


Рисунок 2.23 – Навчальні посібники з різних напрямків



Заходимо (з картинками, з кодом) і вчимося будувати моделі, які будуть нам розпізнавати рукописний текст.. Код можемо завантажувати у вигляді ноутбука під Jupyter (все розписано).

У вкладці *Документація* по кожному із пекетжів Pytorch можемо знайти документацію (рис.2.24).

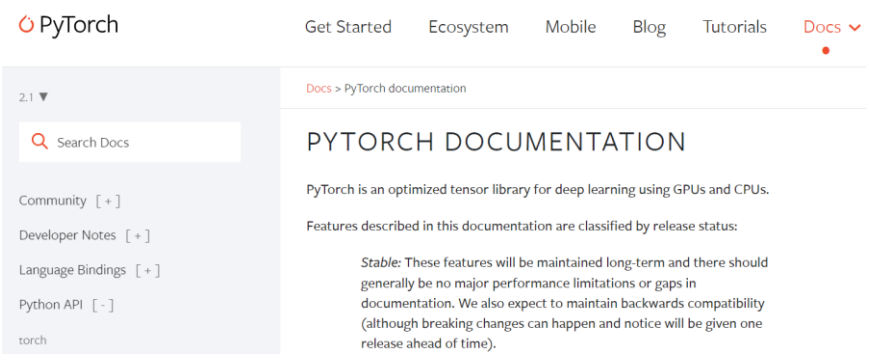


Рисунок 2.24 – Документація Pytorch

І нарешті *Ресурси* (рис.2.25).

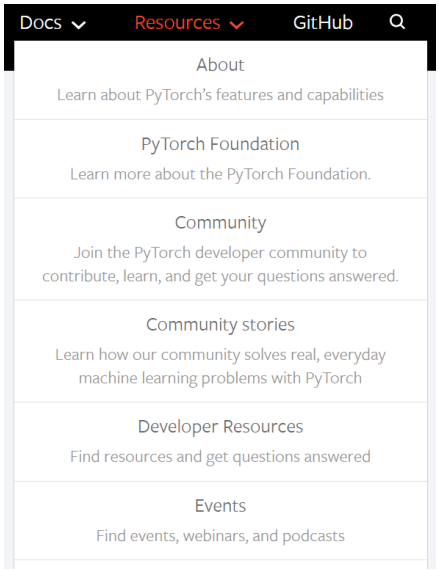


Рисунок 2.25 – Ресурси Pytorch

Є вихід на моделі. Тобто за допомогою даного фреймворку вже створені моделі. Можна експериментувати.

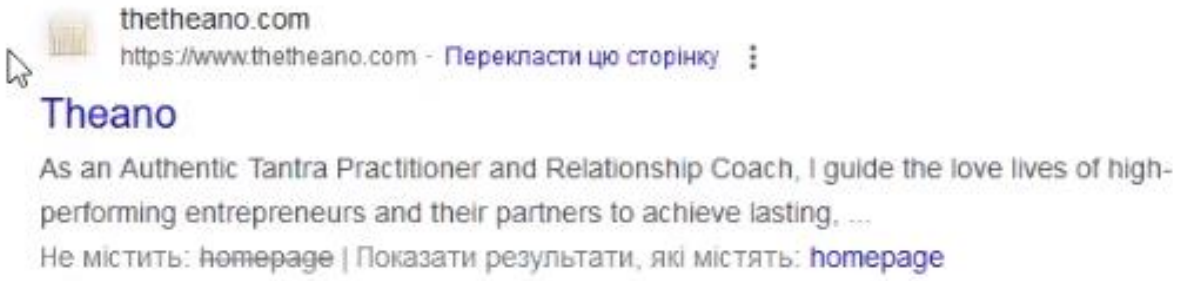
Визначимо особливості роботи з фреймворком Pytorch. У той же час, якщо ми захочемо віддати свої серця даному фреймворку, ми маємо розуміти, що це не конструктор лєго, тут дуже багато речей доведеться створювати вручну, а значить доведеться вникати, експериметнувати. У посібниках, підручниках та документації вже будуть готові блоки коду, але нам потрібно буде вміти цей код читати, аналізувати, щоб він підходив конкретно під наші потреби. Нам доведеться самостійно писати функцію fit(), доведеться самостійно писати функцію predict(), нам доведеться самостійно писати дуже багато речей, які в інших фреймворках вже реалізовані. Але ми їх можемо написати так, що вони будуть максимально тонко підходити під нашу задачу. От в цьому основна перевага і основний недолік даного фреймворку. Фреймворк хороший, але з ним потрібно розбиратися.

2.6 Фреймворк Theano

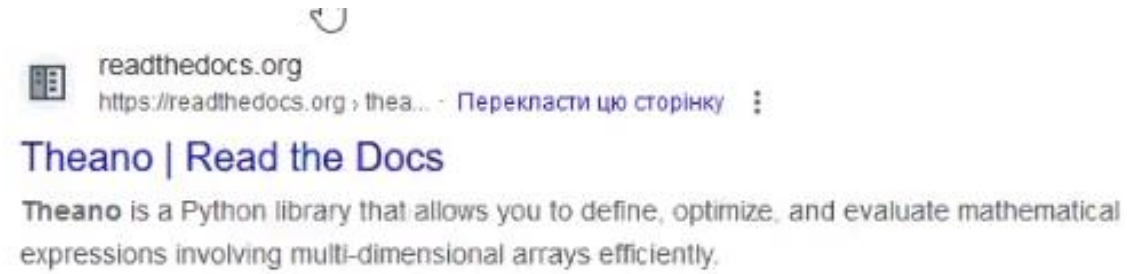
Менш відомий, але теж до певної міри популярний – це платформа Theano. Це платформа штучного інтелекту, яка забезпечує створення нейронних мереж

максимальної точності і які потребують у той же час високі обчислювальні потужності. Вона створена в Монреальському університеті, створена вченими, створена для вчених і дуже часто застосовується у створенні моделей машинного навчання для моделей нейронних мереж.

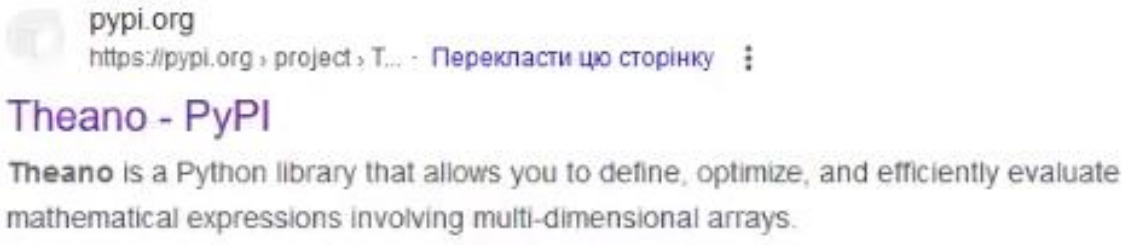
Офіційний сайт даного фреймворку – *thetheano.com*



Знайти по даному фреймворку документацію – задача ще та. Єдине, що нас може порадувати – це ось цей ресурс (рис.2.26).



Можна знайти на GitHub і можна знайти на pip.



Посилання на сайт Theano. Сайт називається DL NET. Станом на зараз цей сайт не запускається. Можливо там на брандмауері стоїть фільтрація пакетів з певних напрямків і наші пакети вони фільтрують.

Можна ще яким чином перевірити – запустити VPN, ось такий, наприклад:

І спробувати завантажити теж саме тільки з VPN. Можливо з американського VPN воно і завантажиться. Інформацію про Theano можна знайти в Інтернеті. Але, оскільки даний фреймоврк досить популярний в наукових колах, можна самостійно по ньому щось подивитися.

Провівши вищевказаний аналіз, зазначимо, що ML і DL на найближче майбутнє як мінімум є перспективними напрямками.

Перспективні напрямки застосування ML та DL:

- створення систем автономного управління бойовим транспортом;
- створення інноваційних систем моніторингу та захисту житла, здатних запобігати загрозі проникнення;
- розвиток «Інтернету речей» - більш глибоке впровадження штучного інтелекту у роботу побутової техніки;
- створення більш реалістичних діпфейків – та, водночас, засобів детекції і протидії цим діпфейкам;
- розвиток рекомендаційних систем, створення максимально персоналізованої реклами.

