НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра штучного інтелекту

До захисту допущено:
Завідувач кафедри
_____ Чумаченко Олена
«__» _____20__ р.

Дипломна робота
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи та методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Система генерування навчальної вибірки в задачах напівкерованого навчання на основі генеративно-змагальних мереж»

Виконала:
студентка IV курсу, групи КІ-91
Маєвська Катерина Сергіївна _____

Керівник:
д.т.н. проф. Синєглазов В. М. _____

Консультант з економічного розділу:
Доцент, к.е.н. Рощина Надія Василівна _____

Консультант з нормоконтролю:
Гончарук Максим Миколайович _____

Рецензент:
к.т.н. Василенко Микола Павлович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О. І. Чумаченко

«__» _____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Маєвської Катерини Сергіївни

1. Тема роботи: «Система генерування навчальної вибірки в задачах напівкерованого навчання на основі генеративно-змагальних мереж», керівник роботи Синєглазов Віктор Михайлович, затверджений наказом по університету від «30» травня 2023 року №2065-с.
2. Термін подання студентом роботи 15.06.2023
3. Вихідні дані до роботи: чорно-білі зображення цифр рукописним шрифтом.
4. Зміст роботи: Аналіз предметної області, обробка літературних джерел, аналіз алгоритму розв'язку задачі напівкерованого навчання за допомогою генеративно-змагальних мереж, розробка програмного продукту для розв'язку задачі класифікації, його тестування та аналіз результатів.
5. Перелік ілюстративного матеріалу: опис експериментальних наборів даних, схеми роботи алгоритму, архітектура нейронної мережі, результати роботи програмного продукту.
6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Економічний	Рощина Надія Василівна, канд.економ.наук, доцент		

7. Дата видачі завдання: 21 лютого 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури за темою	12.03.2023	
2.	Підготовка першого розділу	16.04.2023	
3.	Підготовка другого розділу	7.05.2023	
4.	Розробка програмного продукту	18.05.2023	
5.	Підготовка третього розділу	21.05.2023	
6.	Підготовка економічної частини	24.05.2023	
7.	Оформлення розділів відповідно до нормоконтролю	26.05.2023	
8.	Оформлення дипломної роботи	28.05.2023	
9.	Підготовка презентації доповіді	28.05.2023	

Студент Катерина МАЄВСЬКА

Керівник Віктор СИНЄГЛАЗОВ

РЕФЕРАТ

Дипломна робота: 124 с., 9 табл., 30 рис., 2 додатки, 36 джерел.

МАШИННЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, КЛАСИФІКАЦІЯ, ГЕНЕРАТИВНО-ЗМАГАЛЬНІ МЕРЕЖІ, НАПІВКЕРОВАНЕ НАВЧАННЯ.

У роботі було розглянуто теоретичні відомості в області штучного інтелекту, нейронних мереж та машинного навчання і їх класифікацій. Проведено огляд принципу роботи та архітектури генеративно-змагальних мереж, побудована експериментальна реалізація для двох задач. Було досліджено основні принципи напівкерованого навчання та розглянуто його архітектуру з інтеграцією генеративно-змагальних мереж, побудовано власну реалізацію розв’язку задачі класифікації зображень рукописних цифр, проведено тестування та порівняльний аналіз.

Об’єктом дослідження стали генеративно-змагальні мережі та напівкероване навчання.

Предметом дослідження були методи застосування генеративно-змагальних мереж для розв’язку задач напівкерованого навчання.

ABSTRACT

Thesis: 124 p., 9 tabl., 30 fig., 2 appendixes, 36 sources.

MACHINE LEARNING, ARTIFICIAL INTELLIGENCE, CLASSIFICATION, GENERATIVE-ADVERSARIAL NETWORKS, SEMI-SUPERVISED LEARNING.

Theoretical knowledge in artificial intelligence, neural networks, machine learning and their classification was considered in this work. There was also an overview of the operation methods and the architecture of generative-adversarial networks and the experimental realization for two problems was built. The main principles of semi-supervised learning, its architecture with generative-adversarial networks integration was considered, custom realization was built to solve the problem of handwritten digits recognition. Testing and comparative analysis were provided.

The object of the studies are generative-adversarial networks and semi-supervised learning.