3 РОЗРОБКА НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ РУКОПИСНИХ ЦИФР

3.1 Застосування бібліотек Warnings, TensorFlow, Matplotlib для розробки нейронної мережі

Розглянемо, яким чином в коді пишуться нейронні мережі. Напишемо нейронну мережу за допомогою комп'ютерного коду, визначимо основні особливості цієї мережі.

На сьогоднішній день нейронні мережі пишуть з використанням спеціальних бібліотек, спеціальних фреймворків. Якщо ми програмуємо нейронну мережу з використанням середовища Jupyter Notebook, то нам потрібно в першу чергу подбати про те, щоб у нас не висвічувалося дуже багато мусора. Тому що, коли ми працюємо з нейронними мережами, у нас постійно в Jupyter Notebook надходять різного роду повідомлення, які малоінформативні і на них просто потрібно не зважати. Тому бажано в подальшому використовувати ось ці дві директиви: імпортувати бібліотеку warnings, а потім зразу писати warnings.filterwarnings("ignore"). Ми принаймні цього бачити не будемо.

```
import sys
print("version:", sys.version)
import warnings
warnings.filterwarnings("ignore")
```

```
version: 3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
```

Бібліотека warnings в Python використовується для управління та виведення попереджень (warnings). Попередження – це повідомлення, яке інформує розробника про потенційні проблеми або небезпеки в коді, але не призводить до припинення виконання програм. Бібліотека warnings надає інструменти для управління та пригнічення попереджень. Розробники можуть пригнічувати попередження за допомогою модуля warnings, щоб вони не виводилися на екран або не логувалися.

Тепер спеціалізовані бібліотеки. Будемо використовувати бібліотеку TensorFlow від Google і якщо сказати точніше - пекетж Keras. Keras – окремий проєкт, але крім цього Keras включений в TensorFlow як пекетж і на сьогоднішній

день цей пекетж досить популярний. Тож ми скористаємося цим пекетджем сьогодні, щоб запрограмувати нашу нейронну мережу. Імпортувати ми будемо по-перше датасет, який ми у подальшому будемо використовувати для роботи нашої нейронної мережі для її навчання. Датасет цей називається MNIST – це датасет рукописних символів - цифр – це піксельні зображення цифр, які написані від руки.

MNIST використовується в галузі машинного навчання та комп'ютерного бачення для завдань розпізнавання рукописних цифр. Датасет MNIST містить 60000 зображень рукописних цифр у навчальному наборі та 10000 зображень у тестовому наборі. Кожне зображення має розмірність 28x28 пікселів і представлене у відтінках сірого. Кожне зображення містить одну з десяти арабських цифр (від 0 до 9), яку потрібно розпізнати за допомогою алгоритмів машинного навчання.

MNIST є популярним датасетом в галузі навчання нейронних мереж та класифікації зображень, оскільки він є відносно невеликим та легко доступним. Багато навчальних прикладів та бібліотек для машинного навчання вже мають підтримку для датасету MNIST, що робить його зручним для використання в експериментах та навчальних завданнях.

Далі, імпортуємо модель нейронної мережі з пекетжа TensorFlow.keras.models ми імпортуємо модель Sequential – це сама проста нейронна мережа однонаправлена. Це означає, що всі нейрони в цій нейронній мережі мають зв'язки тільки з нейронами на сусідніх рівнях, між собою вони зв'язків ніяких не мають. Використаємо саму просту нейронну мережу клас якої називається Sequential.

Модель Sequential у контексті бібліотеки Keras, яка входить до складу бібліотеки TensorFlow, представляє собою послідовну модель нейронної мережі. Sequential – це спрощений спосіб створення та навчання нейронних мереж для багатьох завдань машинного навчання, особливо для послідовних задач, де шари нейронів розташовані один за одним в порядку виконання. Ця нейронна мережа буде складатися з певної кількості шарів. Для шарів в TensorFlow є також

шаблони. Ми сьогодні будемо використовувати шаблон Dense – повнозв'язний так би мовити нейронний шар.

Шаблон Dense у нейронних мережах відноситься до повнозв'язного шару (англ. Dense layer), який є одним із основних типів шарів у нейронних мережах. Шар Dense – це шар, у якому кожен нейрон пов'язаний з кожним нейроном попереднього та наступного шару. Шари Dense використовуються для різних завдань, включаючи задачі класифікації, регресії та інші завдання машинного навчання.

Для нормальної роботи нейронної мережі нам ще знадобляться деякі елементи. Нам знадобиться оптимізатор (Adam) – популярний алгоритм оптимізації для навчання нейронних мереж. Adam є досить ефективним оптимізатором для нейронних мереж і зазвичай допомагає прискорити навчання та покращити загальну продуктивність моделей. Він є частиною багатьох бібліотек для глибокого навчання, таких як TensorFlow та Keras, і швидко використовується у практиці машинного навчання.

Деякі бібліотеки для програмування нейронних мереж пропонують додаткові набори функцій не пов'язаних безпосередньо з нейронними мережами. Зокрема у TensorFlow є ціла купа функцій, які призначені зокрема для обробки піксельних зображень. Ми можемо робити їх трансформацію, зсувати, робити zoom по середині, по краях, перевертати, змінювати яскравість, будь-які характеристики растрового зображення. Все це у нас знаходиться в пекетжі preprocessing і в окремому підпекетжі, який називається image – тут знаходяться всі необхідні функції для роботи з зображенням. Звичайно ми тут будемо використовувати matplotlib і будемо використовувати бібліотеку Pillow – це бібліотека для роботи з зображеннями. Вона нам знадобиться там на певному етапі.

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import utils
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
%matplotlib inline
```

Бібліотека Matplotlib – бібліотека, яка використовується для створення візуалізації даних та графіків. Matplotlib використовується у багатьох галузях, включаючи науку, інженерію, аналіз даних, машинне навчання, фізику, біологію та інші області для візуалізації та аналізу даних. Вона є потужним інструментом для створення візуальних представлень даних і спрощення комунікації результатів досліджень та аналізу.

Pillow дуже корисна для веб-розробників, обробників зображень, а також для автоматизації завдань, пов'язаних з обробкою зображень у Python.

3.2 Програмування нейронної мережі

Запрограмуємо нейронну мережу. Спочатку нам потрібно створити екземпляр нашої моделі. От ми її робимо, присвоюючи їй клас *Sequential()* – той клас Sequential(), який ми імпортували. А потім ми з нею починаємо працювати шляхом нарощування її шарів. Створимо нейронну мережу, яка буде складатися з 5-ти нейронів. Це будуть два нейрони зовнішніх, два нейрони прихованого рівня і один нейрон вихідного рівня. Дивимося, яким чином ми це робимо:

```
model = Sequential()
model.add(Dense(2, input_dim = 2, use_bias=False))
model.add(Dense(1, use_bias=False))
```

Ми додаємо рівні до нашої моделі за допомогою функції *add*. І коли ми додаємо перший рівень, ми одночасно додаємо два рівні, тобто коли ми задаємо перший рівень, ми одночасно задаємо input-рівень і перший прихований, який знаходиться під ним. І у такий спосіб ми отримуємо наступну нейронну мережу.

Input_dim=2 – це у нас нейрони вхідного рівня. А перша 2 – це нейрони першого прихованого рівня. Таким чином ми за допомогою першої функції *add* додаємо до нашої нейронної мережі одразу два шари. Тепер. Додаємо ще один шар – вихідний – він у нас буде знаходитися з самого права (один). Ми його знову ж таки додаємо за допомогою функції *add*. Звернемо увагу, як працює функція *add*. Коли ми її використовуємо, перший параметр, який ми повинні їй передати –

це клас шару нашої нейронної мережі. В нашому випадку ми імпортували шар *Dense* і от ми два рази додаємо шари *Dense* і ще один шар *Dense*, а потім вже всередині параметрів даної функції ми вказуємо параметри для самого *Dense*. Тобто перший і другий і всі подальші *add,* які ми використовуємо – вказуємо тут їх параметри. В даному випадку ми сказали, що у нас буде ще один шар, який складається з нейрону. Між цими нейронами формуються наступні зв'язки. Вони зв'язуються один з одним наступним чином: перший нейрон вхідного рівня з'єднується з двома нейронами першого приховано рівня; другий нейрон вхідного рівня з'єднується з двома нейронами першого приховано рівня; два нейрони першого прихованого рівня з'єднуються з одним вихідним нейроном.

Щоб знати, що саме такі зв'язки формуються, це закладено в моделі *Sequential().* Тобто це витікає від типу моделі, яку ми використовую. Тобто ми повертаємося до особливостей фреймворку *TensorFlow,* який виступає в даному випадку як конструктор лєго. Він нам вже пропонує готову модель *Sequential,* готові шаблони шарів нашої моделі (*Dense*), ми їх беремо і використовуємо.

Отримаємо вихідні дані нашої моделі, тобто її *summary* (короткий зміст). Вони вже знаходяться у нашому об'єкті model. Ми їх викликаємо за допомогою функції summary(). І ось, що ми отримали, що нам наша функція summary() за нашу мережу.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 4

 dense_1 (Dense)             (None, 1)                 2

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

А вона нам звітується про нашу нейронну мережу не на рівні окремих шарів, а на рівні зв'язків. Ось що вона говорить: рівень Dense має param, тобто кількість зв'язків - 4. Перший нейрон зв'язується з двома нейронами сусіднього рівня, тобто 1 і 2. І другий нейрон теж зв'язується з двома нейронами сусіднього рівня, тобто 3,4. Ось звідси у нас витікає цифра 4. Потім йде наступний шар dense_1 – він має у нас тільки два зв'язки (1 і 2), які підключаються до вихідного одного нейрону справа). Тобто в даному випадку наша модель сприймає як шари не безпосередньо наші нейрони, а перемички між ними. Тому що саме через ці перемички відбувається обмін даними і у функціональному відношенні вони у нас у даному конкретному випадку самі цінні.

У наших нейронах знаходяться якісь певні коефіцієнти. Як же ці коефіцієнти змінюються і змінюють інформацію, яка проходить через нашу нейронну мережу. Для того, щоб отримати коефіцієнти від нашої нейронної мережі, їх ще називають *ваги*, нам потрібно створити змінну і з нашої моделі через функцію get.weights (передати) параметр. Ваги нашої моделі:

```
weights = model.get_weights()
print(weights)
```

```
[array([[ 0.24123538, -0.37492055],
       [ 0.16657782, -1.1274049 ]], dtype=float32), array([[-0.70594555],
       [ 0.8774115 ]], dtype=float32)]
```

Ці коефіцієнти знаходяться в наших нейронах (6 коефіцієнтів). Вони знаходяться у цих самих зв'язках. Це звичайні флотові значення – звичайні натуральні числа. Більше того, вони рандомні. У даному конкретному випадку вони рандомні. Це означає, що коли ми використали ось ці шаблони *dense* для створення наших шарів, то ця функція *dense* рандомно задала для нашої мережі випадкові показники. Далі, коли ми будемо навчати, а потім використовувати нашу нейронну мережу, ці показники звичайно будуть змінюватися. А зараз вони ось такі. Тому що вони не можуть мати пусте значення, вони не можуть мати значення NaN. Тому що ці коефіцієнти приймають участь в обчисленнях і якщо вони NaN, то це буде множення/ділення на 0 з усіма витікаючими наслідками, тобто наша мережа не буде навчатися і не буде змінюватися.

Ці коефіцієнти ми можемо зразу змінити у такий спосіб як ми захочемо. Ми можемо задати їх вручну. Ми можемо задати це ось таким чином:

```
w1 = 0.42
w2 = 0.15
w3 = -0.56
w4 = 0.83
w5 = 0.93
w6 = 0.02
new_weight = [np.array([[w1,w3],[w2,w4]]), np.array([[w5],[w6]])]
print(new_weight)
model.set_weights(new_weight)
```

6 наших зв'язків у нашій нейронній мережі, ми задаємо для них ручні значення і передаємо на нашу нейронну мережу і тепер, коли ми хочемо подивитися model.set.weights – вони у нас висвічуються:

```
[array([[ 0.42, -0.56],
       [ 0.15,  0.83]]), array([[0.93],
       [0.02]])]
```

От зараз у нашої моделі ось такі коефіцієнти знаходяться, які ми їй задали.

Коли ми створюємо нейронну мережу вручну, то ця процедура обов'язкова, тому що у нас немає функції створення нових шарів, яка рандомним чином буде заповнювати коефіцієнти нашої нейронної мережі і нам потрібно їх задавати вручну. Сама звичайна математика. Тепер потрібно через нашу нейронну мережу пропустити якесь певне значення. У нас два вхідних шари, ми хочемо на неї запустити значення x1 і значення x2 ось таким чином:

```
x1 = 7.2
x2 = -5.8
x_train = np.expand_dims(np.array([x1, x2]), 0)
x_train.shape
```

```
(1, 2)
```

Формуємо із цих значень тренувальний набір за допомогою NumPy. Переводимо їх в array:

```
k = np.array([x1, x2])
k.shape
```

```
(2,)
```

Запускаємо їх на нашу нейронну мережу. Оскільки ми використовуємо *TensorFlow*, ми можемо використати вже готову функцію *predict()*. У даному

конкретному випадку ми на нашу модель за допомогою функції *predict()* передаємо набір даних *x_train*, який ми тільки що згенерували, і отримуємо *y*-ки, тобто шукомі величини з нашого *x_train*. Давайте подивимось, що у нас будуть за *y*-ки на виході:

```
y_linear = model.predict(x_train)
print(y_linear)
```

```
1/1 [==============================] - 0s 155ms/step
[[1.8262998]]
```

Процес пішов, відпрацювалося, ми отримали такий показник нашого *y*-ку.

Подивимося на коефіцієнти нашої моделі:

```
model.get_weights()
```

```
[array([[ 0.42, -0.56],
        [ 0.15,  0.83]], dtype=float32),
 array([[0.93],
        [0.02]], dtype=float32)]
```

Те, що у нас було. Дослідимо, щоб ми отримали, пропустивши їх через певні ваги:

```
model(x_train)
```

```
<tf.Tensor: shape=(1, 1), dtype=float32, numpy=array([[1.8262998]], dtype=float32)>
```

Це у нас результат виконання роботи нашої нейронної мережі. Ось ми з вами тільки що отримали показник (1.82…), який нам демонструє як у нас змінилися два ось цих вхідних числа (7.2 і -5.8) – після того, як ми їх пропустили через нашу нейронну мережу.

Що стосується самих розрахунків, яким чином у нас відбуваються самі розрахунки. Вони у нас відбуваються по дуже простим формулам (3.1-3.3).

$$Y = N_1 * w_5 + N_2 * w_6 \quad (3.1)$$

$$N1 = x_1 * w_1 + x_2 * w_2 \quad (3.2)$$

$$N2 = x_1 * w_3 + x_2 * w_4 \quad (3.3)$$

N1 ми розраховуємо по другій формулі, N2 ми розраховуємо по третій формулі. Це у нас все важелі. А остаточний Y ми розраховуємо по першій

формулі. Ось так у нас відбуваються обчислення в нейронній мережі *Sequential()* між шарами *dense* – математика не дуже складна. Але ця математика передбачає, що наші коефіцієнти ні в якому разі не можуть дорівнювати нулю, бо якщо вони у нас дорівнюють нуль, то ми бачимо, що у нас йде операція множення і у нас все обнулюється.

Ми можемо в принципі точно таким же чином порахувати все це діло вручну і отримати той же самий показник, який ми отримали 1.82. Ми задаємо всі ці показники вручну. Ось у нас розрахунки для N1 і N2:

```
N1 = x1 * w1 + x2 * w2
N2 = x1 * w3 + x2 * w4
print(N1)
print(N2)
```

```
2.154
-8.846
```

І останній розрахунок у нас йде для нашого Y:

```
Y_linear = N1 * w5 + N2 * w6
print(Y_linear)
```

```
1.8263000000000003
```

У будь-якому випадку ось математика нашого процесу.

3.3 Функції активації нейронної мережі

Для того, щоб наша мережа працювала більш продуктивно, нам до неї потрібно додати деякі додаткові елементи.

Першим таким додатковим елементом у нас виступає елемент під назвою функція активації нейронної мережі. *Функція активації* визначає вихідне значення нейрона залежно від результату зваженої суми входів та порогового значення. Фактично, функція активації – це спосіб нормалізації даних у нейроні. Основними типами функцій активації є: ступінчаста, лінійна, сигмоїдна або логістична (рис.3.31).

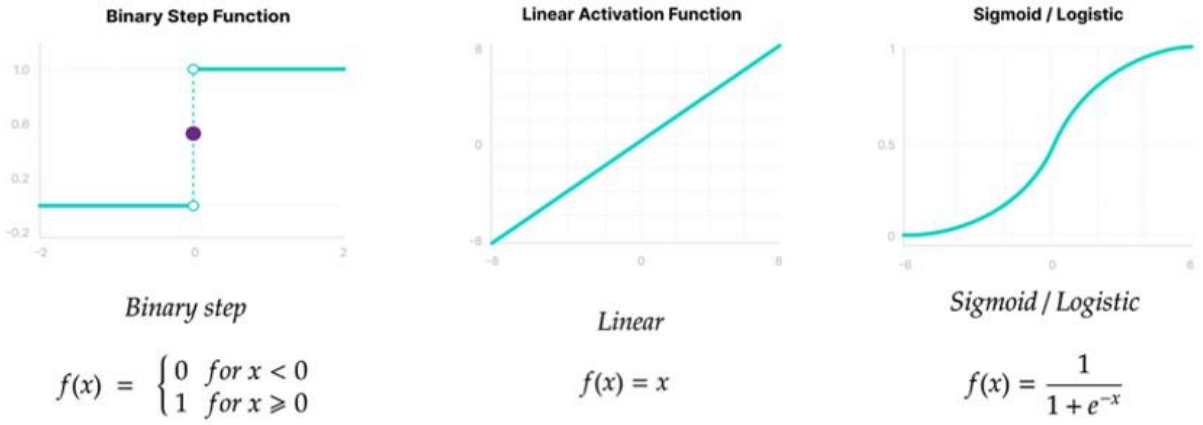


Рисунок 3.31 – Функції активації нейромереж (1)

Є ще наступні функції активації нейронних мереж: тангенса, ReLU і Leaky ReLU (рис.3.32).

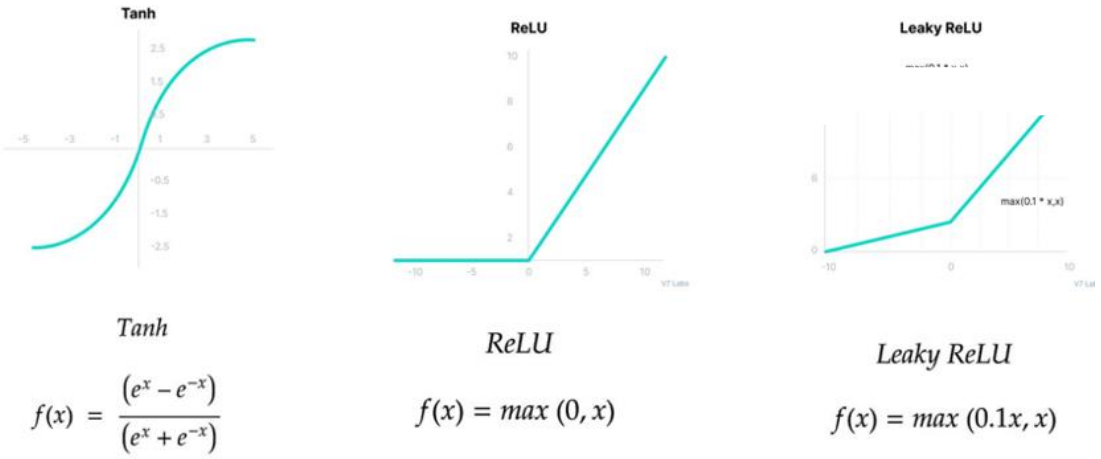


Рисунок 3.32 – Функції активації нейромереж (2)

Кожна із цих функцій мають своє певне обчислення. Маючи своє певне обчислення, вони по різному навантажують комп'ютерну систему при їх використанні. Бувають нейронні мережі по декілька сотень прихованих рівнів, на яких знаходяться тисячі нейронів і кожен зв'язок між ними активується за допомогою цієї функції. Якщо уявити декілька сотень тисяч обчислень, де у нас присутня експонента, тому потрібна потужна машина для роботи з нашими функціями.

Лінійна функція активації – найпростіша і менше всього навантажує систему. Її також називають функцією лінійного випрямлювача. Вона розраховується по формулі (3.4):

$$f(x) = \{0 \;\; x<0 \;\; або \; x \;\; x>=0\} \tag{3.4}$$

Яким чином дана функція програмується. Для того, щоб її передати на нейронну мережу, нам ось цю формулу потрібно було б записати в коді. Але ж ми цього робити не хочемо. Ми хочемо, щоб у нас була автоматизація. Тому функція, яка описує лінійний випрямлювач, знаходиться практично у всіх бібліотеках, які призначені для програмування нейронних мереж. Зокрема в бібліотеці TensorFlow вона теж присутня.

Спочатку створимо цю функцію розрахунку вручну:

```
def relu(x):
    return np.clip(x, 0, np.inf)
```

Функція ReLU (Rectified Linear Unit) – це одна з популярних функцій активації у штучних нейронних мережах, яка застосовується для активації внутрішніх шарів (прихованих шарів) нейронної мережі. Побудуємо невелику мережу ще одну, але вже з використанням цієї функції.

```
model_relu = Sequential()
model_relu.add(Dense(2, input_dim = 2, activation = 'relu', use_bias=False))
model_relu.add(Dense(1, activation = 'relu', use_bias=False))
model_relu.summary()
model_relu.set_weights(new_weight)
```

Функції активації передаються на шари нейронної мережі за допомогою аргументу, який називається activation. Зокрема функція активації методом лінійного випрямлювача у нас задається в TensorFlow за допомогою константи relu (зустрічається часто при роботі з нейронними мережами). Передаємо її на вхідний, перший прихований і на вихідний шари. Отримали summary.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 2)                 4

 dense_3 (Dense)             (None, 1)                 2

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

В summary як ми бачимо практично нічого не помінялося. Подивимось, як у нас зміниться результат після того, як ми на неї відправимо два числа, які ми використовували в самому першому випадку там де у нас ця функція активації не фігурувала. Давайте подивимося. Відправляємо числа і ми бачимо, що у нас тут число дещо змінилося:

```
y_relu = model_relu.predict(x_train)
print(y_relu)
```

```
1/1 [==============================] - 0s 80ms/step
[[2.0032198]]
```

Якщо ми його розрахуємо вручну, як ми це робили минулого разу:

```
H1_relu = relu(x1 * w1 + x2 * w2)
H2_relu = relu(x1 * w3 + x2 * w4)
print(H1_relu)
print(H2_relu)
```

```
2.154
0.0
```

Бачимо на виході наступний результат:

```
Y_relu = relu(H1_relu * w5 + H2_relu * w6)
print(Y_relu)
```

```
2.0032200000000002
```

Відповідно бачимо, що значення відрізнятися. Якщо раніше ми робили обчислення без функції активації, то тут ми вже задіюємо нашу функцію активації на кожному із кроків, тобто ось в даному конкретному випадку наше обчислення виступає як аргумент для даної функції.

Тепер, крім функції активації лінійним випрямлювачем, існують ще дві, які використовуються дуже часто. Справа в тому, що перевага лінійного

випрямлювача полягає в тому, що він рахує, що нейронна мережа з ним працює дуже швидко, але він не завжди може використовуватися. Тому дуже часто в якості функції активації у нейронних мережах використовується функція активації сигмоїдом:

$$f(x)=G(x)=\frac{1}{1+e^{-x}} \tag{3.5}$$

Запрограмуємо окремо функцію сигмоїди:

```
def sigmoid(x):
    return 1/1+np.e**(-x)
```

Створюємо ще одну нейронну мережу:

```
model_sigmoid = Sequential()
model_sigmoid.add(Dense(2, input_dim = 2, activation = 'sigmoid', use_bias=False))
model_sigmoid.add(Dense(1, activation = 'sigmoid', use_bias=False))
model_sigmoid.summary()
model_sigmoid.set_weights(new_weight)
```

Тут ми вже наші шари активуємо сигмоїдом.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 2)                 4

 dense_5 (Dense)             (None, 1)                 2

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Отримуємо наступний результат:

```
y_sigmoid = model_sigmoid.predict(x_train)
print(y_sigmoid)
```

```
1/1 [==============================] - 0s 82ms/step
[[0.6970569]]
```

```
H1_sigmoid = sigmoid(x1 * w1 + x2 * w2)
H2_sigmoid = sigmoid(x1 * w3 + x2 * w4)
print(H1_sigmoid)
print(H2_sigmoid)
```