The subject of the studies are the methods of generative-adversarial networks integration for solving semi-supervised learning problems.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТНИХ СКОРОЧЕНЬ……...10

ВСТУП……..……11

РОЗДІЛ 1 ПРОБЛЕМИ ШТУЧНОГО ІНТЕЛЕКТУ……..13

1.1 Розвиток штучного інтелекту в Україні……...14
1.2 Штучні нейронні мережі та їх класифікація……19
1.2.1 Визначення штучних нейронних мереж……...19
1.2.2 Навчання нейронної мережі……...21
1.2.3 Класифікація нейронних мереж……22
1.2.4 Нейронні мережі прямого поширення……..23
1.2.5 Нейронні мережі зворотного поширення……23
1.2.6 Багатошарові нейронні мережі……..24
1.2.7 Рекурентні нейронні мережі……...25
1.2.8 Згорткові нейронні мережі……25
1.3 Машинне навчання. Класифікація……27
1.3.1 Визначення машинного навчання……27
1.3.2 Класифікація методів машинного навчання……29
1.3.3 Навчання з учителем……..30
1.3.4 Навчання без учителя……31
1.3.5 Напівкероване навчання……31
1.3.6 Навчання з підкріпленням……...32
1.3.7 Класичне навчання……..33
1.3.8 Глибинне навчання……35
1.4 Генеративно-змагальні мережі та їх застосування……35
1.5 Висновки до першого розділу……...39

РОЗДІЛ 2 ГЕНЕРАТИВНО-ЗМАГАЛЬНІ МЕРЕЖІ……40

2.1 Топологія генеративно-змагальної мережі……..40

2.1.1 Генеративно-змагальні мережі, побудовані за принципом максимальної правдоподібності……..40
2.1.2 Класифікація моделей, побудованих за принципом максимальної правдоподібності……..42
2.1.3 Принцип роботи генеративно-змагальних мереж……43
2.2 Структурно-параметричний синтез генеративно-змагальних мереж. Постановка завдання……...45
2.2.1 Постановка завдання структурно-параметричного синтезу генеративно-змагальних мереж…….……46
2.2.2 Постановка завдання……...48
2.3 Огляд методів побудови генеративно-змагальних мереж……..48
2.3.1 Мова програмування Python……49
2.3.2 Бібліотеки PyTorch та math…….50
2.3.3 Середовище розробки Colab……51
2.3.4 Огляд інших методів побудови генеративно-змагальних мереж……..51
2.4 Алгоритм структурно-параметричного синтезу генеративно-змагальних мереж……53
2.5 Перевірка алгоритму на практиці…….56
2.5.1 Реалізація задачі 1……56
2.5.2 Реалізація задачі 2……59
2.5.3 Висновок……...61
2.6 Висновки до другого розділу……...62
РОЗДІЛ 3 МОДИФІКАЦІЯ МЕТОДІВ СТРУКТУРНО-ПАРАМЕТРИЧНОГО СИНТЕЗУ ГЕНЕРАТИВНО-ЗМАГАЛЬНИХ МЕРЕЖ. ПРАКТИЧНА РЕАЛІЗАЦІЯ……...63
3.1 Аналіз запропонованого алгоритму структурно-параметричного синтезу генеративно-змагальних мереж……...63
3.1.1 Балансування генератора і дискримінатора……..65
3.1.2 Функції вартості……...66

3.1.3 Кількість шарів та їх представлення……..……66
3.2 Алгоритм генеративно-змагальних мереж в задачах напівкерованого навчання……...…66
3.2.1 Основні принципи напівкерованого навчання……..67
3.2.2 Порівняльний аналіз GAN і SGAN…….68
3.3 Опис програмної реалізації алгоритму напівкерованого навчання з використанням генеративно-змагальних мереж……..70
3.3.1 Постановка задачі…….71
3.3.2 Опис структури розробленої мережі……..71
3.3.3 Опис програмної реалізації…….73
3.3.4 Аналіз результатів……76
3.3.5. Порівняльний аналіз роботи розробленої моделі та звичайної багатошарової нейронної мережі……79
3.4 Огляд ідей практичного застосування генеративно-змагальних мереж……81
3.4.1 Циклічна генеративно-змагальна мережа (cycle GAN)……82
3.4.2 Покращення якості зображень……83
3.4.3 Синтез зображень високої якості……83
3.4.4 Відновлення пошкоджених зображень……..84
3.4.5 Розпізнавання медичних аномалій…….85
3.5 Висновки до третього розділу……..86
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ……87
4.1 Постановка задачі……..87
4.1.1 Обґрунтування функцій програмного продукту……...88
4.1.2 Варіанти реалізації основних функцій……...89
4.2 Обґрунтування системи параметрів програмного продукту……92
4.2.1 Опис параметрів……...92
4.2.2 Аналіз експертного оцінювання параметрів……..94
4.3 Економічний аналіз варіантів розробки ПП……...97

4.4 Висновки до четвертого розділу……104

ВИСНОВКИ……105

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ…….107

ДОДАТОК А……112

ДОДАТОК Б……..……...121

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

НМ – нейронна мережа;

ПП – програмний продукт;

ШІ – штучний інтелект;

CNN – Convolutional Neural Network – згорткова нейронна мережа;

GAN – Generative Adversarial Networks - генеративно-змагальні мережі;

ML – Machine Learning – машинне навчання;

SGAN – Semi-supervised Generative Adversarial Networks - генеративно-змагальні мережі з напівкерованим навчанням;

SGD – Stochastic Gradient Descent – стохастичний градієнтний спуск;

SSL – Semi-Supervised Learning – напівкероване навчання.