```
1.116019151774409
6947.547135018005
```

```
Y_sigmoid = sigmoid(H1_sigmoid * w5 + H2_sigmoid * w6)
print(Y_sigmoid)
```

```
1.0
```

Активація сигмоїдом використовується дуже часто, особливо в тих нейронних мережах, які призначені для обробки зображення. Для того, щоб такі нейронні мережі продуктивно працювали, найчастіше ми шари в таких нейронних мережах активуємо за допомогою константи sigmoid. Функція активації специфічним чином проводить зміну наших коефіцієнтів на кожному із наших шарів. Якщо б нейронні мережі активувалися безпосередньо по простій формулі, то кожна нейронна мережа працювала б однотипно. У нас би не було б такого різноманіття нейронних мереж, що одна працює оброблюючи тексти, інша - розпізнає зображення, третя - генерує певне зображення. Ми б не могли кастомізувати роботу нейронних мереж таким чином, щоб вони виконували різноманітні задачі. Саме за рахунок функцій активації ми вносимо в поведінку наших нейронних мереж необхідний сумбур, який дозволяє отримувати нам нейронні мережі унікальні.

Третя функція активації, яка використовується досить часто – це функція активації гіперболічним тангенсом:

$$f(x)=th(x)=\frac{(e^{x}-e^{-x})}{(e^{x}+e^{-x})} \tag{3.6}$$

Як бачимо, дана функція активації дуже складна. І ми бачимо це з формули, тому що нам, щоб активувати кожен нейрон нашої нейронної мережі, потрібно вираховувати 4 експоненти для кожного. Це ускладнює розрахунки і навантажує систему. Напишемо цю функцію вручну:

```
def th(x):
    return (np.e ** x - np.e ** (-x)) / (np.e ** x + np.e ** (-x))
```

А тепер створимо нейронну мережу, шари якої активуються цією функцією, тобто вона абсолютно однотипна тим, що ми створювали раніше, але тут у нас вже в якості функції активації присутня константа *th*, причому звертаємо увагу – не в лапках, а просто *th*. Це константа для активації гіперболічним тангенсом.

```
model_th = Sequential()
model_th.add(Dense(2, input_dim = 2, activation=th, use_bias=False))
model_th.add(Dense(1, activation=th, use_bias=False))
model_th.summary()
model_th.set_weights(new_weight)
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 2)                 4

 dense_10 (Dense)            (None, 1)                 2

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Подивимось, як вона буде працювати з тими самими даними:

```
y_th = model_th.predict(x_train)
print(y_th)
```

```
1/1 [==============================] - 0s 113ms/step
[[0.70906264]]
```

Якщо підрахуємо вручну, то виходимо в принципі на той самий показник:

```
H1_th = th(x1 * w1 + x2 * w2)
H2_th = th(x1 * w3 + x2 * w4)
print(H1_th)
print(H2_th)
```

```
0.9734366670870239
-0.9999999585531038
```

```
Y_th = th(H1_th * w5 + H2_th * w6)
print(Y_th)
```

```
0.709062603515898
```

3.4 Використання метрик у нейронних мережах

Для того, щоб ми могли оцінити роботу нашої моделі, вона повинна мати якісь обчислювальні метричні показники. Ці метрики в нейронних мережах називаються *функціями втрат*. Їх різновидів вистачає, але найчастіше в якості

функцій втрат використовуються функції середньоквадратичної похибки, середньої абсолютної похибки, бінарної крос-ентропії.

Функція втрати (loss function) дозволяє порівняти значення прогнозу, зробленого нейромережею, із фактичними значеннями параметру, і таким чином оцінити ефективність роботи нейромережі. Розповсюджені функції:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 \tag{3.7}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y^{(i)} - \hat{y}^{(i)}| \tag{3.8}$$

$$CE\ Loss = \frac{1}{n}\sum_{i=1}^{N} -(y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)) \tag{3.9}$$

По формулі (3.7) розраховується значення середньоквадратичної похибки (Mean Squared Error, MSE). По формулі (3.8) розраховується значення середньої абсолютної похибки (Mean Average Error, MAE). По формулі (3.9) розраховується значення бінарної крос-ентропії або лог-лосс (Binary Cross-Enthropy, Log-loss).

Ці показники (характеристики) в нейронній мережі задаються на рівні активізації або компіляції нейронної мережі. Задаються вони так само як і функції активації певними константами, тільки константи ці називаються не activation, а називаються вони loss і metrics. Коли ми задаємо параметр metrics, ми можемо задавати декілька показників metrics, задавати у вигляді списку, тобто ми можемо прописувати metrics MSE, MAE і крос-ентропію, все це записати як текстові константи і передати. Що стосується функції втрат loss, як правило, використовується тільки одна.

У процесі навчання ми намагаємося мінімізувати значення функції втрати та оновлювати параметри нейромережі (ваги, присвоєні зв’язкам між нейронами) для підвищення точності. Ці параметри налаштовуються самим алгоритмом.

Оптимізатор – це метод досягнення найкращих результатів, що допомагає прискорити навчання, іншими словами, оптимізатор – це алгоритм, який

використовується для незначної зміни параметрів (таких як ваги та швидкість навчання), щоб модель працювала якомога правильніше та швидше.

Оптимізатор дозволяє прискорити адаптацію нашої нейронної мережі. Наша нейронна мережа в процесі свого навчання, а саме головне – в процесі своєї адаптації вона повинна постійно змінюватися, змінювати свої коефіцієнти. Значить ми повинні в нашу нейронну мережу внести деякий цифровий показник, який шляхом кожного проходження нових *x* через неї як під час навчання, так і під час її роботи, буде здвигати коефіцієнти (*w1…w6*) на якийсь невеликий показник.

Потрібно згадати математику, згадати тему присвячену лінійній регресії. Є один математичний метод, який нам дозволяє здвигати наш шарик, який рухається по сигмоїді, щоб він досяг найнижчого результату. Ця методика називається *градієнтний спуск.*

Стохастичний градієнтний спуск (Steep Gradient Descent, SGD) – алгоритм, у мінімум функції втрат шукається шляхом руху вздовж градієнта (вектора найшвидшої зміни значення функції). На відміну від класичного градієнтного спуску, SGD оновлює параметри моделі, використовуючи один випадково обраний елемент навчальної вибірки за одну ітерацію.

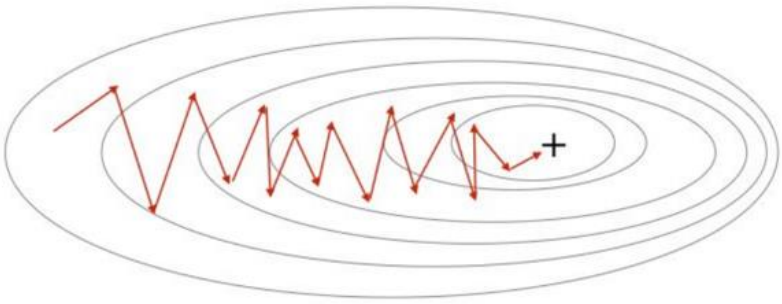


Рисунок 3.33 – Графічна репрезентація SGD

Градієнтний спуск – це методика розрахунку найнижчого показника значення нашої функції на певному відрізку її роботи. У градієнтному спуску для того, щоб наш шарик не почав літати з одної стінки на іншу і не завис у такому стані, використовується так званий крок сходження алгоритму – невелике число, яке у

кожного разу скорочує проміжок руху нашого шару. Подібний метод використовується в нейронних мережах, щоб кожного разу, коли через неї проходить інформація, у нас відбувалося зміщення наших коефіцієнтів. Для цього використовується оптимізатор, а в якості оптимізатора використовуються різні варіанти градієнтного спуску. Їх досить багато варіантів існує, але найчастіше і дуже популярний варіант градієнтного спуску для нейронних мереж – це так званий адаптивний градієнт. Він позначається спеціальною константою, називається ця константа *Adam* і зустрічається в такому вигляді практично у всіх фреймворках.

В основі методу *Adam* лежить розрахунок експоненційного середнього значення градієнта *m* та експоненційного середнього квадратів градієнта v. Кожне експоненційне середнє також має свій гіперпараметр β, що визначає параметр усереднення.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.10}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3.11}$$

$$w_{t+1} = w_t - \alpha \frac{\hat{m}}{\sqrt{\hat{v}} + \varepsilon} \tag{3.12}$$

Adam у нас зустрічається на етапі активації нашої нейронної мережі.

3.5 Практична реалізація нейронної мережі в середовищі Jupyter Notebook

Розробимо нейронну мережу, за допомогою якої ми будемо проводити розпізнавання рукописних цифр. Всі необхідні імпорти зробили. Потрібно завантажити дата сет:

```
(x_train_org, y_train_org), (x_test_org, y_test_org) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 23s 2us/step
```

У кожній бібліотеці є функція для завантаження навчальних дата сетів. У Sciket Learn є функція *load* для завантаження її навчальних дата сетів. У даному прикладі - це функція *load_data()*. Ця функція тут працює одночасно як функція *load* і як функція *train_test_split()*. Там де нам потрібно розбивати дані на навчальну і тренувальну вибірки. Функція *load* в TensorFlow зразу завантажує дані і зразу їх розкидає по необхідним дата сетам. Виконання прикладу займає деякий час, тобто у нас всі дані будуть завантажуватися. Визначаємо, що собою представляють *x*. Візьмемо нульовий індекс і подивимося:

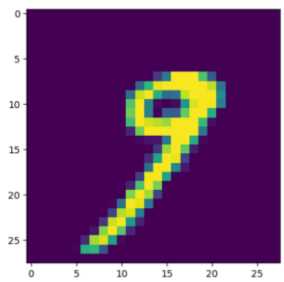
```
x_train_org[0]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
```

…

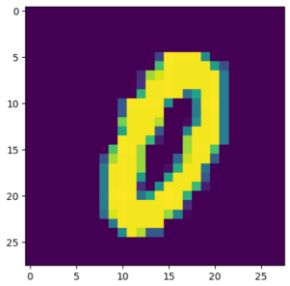
Діапазон цифр від 0 до 255. Три набори показників в діапазоні від 0 до 255 і таких наборів декілька. Ці цифри описують показники пікселів, бо даний тренувальний набір представляє собою набір піксельних зображень. Якщо ми візьмемо номер картинки 33 і спробуємо її візуалізувати за допомогою Matplotlib, отримаємо рукописну 9 (графічний файл рукописної дев’ятки):

```
n = 33
plt.imshow(x_train_org[n], cmap='viridis')
plt.show()
```



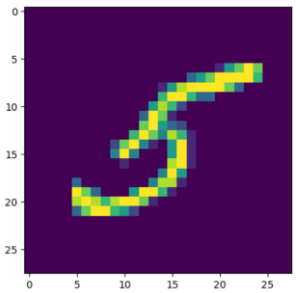
Кожний із квадратів (кожне із цих зображень) – це піксель. В Out[33] – його цифрові характеристики. Таких пікселів у кожному зображенні 28 на 28 = 784. Якщо подивитися на інше зображення, наприклад, на номер 34 у нашому дата сеті - це у нас рукописний 0:

```
n = 34
plt.imshow(x_train_org[n], cmap='viridis')
plt.show()
```



Номер 35 – це рукописна 5:

```
n = 35
plt.imshow(x_train_org[n], cmap='viridis')
plt.show()
```



Таких варіантів зображень дуже багато.

Перед тим як використовувати цей дата сет для роботи з нейронними мережами, нам потрібно з ним зробити маніпуляцію: це зображення 28 на 28 пікселів можна представити у вигляді ниточки. Коли ми його почнемо розпускати, ми його можемо витягнути в 784 пікселі нитку одну і перед тим як це передавати на нейронну мережу, потрібно це зображення перетворити на ниточку, а потім перевести ці значення до значень float32. Зараз ці значення у нас цілочисельні, а для нейронної мережі потрібно ці значення перевести у float32.

Робимо переформатування нашого дата сету, ми його розпускаємо: було 28 на 28, зараз стало 1 на 784.

```
x_train = x_train_org.reshape(60000, 784)
x_test = x_test_org.reshape(10000, 784)
print(x_train_org.reshape)
print(x_train.shape)
```

```
<built-in method reshape of numpy.ndarray object at 0x000001BBA28E7270>
(60000, 784)
```

Переводимо все у float32:

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

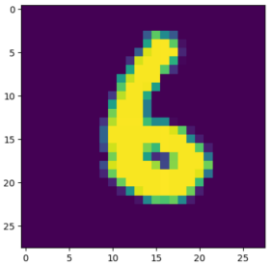
Ккрім предикторів є ще шукомі величини. Якщо у нас в якості предиктора йдуть значення пікселів зображень (цифра 5 на фіолетовому полі), то в якості шукомих величин у нас будуть звичайні цифри, які будуть відповідати тим самим цифрам.

```
y_train_org[35]
```

```
5
```

Вводимо номер 35 - показує 5, 36 по предикторам - 6.

```
n = 36
plt.imshow(x_train_org[n], cmap='viridis')
plt.show()
```



Вводимо по предикторам - отримуємо 6.

```
y_train_org[36]
```

```
6
```

Таким чином влаштований дата сет для нейронних мереж, який займається розпізнаванням зображень. Там є два набори: у першому наборі знаходяться

предиктори з певними індексами. Предиктори складаються з графічних зображень. У другому наборі знаходяться шукомі величини теж з індексами, які відповідають індексам предикторів, в яких знаходиться розшифровка у вигляді звичайної цифри або літери, або слова, якщо це у нас фотографія чиясь, наприклад, там літак – от там буде написано flaight і т.д. Категорізуємо це, поправимо наші *y*:

```
utils.to_categorical(y_train_org[0], 10)
```

```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

Переводимо дані у формат *one_hot_encoding* для того, щоб наш TensorFlow зміг з ними працювати.

```
y_train = utils.to_categorical(y_train_org, 10)
y_test = utils.to_categorical(y_test_org, 10)
```

y_train буде представляти собою такий набір: 10 значень, а всього записів 60000, тобто в нашому дата сеті 60000 варіантів цифрових чисел.

```
print(y_train.shape)
```

```
(60000, 10)
```

10 цифр від 0 до 9.

```
print(y_train[n])
```

```
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

```
print(y_train_org.shape)
```

```
(60000,)
```

```
print(y_train_org[36])
```

```
6
```

Завантажили і підготували дані, тепер нам потрібно створити нашу нейронну мережу. Створюємо нейронну мережу. Використовуємо тип мережі Sequential, шари типу dense, але з деякими невеликими змінними.

Нейронна мережа:

```
model = Sequential()
model.add(Dense(800, input_dim = 784, activation = 'relu'))
model.add(Dense(400, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
```

Модель Sequential, далі перший прихований шар - 800 нодів, input_layer – 784. На вхідному шарі буде 784 нейрони, кожен з яких буде приймати по одному пікселю, який він потім буде передавати на приховані шари. Далі у нас йде другий прихований шар – він у 2 рази менший, тобто ми починаємо згортати роботу нашої мережі, вона у нас конус має. І у першому, і у другому випадку ми активуємо наші шари за допомогою лінійної функції. Тому що нам тут потрібне мінімальне перетворення, ми працюємо із стандартними зображеннями з навчального набору. Якщо б це було б щось більш складне, наприклад, не зображення рукописних цифр, а фотографії, тут би нам знадобилася якась інша функція активації, наприклад, активація сигмоїдою. Але в даному випадку з нас буде достатньо значення relu. Самий останній шар має розмір 10, він нам буде видавати відповідь від 0 до 9, якраз 10 нейронів нам для цього вистачить. Ми його активуємо за допомогою специфічної функції, яка називається softmax. Softmax як правило використовується для активації прихованих шарів. Використовується при роботі з класифікацією зображень і використовується тоді, коли нам потрібно отримати два або трохи більше кількості класів на виході. У даному випадку у нас класів буде 10 від 0 до 9. Ми її тут будемо використовувати. Це структура нашої мережі. Тепер нам нашу мережу потрібно скомпілювати, передавши на неї додаткові параметри. Ось тут ми вже передаємо на неї наш оптимізатор, наші метрики і наші функції втрат. Проводимо компіляцію за допомогою функції compile:

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

Передаємо функцію втрат categorial_crossentropy (використовуємо дану модель для категоріальної класифікації). Тому функція втрат буде відповідати саме такій, яка нам потрібна для нашої мережі для роботи її як класифікаційної. Далі, щоб коефіцієнти у наших нейронах змінювалися, ми будемо використовувати оптимізатор *Adam*. Кожного разу під час навчання і під час роботи *Adam* буде зсувати значення наших коефіцієнтів в якийсь певний напрямок. І нарешті в якості метрики будемо використовувати самий звичайний accuracy. А якщо нам

потрібно щось ще, ми можемо поставити тут кому і написати щось інше, наприклад, MAE. Отримаємо MAE, але нам тут в першому наближенні достатньо буде самого accuracy.