ВСТУП

Машинне навчання сьогодні набуло широкої популярності і застосовується в багатьох сферах людської діяльності. Алгоритми машинного навчання, відповідно, дозволяють вирішувати широкий спектр задач, вдосконалювати існуючи моделі та створювати нові.

Актуальність даної дипломної роботи полягає в тому, що напівкероване навчання є видом машинного навчання, яке допомагає вирішити проблему, з якою стикаються багато розробників і спеціалістів, а саме – недостатність екземплярів навчальної вибірки. Для багатьох практичних задач збір та позначення навчального набору даних є ледь не найбільш витратною частиною проектування, оскільки займає багато часу, ресурсів та інколи є досить коштовним, в залежності від специфіки задачі. Напівкероване навчання, в свою чергу, потребує лише невеликого набору позначених даних і набору непозначених, які значно легше знайти. В основі роботи даного типу машинного навчання є генеративно-змагальні мережі, що генерують навчальну вибірку, тим самим доповнюючи дані і забезпечуючи ефективне навчання нейронної мережі. Незважаючи на те, що дані алгоритми є досить молодими, вони вже набули великої популярності у дослідників і допомагають розв’язати великий обсяг різноманітних задач, а активні дослідження у цій області допомагають знаходити нові модифікації та удосконалення кожного року.

Об’єктом дослідження стали генеративно-змагальні мережі та напівкероване навчання.

Предметом дослідження були методи застосування генеративно-змагальних мереж для розв’язку задач напівкерованого навчання.

Тематика першого розділу присвячена огляну предметної області, а саме ознайомленню зі станом галузі штучного інтелекту в Україні, основними проблемами та засобами їх вирішення, з теоретичною базою в області штучних нейронних мереж та їх класифікацією, а також з машинним навчанням, його основними принципами і видами. Також було розглянуто основні передумови виникнення генеративно-змагальних мереж та їх сучасне застосування, принцип роботи та архітектури генеративно-змагальних мереж, їх інтеграції у алгоритми напівкерованого навчання, перспективи подальшого вдосконалення та практичного застосування.

У другому розділі було більш детально розглянуто алгоритм та архітектуру генеративно-змагальних мереж, проаналізовано перспективи їх досліджень та їх інтеграції в напівкероване навчання. Також було розглянуто алгоритм структурно-параметричного синтезу генеративно-змагальних мереж, основні засоби та інструменти програмної реалізації, побудовано та протестовано програмну модель, зроблено висновки з результатів експерименту.

У третьому розділі було розглянуто модифікацію методів структурно-параметричного синтезу генеративно-змагальних мереж, побудовано програмну модель, яка була протестована на ряді прикладів. Проведено порівняльний аналіз, зроблено висновки.

У четвертому розділі було проведено функціонально-вартісний аналіз розробленого програмного продукту, оцінку вартості та пошук оптимальних ресурсів. Зроблено висновки.

РОЗДІЛ 1. ПРОБЛЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Розвиток штучного інтелекту в Україні

На сьогоднішній день сфера штучного інтелекту привертає все більше уваги дослідників та спеціалістів, оскільки важко не помітити того активного розвитку, в якому наразі перебуває ця галузь. Штучний інтелект сьогодні є не тільки зручним та сучасним інструментом, що дозволяє розв'язувати цілу низку важливих практичних та теоретичних задач, а ще й надзвичайно перспективної з наукової точки зору галуззю, яка дозволяє спеціалістам експериментувати з впровадженням різноманітних алгоритмів, підходів та теорій, розвивати свої знання в області математики, фізики, хімії, біології, психології, медицини, комп'ютерних наук тощо, робити відкриття та реалізовувати свої ідеї.

Незважаючи на те, що термін штучного інтелекту наразі широко використовується в науковій літературі, в публіцистиці та в медіа, однозначного формулювання у нього досі немає. Деякі автори визначають це як окрему галузь інформатики, інші ж представляють цей термін як розділ комп'ютерної лінгвістики, а дехто визначає це як окрему властивість машин та алгоритмів. Проте, найчастіше поняття штучного інтелекту вживається в контексті окремої наукової сфери, або галузі, що займається розробкою машин, або програмного забезпечення, що здатні розв'язувати задачі, які вимагають людського інтелекту, як-от розпізнавання образів, звуків, прийняття рішень, знаходження розв'язку певних задач і проблем [1].

2 грудня 2020 року було опубліковане розпорядження Кабінету Міністрів України про схвалення Концепції розвитку штучного інтелекту в Україні, де були визначені основні терміни, оголошено напрямок та

стратегію розвитку галузі штучного інтелекту, розглянуто основні перспективи та загрози [2]. Згідно з цим розпорядженням, термін штучного інтелекту має наступне тлумачення:

Штучний інтелект - організована сукупність інформаційних технологій, із застосуванням якої можливо виконувати складні комплексні завдання шляхом використання системи наукових методів досліджень і алгоритмів обробки інформації, отриманої або самостійно створеної під час роботи, а також створювати та використовувати власні бази знань, моделі прийняття рішень, алгоритми роботи з інформацією та визначати способи досягнення поставлених завдань;

Галузь штучного інтелекту - напрям діяльності у сфері інформаційних технологій, який забезпечує створення, впровадження та використання технологій штучного інтелекту.

У документі [2] зазначено наступну низку проблем, пов’язаних з розвитком технологій штучного інтелекту, які наразі є в Україні та потребують вирішення. А саме:

- Недостатній рівень поінформованості населення щодо основних перспектив, принципів, аспектів, ризиків та потенційних загроз, пов’язаних з використанням штучного інтелекту, низька цифрова грамотність;
- Недостатність або відсутність правового регулювання у галузі штучного інтелекту а також у сфері захисту персональних даних;
- Недостатній (низький) рівень інвестицій у розроблення штучного інтелекту;
- Низький рівень впровадження технологій штучного інтелекту на підприємствах, господарствах, установах, тощо, що призводить до зниження продуктивності праці та зросту необхідної кількості працівників;

- Низький рівень математичної грамотності випускників закладів середньої освіти, що перешкоджає молодому поколінню продовжити навчання у сфері штучного інтелекту та суміжних з нею сфер.
- Низький рівень якості вищої освіти та освітніх програм, що готують спеціалістів у галузі штучного інтелекту в закладах вищої освіти;
- Відсутність або недоступність програм підвищення кваліфікації для викладачів та спеціалістів закладів вищої освіти для продовження навчання та досліджень у галузі штучного інтелекту;
- Відсутність грантового фінансування та низький рівень інвестицій у розвиток освіти в сфері штучного інтелекту;
- Недостатня кількість наукових робіт та публікацій щодо штучного інтелекту у провідних вітчизняних виданнях;
- Низький рівень кібербезпеки, інформаційної безпеки, особливо у інформаційно-телекомунікаційних системах державних органів, що був спричинений застарілістю автоматичних систем, слабким виявленням інформаційних загроз та відсутністю прогнозування потенційних атак і відповідної підготовки;
- Зростання кількості випадків кіберзлочинів та спроб несанкціонованого втручання в роботу автоматизованих систем та комп’ютерних мереж;
- Бюрократія в системах та структурах надання адміністративних послуг, застарілість механізмів прийняття рішень та роботи з інформацією, її низька якість або недоступність, низький ступінь оцифрованості даних;
- Неоднозначність та складність застосування до систем штучного інтелекту існуючих законодавчих та морально-етичних принципів;
- Загроза зростання рівня безробіття, загострення економічної кризи в результаті активної автоматизації та скорочення робочих місць;

В результаті аналізу даної постанови можна зробити висновок, що основні проблеми та перешкоди розвитку в сфері штучного інтелекту в нашій державі пов’язані з низьким рівнем поінформованості та освіти, недостатністю фінансування та інвестицій а також із застарілими технологіями та бюрократичними механізмами роботи державних установ, що спричиняють відповідні ризики у галузі безпеки.

Для розв’язку зазначених вище проблем та для позиціонування України як держави з розвинутими передовими технологіями у постанові [2] було визначено ряд стратегій розвитку та вдосконалення сфери штучного інтелекту. Після аналізу документа [2] можна зробити певні висновки.

Для вирішення проблем у сфері освіти було зазначено [2] впровадження інноваційних технологій в освітній процес, появу відповідних курсів та освітній програм, які б дозволяли дітям отримувати навички комп’ютерної грамотності, вміння працювати з цифровими даними, знання в сфері захисту даних та кібербезпеки. Також було зазначено впровадження відповідних освітніх компонентів в заклади вищої освіти, створення кафедр та інститутів, фокус яких був би направлений на освіту та дослідження в сфері штучного інтелекту, розробку програми курсів підвищення кваліфікації для викладачів та спеціалістів.