Після того як ми через функцію compile додали до нашої моделі ці додаткові параметри, можемо подивитися на модель summary:

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 800)               628000

 dense_12 (Dense)            (None, 400)               320400

 dense_13 (Dense)            (None, 10)                4010

=================================================================
Total params: 952410 (3.63 MB)
Trainable params: 952410 (3.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

628000 – кількість зв’язків між нейронами вхідного рівня, яких 784 і нейронами першого прихованого рівня, яких 800. 320400 – кількість зв’язків між нейронами першого прихованого рівня і другого прихованого рівня. 4010 – кількість зв’язків між нейронами другого прихованого рівня і вихідного шару.

Тепер нам потрібно навчити нашу модель. Робимо це за допомогою функції fit():

```
model.fit(x_train, y_train, batch_size=128, epochs=15, verbose=1, validation_split=0.2)
```

Працює вона приблизно так само як і функції в моделях ML. Тобто ми передаємо навчальний набір *x*-ів, навчальний набір *y*-ів, далі у нас пішли ньюанси. Самими важливими показниками, які тут у нас є – це кількість епох навчання. Епоха навчання – це кількість прогонок нашого навчального дата сету через нашу нейронну мережу. Якщо в моделі ML ми навчаємо за одну епоху, то наші нейронні мережі, щоб досягти певного рівня готовності мають навчатися декілька епох.

Batch_size – кількість елементів зображень, які ми кидаємо за кожний хоп нашого навчання. З іншими показниками будемо розбиратися. А зараз як власне кажучи відбувається процес навчання нашої нейронної мережі:

```
model.fit(x_train, y_train, batch_size=128, epochs=15, verbose=1, validation_split=0.2)
```

```
Epoch 1/15
375/375 [==============================] - 10s 22ms/step - loss: 0.2274 - accuracy: 0.9336 - val_loss: 0.1092 - val_accuracy:
0.9667
Epoch 2/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0838 - accuracy: 0.9748 - val_loss: 0.0844 - val_accuracy: 0.
9730
Epoch 3/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0496 - accuracy: 0.9844 - val_loss: 0.0836 - val_accuracy: 0.
9750
Epoch 4/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0352 - accuracy: 0.9887 - val_loss: 0.0834 - val_accuracy: 0.
9766


Epoch 5/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0260 - accuracy: 0.9918 - val_loss: 0.0840 - val_accuracy: 0.
9771
Epoch 6/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0239 - accuracy: 0.9922 - val_loss: 0.0779 - val_accuracy: 0.
9791
Epoch 7/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0164 - accuracy: 0.9946 - val_loss: 0.0861 - val_accuracy: 0.
9787
Epoch 8/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0157 - accuracy: 0.9948 - val_loss: 0.1061 - val_accuracy: 0.
9745
Epoch 9/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0161 - accuracy: 0.9947 - val_loss: 0.0991 - val_accuracy: 0.
9773
Epoch 10/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0118 - accuracy: 0.9962 - val_loss: 0.1018 - val_accuracy: 0.
9783


Epoch 11/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0134 - accuracy: 0.9953 - val_loss: 0.1282 - val_accuracy: 0.
9746
Epoch 12/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0090 - accuracy: 0.9973 - val_loss: 0.1093 - val_accuracy: 0.
9783
Epoch 13/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0148 - accuracy: 0.9954 - val_loss: 0.1002 - val_accuracy: 0.
9797
Epoch 14/15
375/375 [==============================] - 8s 20ms/step - loss: 0.0103 - accuracy: 0.9970 - val_loss: 0.1096 - val_accuracy: 0.
9770
Epoch 15/15
375/375 [==============================] - 8s 21ms/step - loss: 0.0058 - accuracy: 0.9981 - val_loss: 0.0977 - val_accuracy: 0.
9802

<keras.src.callbacks.History at 0x1bba5f02b00>
```

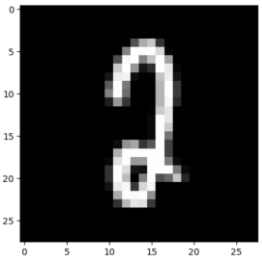
Відбувається навчання моделі. Все це буде відбуватися від 1 епохи до 15-тої.

Коли ми говоримо про ітерацію – це ітерація якогось певного циклу. Коли ми навчаємо модель машинного навчання, ми її навчаємо не в циклі, ми її навчаємо просто передаючи на неї навчальний набір і він проходить через неї один раз. Тут ми навчальні набори пропускаємо через нашу нейронну мережу (один і той же навчальний набір) декілька разів – це називається *епоха*. З моделями ML ми такого не робимо, ми там раз через неї прогнали і працюємо – дивимось, які

результати. Тут у нас наш тренувальний набір був пропущений 15 разів і на кожному із пропусків бачимо цікаві для нас показники. Бачимо показник метрику accuracy нашої моделі – вона змінюється з проходженням кожної епохи і зрештою від 0.9317 збільшується до 0.9966 на 15-тій епосі. Але саме головне – це функція втрат – коефіцієнт необроблених даних. Якщо у нас на перших етапах були показники 0.23, то на самому останньому етапі у нас показник йде 0.0115. Ми можемо прогнати навчальний набір не 15, а 30-40 разів, 100 разів – скільки захочемо, скільки ми сюди задамо. Але кількість цієї прогонки не завжди є оптимальною. Тому що якщо ми проганяємо через нашу нейронну мережу навчальний набір – велику кількість разів, у нас на певному етапі наші коефіцієнти перестають суттєвим чином змінюватися. Коли ми почнемо пропускати через таку мережу якісь дані із тестового набору, вона їх просто не буде розпізнавати. Проте вона буде прекрасно розпізнавати дані з набору тренувального. Це називається перенавченість мережі. Ці показники потрібні для того, щоб ми слідкували чи відбувається у нас зміна наших коефіцієнтів. Якщо у нас на певному етапі accuracy перестає змінюватися і перестає змінюватися показник функції втрат, нам потрібно робити висновок, що от на даній конкретній епосі у нас наша мережа перестала навчатися, тоді ми перериваємо процес і запускаємо навчання нашої мережі, але вже з цим остаточним показником нашої епохи, на якій у нас перестали змінюватися ось ці показники.

Звичайно є спеціальні прийоми для того, щоб відслідковувати це автоматично. У даному конкретному випадку ми пропустили наш навчальний набір 15 разів і тепер все, що нам залишається – це перевірити результати. Для перевірки результатів використовуємо дані із нашого тестового набору In[32] (*x_test_org*), тобто той, який навчання не проходив і якщо наша нейронна мережа буде розпізнавати ці цифри, значить ми можемо сказати, що вона нормально навчилася і працює теж нормально. Візьмемо це зображення під номером 1356:

```
n_rec = 1356
plt.imshow(Image.fromarray(x_test_org[n_rec]).convert('RGBA'))
plt.show()
```



Потрібно його перетворити до того вигляду, в якому ми пускаємо його на нейронну мережу. Тобто з розміру 28 на 28 нам потрібно його перетворити на 1 на 784, тобто переформатувати. Виконуємо це за допомогою характеристик NumPy:

```
x = x_test[n_rec]
x = np.expand_dims(x, axis=0)
```

Предикція відбувається за допомогою функції *predict*. В якості передачі даних ми передаємо дану цифру.

```
prediction = model.predict(x)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001BBA9
2D2170> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating
@tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. Fo
r (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avo
id unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://
www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 0s 107ms/step
```

Відбулася наша предикція. Результати предикції:

```
print(prediction)
```

```
[[1.33180865e-05 9.80238113e-08 9.99731481e-01 3.38007267e-05
  1.11515886e-13 4.26566205e-09 4.46896875e-09 9.98641735e-06
  2.11305320e-04 3.27115487e-11]]
```

Для того, щоб нам все ж таки з’ясувати, що ж воно там розпізнало, нам потрібно за допомогою функції NumPy *argmax* вибрати максимальне значення з цього і подивитися, якому індексу воно буде відповідати. Тобто ми за допомогою функції *argmax* вибираємо аргумент з максимальним індексом. Тому що за кожним із цих даних закріплений певний аргумент у вигляді цифри від 0 до 9.

```
np.argmax(prediction)
```

```
2
```

Змінимо умови. Замість 1356 номеру поставимо 1360. Це у нас одиниця. Робимо з нею те ж саме – запускаємо на предикцію:

```
prediction = model.predict(x)
```

```
1/1 [==============================] - 0s 35ms/step
```

Візьмемо ще варіант 1366. Предикція, *argmax*. В результаті отримуємо:

```
np.argmax(prediction)
```

```
2
```

Отже, ми навчили нашу нейронну мережу розпізнавати рукописні символи. Для цього ми з вами використали нейронну мережу, яка складається з чотирьох шарів, включаючи вхідний, вихідний і два прихованих, 800 і 400 нодів. Ми використовували для активації цих шарів тільки функцію лінійного випрямлювача і в якості оптимізації ми використовували оптимізатор - адаптивна оцінка моментів (Adam).