У сфері науки було запропоновано [2] організацію конференцій і форумів з метою підвищення рівня зацікавленості українських дослідників у роботі в цій сфері а також розвитку міжнародної співпраці. Також було зазначено підвищення рівня фінансування та грантової підтримки проектів у сфері штучного інтелекту.

У секторі безпеки було зазначено необхідність створення українських програмних продуктів задля використання державними органами, роботи з цифровими даними, тощо, для зменшення впливу іноземних компаній на роботу українських установ і відповідного зменшення ризику

несанкціонованого доступу до конфіденційних даних. Також була вказана необхідність підвищення рівня підготовки спеціалістів у галузі кібербезпеки, розробки і впровадження відповідних освітніх програм. Іншим важливим пунктом було зазначено вдосконалення правової бази та існуючого законодавства, що регулює використання інформації та притягання суб'єкту до адміністративної або кримінальної відповідальності в разі порушення законів про захист персональних даних та кібербезпеку.

Сьогодні надзвичайно актуальним постає питання застосування новітніх технологій у військовій галузі. У зв'язку з цим у постанові було запропоновано використання штучного інтелекту під час прийняття рішень, управління та командування, передачі інформації, збору та аналізу даних, обробки зображень, карт, знімків, а також для прогнозування ходу бойових дій та їх наслідків.

Для розвитку у економічній сфері було запропоновано [2] використання технологій штучного інтелекту для автоматизації виробничих процесів, підвищення продуктивності, заохочення інвестицій, стимулювання розвитку підприємств шляхом впровадження нових технологій, розвиток IT-сектора.

Для удосконалення області публічного управління у постанові [2] було запропоновано впровадити автоматизацію ряду адміністративних послуг з метою зменшення участі державних службовців та/або співпрацівників а також переходу на цифрову документацію. Також було вказано [2] використання методів штучного інтелекту для оцінки ефективності роботи окремих державних апаратів та установ, проведення аналізу та прогнозування. Це дало б змогу здійснювати моніторинг світових тенденцій та окремих соціальних та економічних явищ задля прийняття оптимальних рішень з урахуванням усіх факторів та чинників, що впливають та стан суспільства та на добробут населення.

У правовому секторі пропонується [2] впровадження штучного інтелекту задля попередження злочинів та небезпечних явищ завдяки методів аналізу і прогнозування, створення програмних продуктів для юридичних консультацій, що, в свою чергу, має підвищити рівень юридичної грамотності населення. Також пропонується створення консультативних додатків для допомоги спеціалістам, які мали б полегшити пошук інформації та допомогли б в задачах прийняття рішень. Ще вказується [2] необхідність створення юридичного механізму, що контролював би процес розвитку і впровадження технологій штучного інтелекту без шкоди для суспільства.

Для вирішення проблем етики було запропоновано [2] створено окремих документів з дотриманням міжнародних норм і стандартів, що прописували б етичні норми, якими мали б керуватися розробники, власники продуктів та користувачі задля безпечного і ефективного користування технологіями, співпраця з міжнародними організаціями та внесення на розгляд ряд законопроектів, що регулювали б політику даних, методи та принципи використання і створення продуктів у галузі штучного інтелекту.

Таким чином, варто зазначити, що технології штучного інтелекту активно розвиваються в Україні і в усьому світі. Тому важливо проаналізувати різноманітні загрози, проблеми та перешкоди, які ми можемо спостерігати, та прийняти ряд заходів щодо покращення стану добробуту в усіх сферах людської діяльності. Впровадження вище вказаних завдань має не тільки розробити майданчик для активного розвитку та досліджень в сфері ШІ, а ще й покращити загальний стан освіти, охорони здоров’я, військового сектору, науки, економіки і тд. шляхом впровадження новітніх технологій.

Актуальність даної роботи полягає в перспективності сфери штучного інтелекту та потреби держави в рішенні вказаних вище проблем.

1.2 Штучні нейронні мережі та їх класифікація

У цьому пункті буде розглянуто визначення та основні принципи будови штучних нейронних мереж, навчання нейронних мереж та його класифікація, а також основні типи нейронних мереж, їх характеристики, особливості, відмінності та головні сфери застосування

1.2.1 Визначення штучних нейронних мереж

Важливою складовою технологій та алгоритмів штучного інтелекту є штучні нейронні мережі. Простими словами [3] їх основну концепцію можна описати як інтерпретацію біологічних нейронних зв’язків, що притаманні людському мозку, у вигляді математичних функцій та алгоритмів, які, в свою чергу, реалізуються засобами програмування при цьому створюючи потужний інструмент для обробки даних. Типовими прикладами задач, які розв’язують штучні нейронні мережі, є задачі класифікації, розпізнавання, аналізу даних та прогнозування. Сьогодні ми можемо спостерігати великий стрибок у розвитку штучних нейронних мереж та їх практичну імплементацію у повсякденне життя, оскільки наразі нейронні мережі використовуються для аналізу даних у фінансовій та адміністративній сфері, для прийняття рішень, діагностування та консультування у медицині, для моделювання, проведення експериментів та прогнозування у науці, в статистиці, в соціології, у промисловості тощо.

Штучні нейронні мережі побудовані за аналогією мозку людини, тобто є послідовністю певних нейронів, що поєднані між собою. Кожне таке з’єднання здатне передавати певний сигнал, тим самим збуджуючи нейрон. Збуджений нейрон може обробляти отриманий сигнал та передавати його у

відповідні сусідні нейрони. Таким чином сигнал може розповсюджуватися по мережі.

Термін «нейрон» [3] був взятий напряму з біології, оскільки це є клітина, функціональна одиниця нервової системи, що здатна приймати, обробляти зберігати та відправляти певну інформацію. У мозку мільярди нейронів з'єднані один з одним відростками, тим самим утворюючи величезну мережу. Проте, варто зазначити, що зв'язки між нейронами бувають різні, тобто, умовно кажучи, час, що потрібен для передачі сигналу від одного нейрона в інший, буде різним в залежності від будови зв'язків, відстані між нейронами та їх взаємного розташуванням. Інколи сигнали згасають, або передаються надто слабкими, що значно ускладнює моделювання системи з математичної точки зору. У штучних нейронних мережах [3] нейрон – це, аналогічно з біологією, певна структурна одиниця мережі, яка здатна виконувати окремі операції та обчислення. Вона отримує на вхід певний сигнал, обробляє його і передає сусідньому нейрону. Штучні нейрони з'єднані зв'язками, що мають один важливий параметр – ваги. Ваги зв'язків застосовуються для моделювання нерівності біологічних зв'язків, оскільки вони здатні збільшувати чи зменшувати силу сигналу.

У штучних нейронних мережах нейрони групуються у шари. Сигнали поступають від вхідного шару до вихідного, при цьому проходячи певну кількість прихованих шарів. Типова структура штучної нейронної мережі зображена на Рис.1.1.

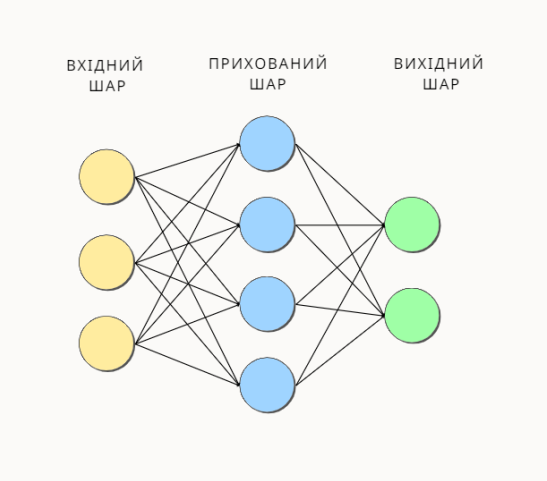


Рисунок 1.1 - Будова штучної нейронної мережі

1.2.2 Навчання нейронної мережі

Для того, щоб нейронна мережа давала правильні результати на виході, необхідно, щоб вона навчалась. Для цього застосовуються коефіцієнти, або корелюючі функції, що впливають на ваги та змінюють їх значення відповідно до вимог задачі. Таким чином, можна сказати, що під навчанням нейронної мережі розуміється [3] пошук певного набору значень вагових коефіцієнтів, що в результаті проходження через зв’язок дозволяють отримати шуканий результат. Проте, варто зазначити, що якщо коефіцієнти ваг будуть визначатися лише на одному прикладі, то нейронна мережа навчиться миттєво, але при лише невеликій зміні вхідних даних результати будуть некоректними, тому для навчання нейронної мережі зазвичай береться велика вибірка даних, а коефіцієнти обчислюються як узагальнюючі для усього набору даних.

Основні види навчання нейронної мережі, їх особливості, відмінності та головні сфери застосування буде більш детально розглянуто у пункті 1.3.