Існують різні варіанти нейронних мереж. Їх можна запрограмувати ось цими параметрами і вибором певних типів шарів, для того, щоб нейронна мережа виконувала конкретні задачі.

У даному прикладі 14-та епоха краща, ніж 15-та. Вистачило 14 епох. Просто ми задали 15. Саме головне – є спеціальний функціонал у фреймворках, який відслідковує при яких параметрах, тобто це точно такий самий механізм, який ми використовували для пошуку гіперпараметрів моделі ML. Існують спеціальні методи підбору гіперпараметрів для нейронних мереж, зокрема кількості епох.

Отже, використання моделі *Sequential* у глибокому навчанні для розробки нейронної мережі має наступні переваги:

1) Простоту використання. Модель Sequential забезпечує простий та інтуїтивно зрозумілий інтерфейс для створення послідовних моделей нейронних мереж. Легко додавати нові шари, визначати конфігурацію та компілювати модель для тренування.

2) Чітку послідовність шарів. Модель Sequential вимагає,щоб кожен шар мережі додавався послідовно, що робить код більш зрозумілим та легким для відлагодження.

3) Підтримку багатьох типів шарів. Sequential підтримує широкий спектр шарів, таких як Dense (повністю з’єднаний), Conv2D (згортковий), LSTM (довгострокова пам’ять) та інші.

4) Інтеграцію з бібліотекою Keras. Модель Sequential є частиною бібліотеки Keras, яка є високорівневим API для глибокого навчання. Це полегшує використання інших функцій Keras, таких як функції втрат та оптимізатори.

5) Високий рівень абстракції. Модель Sequential надає високий рівень абстракції, приховуючи деталі роботи нейронних мереж. Це корисно для початківців та для швидкого прототипування.

6) Вартість навчання та використання. Sequential зазвичай відносно ефективний щодо ресурсів, оскільки не потребує складних налаштувань та конфігурацій.

ВИСНОВКИ

Штучний інтелект використовується для розробки систем, які можуть автоматично приймати рішення на основі аналізу даних. Це може включати в себе прийняття рішень у реальному часі, що важливо у сферах, де швидкість реакції має значення. Штучний інтелект відіграє ключову роль у розвитку та оптимізації процесів аналізу та обробки даних у різних сферах, сприяючи покращенню прийняття рішень та забезпеченню нових можливостей для інновацій.

Нейронні мережі можна успішно використовувати у різних областях, таких як машинне навчання, глибоке навчання і є універсальним інструментом, який може бути застосований у багатьох галузях. Останні роки принесли значний прогрес у глибинному навчанні, яке базується на використанні нейронних мереж з багатьма шарами (глибинними мережами). Це дозволяє досягати кращих результатів у розпізнаванні образів, класифікації даних та інших завданнях.

Найпопулярнішими фреймворками для роботи з технологіями ML/DL є TensorFlow, Keras, PyTorch, Theano. Перше місце розділяють два фреймворки – TensorFlow та PyTorch.

TensorFlow має широкою розповсюджені та активні співтовариства розробників та користувачів. Це дозволяє швидше вирішувати проблеми, отримувати підтримку та використовувати багато готових рішень, які розробили інші користувачі. TensorFlow надає засоби для розгортання моделей на різних платформах, включаючи мобільні пристрої, вбудовані системи та хмарні середовища. Загалом, TensorFlow є потужним та гнучким інструментом, який широко використовується в галузі машинного навчання та глибинного навчання завдяки своїм функціональним можливостям та активній спільноті.

Висока гнучкість PyTorch дозволяє користувачам реалізовувати складні моделі та експериментувати з різними аспектами навчання моделей. PyTorch дозволяє будувати моделі за принципом модульності, розділяючи їх на компактні блоки. Це полегшує перевикористання та модифікацію окремих компонентів.

Кожен фреймворк має свої сильні сторони, і вибір між TensorFlow і PyTorch може залежати від конкретних потреб та особливостей проекту.

Jupyter Notebook є зручним середовищем для розробки, експериментів та документування процесу розробки нейронних мереж та інших алгоритмів машинного навчання, є популярним інструментом у навчальних закладах та серед дослідників. Має широкою розповсюджене і активне співтовариство користувачів.

Модель Sequential робить процес побудови нейронної мережі простим і зручним з можливістю додавати шари до моделі в порядку, в якому вони повинні бути використані для передавання сигналу від входу до виходу. Хоча модель Sequential є простою та зручною для багатьох задач, для складних архітектур може знадобитися використання інших підходів та кастомних класів моделей для більшої гнучкості.

Метод оптимізації Adam автоматично адаптує швидкість навчання для кожного параметра окремо. Це дозволяє алгоритму ефективно адаптуватися до різних змін швидкостей у різних частинах параметричного простору. Adam легко реалізовується у багатьох бібліотеках машинного навчання, таких як TensorFlow та PyTorch і використовується за замовчуванням у багатьох навчальних задачах.

Хоча Adam є ефективним методом оптимізації для багатьох завдань, важливо також враховувати його можливі недоліки та експериментувати з іншими алгоритмами оптимізації в залежності від конкретного завдання та архітектури моделі.

ПЕРЕЛІК ПОСИЛАНЬ

1. http://www.learnpython.org/en/Welcome
2. http://realpython.com/python3-object-oriented-programming/
3. http://leetcode.com/problemset/all/
4. http://perso.limsi.fr/pointal/
5. http://media/python:cours:mementopython3-english.pdf
6. https://numpy.org/
7. https://pandas.pydata.org/
8. https://matplotlib.org/
9. https://scipy.org/
10. https://scikit-learn.org/stable/
11. https://keras.io/
12. https://pytorch.org/
13. https://www.tensorflow.org/
14. W. Yu, W. Cheng, C. Aggarwal, K. Zhang, H. Chen, and Wei Wang. NetWalk: A flexible deep embedding approach for anomaly Detection in dynamic networks, ACM KDD Conference, 2018.
15. Jake VanderPlas Python Data Science Handbook: Essential Tools for Working with Data, 2022, 551p.
16. Heller, Nicholas et al. (2020). The KiTS19 Challenge Data: 300 Kidney Tumor Cases with Clinical Context, CT Semantic Segmentations, and Surgical Outcomes. arXiv: 1904. 00445 [q-bio.QM].
17. Hollo, Kaspar (2019). Exploring the Value of Weakly-Supervised Deep Learning Approaches for Artefact Segmentation in Brightfield Microscopy Images. URL: https://comserv. cs . ut . ee / home / files / hollo _ softwareengineering _ 2021.
18. Wang, Haofan et al. (2020). Score-CAM: Score-Weighted Visual Explanations for Convo-lutional Neural Networks. arXiv: 1910.01279 [cs.CV].
19. Yang, Guanyu et al. (2020). “Weakly-supervised convolutional neural networks ofrenal tumor segmentation in abdominal CTA images”. In: BMC Medical Imaging

20.1, p. 37. ISSN: 1471-2342. DOI: 10.1186/s12880- 020- 00435- w. URL: https://doi.org/10.1186/s12880-020-00435-w.

20. Selvaraju, Ramprasaath R. et al. (2019). “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: International Journal of ComputerVision 128.2, pp. 336–359. DOI: 10 . 1007 / s11263 - 019 - 01228 - 7. URL: https://doi.org/10.1007\%2Fs11263-019-01228-7.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)