1.2.3 Класифікація нейронних мереж

Наразі не існує окремого алгоритму виявлення належності нейронної мережі до того чи іншого типу, оскільки класифікація може відбуватися за багатьма ознаками: за парадигмою навчання, за типом вхідних даних, за властивістю зв’язків, за моделлю задачі та за сферою використання. Проте, варто розглянути два основних види нейронних мереж: прямого поширення та непрямого поширення, серед них - рекурентним нейронним мережам та класичному багатошаровому персептронові. Також особливу увагу варто наділити загортковим нейронним мережам. Схему класифікацій нейронних мереж зображено на Рис. 1.2.

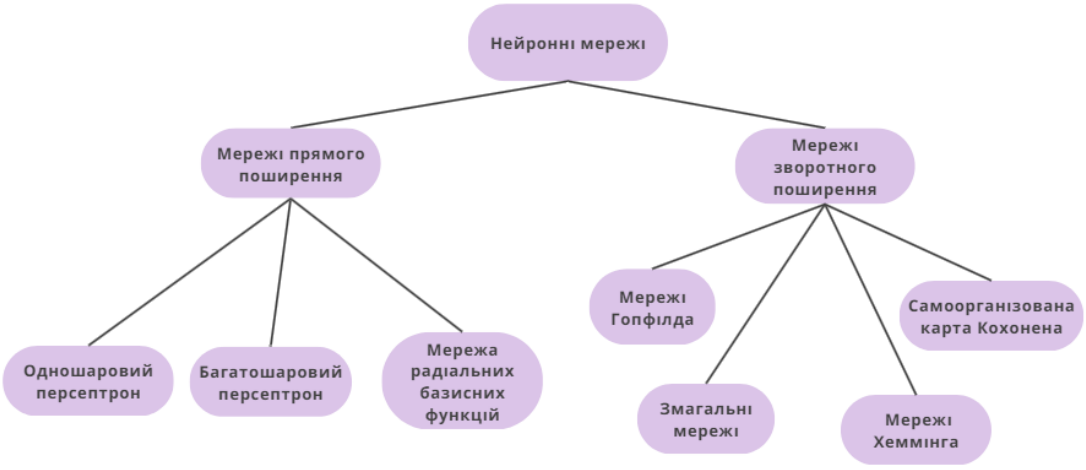


Рисунок 1.2 – Класифікація нейронних мереж

1.2.4 Нейронні мережі прямого поширення

Даним тип нейронних мереж є класичною та звичною нам нейронною мережею, у якій процес поширення сигналів відбуваються в одному напрямку, тобто від вхідного шару, через приховані і до вихідного. Типовим прикладом такої мережі буде традиційний персептрон Розенблатта, що власне і став початком активних досліджень у цій галузі. Такі нейронні мережі широко застосовуються для великої кількості різноманітних задач, проте, деякі завдання вимагали альтернативного підходу, що привело до появи мереж зворотного поширення.

1.2.5 Нейронні мережі зворотного поширення

Мережі зворотного поширення [4] застосовується в більшості для задач, що мають справу з певною сегментацією, наприклад, рукописний текст, певні патерни, слова, мова. Основний його принцип ґрунтується на аналізі груп елементів послідовно, або ж серією різних подій, таким чином створюється граф у часі, що дає змогу системі мати динамічні характеристики, а внутрішня пам’ять дозволяє обробляти непряму послідовність даних, тим самим значно покращуючи процес навчання.

1.2.6 Багатошарові нейронні мережі

Даний тип нейронних мереж [4] є досить старим, проте досі широко розповсюдженим для різного кола задач. Не важко здогадатися, що його архітектура будується на принципі поєднанні декількох шарів, зазвичай це буде вхідний шар, вихідний шар та ряд прихованих шарів. При цьому, кількість нейронів у кожному шарі може бути довільною, але у прихованих шарах зазвичай вона береться однаковою. Зв'язки між шарами будуються за принципом «кожен з кожним», таким чином відбувається взаємодія між усіма нейронами попереднього шару і наступного, тож інформація не втрачається.

Багатошарові нейронні мережі можуть застосовувати як метод прямого поширення, так і метод зворотного поширення, причому на практиці останній використовується значно частіше. Цей метод вимагає подання навчальної вибірки у вигляді пар «вхідні дані» - «правильні вихідні дані». Навчання відбувається наступним чином: вхідні дані проходять через всі шари нейронної мережі і отримані вихідні дані порівнюються з очікуваними. Помилка, що виникає при порівнянні, поширюється назад і корегує відповідні ваги, тим самим навчаючи нейронну мережу.

Цільові задачі даної категорії нейронних мереж зазвичай мають такі вихідні дані, що не залежать від попередніх вхідних даних (тобто від історії), тим самим утворюючи лінійну залежність. Це можуть бути задачі класифікації зображень (з суттєвими відмінностями), обробка текстових даних, тощо. Основною перевагою даного типу нейромереж буде те, що вони надійні та добре досліджені, існує багато готових розробок і фреймворків, що робить досить нескладною реалізацію та інтеграцію їх у вже існуючий програмний продукт. Головний недолік – дані мережі потребують великої навчальної вибірки помічених даних, що може бути проблематичним для

ряду практичних задач. Більш того, багатошарові нейронні мережі не здатні розпізнавати зв’язки у часі, що робить неможливим їх використання для задач з обробки динамічних процесів.

1.2.7 Рекурентні нейронні мережі

Головною відмінністю даного типу нейронних мереж [4] від стандартного персептрону є те, що вони використовують вихідні дані для навчання, тобто замість подання випадково згенерованих даних дані мережі застосовують свій досвіт з попередніх ітерацій, тим самим покращуючи швидкість та якість навчання. Вони також застосовують зазвичай метод зворотного поширення, проте він вимагає ряду модифікацій для роботи з рекурентними зв’язками, оскільки тут виникає ризик відправки помилки на нескінченну кількість ітерацій назад. Проте, загалом, дана модель дуже схожа на стандартну багатошарову нейронну мережу.

Головною перевагою рекурентних нейронних мереж буде їх здатність працювати з динамічними даними. Більш того, хоча даний тип не застосовуються дуже широко на практиці, в нього є великий потенціал для подальших удосконалень. Основний недолік – дані алгоритми слабо досліджені та в них багато точок відказу (single point of failure), що робить досить складною розробку, налаштунки та навчання.

1.2.8 Згорткові нейронні мережі

Даний тип нейронних мереж [5] досить часто застосовується в задачах розпізнавання фотографій, відокремлення конкретної інформації в

зображеннях та відео, аналіз аудіо, мови, тощо. Згорткові нейронні мережі, або CNN, сьогодні дуже розповсюджені, оскільки вони здатні виконувати розпізнавання об’єктів набагато точніше, оскільки в даному випадку враховується двовимірна топологія зображення, що не можна сказати про, наприклад, багатошаровий персептрон. Більш того, даний тип нейронних мереж здатен розпізнавати зображення навіть в умовах невеликих зсувів, поворотів, змін розміру та трансформації об’єктів, що стало причиною широкого практичного застосування CNN, оскільки навіть в умовах постійних змін, як-от відео з камер дорожнього руху, розпізнавання людських облич, вони здатні з великою точністю виконувати поставлені задачі.

При роботі зі звичайними мережами прямого поширення виникали проблеми з обробкою зображень високої точності, оскільки вони містять велику кількість пікселів, що, в свою чергу, суттєво сповільнювало алгоритм навчання та самої роботи. Справа в тому, що звичайні мережі прямого поширення працюють з даними у вигляді векторів, в той час як згорткові нейронні мережі [5] розглядають зображення в цілому, тобто у вигляді масивів матриць чисел.

Головними складовими загорткових нейронних мереж будуть згорткові шари, агрегувальні шари, повноз’єдні шари та шари нормалізації [5]. При цьому, кількість шарів може варіюватися в залежності від специфіки задачі, що дає змогу розробникам бути досить гнучкими та керуватися власними потребами, аніж якоюсь конкретною архітектурою.

Згортковий шар [5] реалізує операцію згортки до результатів з попереднього шару, при цьому вагами ядра згортки виступають навчальні параметри. Пулінговий, або агрегувальний, шар прискорює наші обчислення, оскільки він зменшує розмірність зображення для полегшення роботи наступним шарам. Для цього для окремих блоків зображення застосовуються певна функція, частіше за все береться максимум, або зважене середнє.

Повнозв’язні шари виступають посередниками між нейронами, тобто реалізують передачу даних як у класичних нейронних мережах, в той час як шар нормалізації виконує нормалізацію, тим самим забезпечуючи коректність алгоритму.

Джерелом натхнення для створення даного виду нейронних мереж [3] [4] послугувала зорова кора в головному мозку, яка власне обробляє візуальну складову. Сьогодні згорткові нейронні мережі широко застосовуються в великому спектрі різноманітних задач, а активні дослідження в сфері глибокого навчання дозволяють нам зробити висновок, що цей тип мереж буде дуже актуальним та перспективним наступні роки.

1.3 Машинне навчання. Класифікація

У даному розділі розглянуто визначення терміну «машинне навчання», його основні ознаки та характеристики, а також проаналізовано структуру класифікації машинного навчання.

1.3.1 Визначення машинного навчання

Машинне навчання (ML) – набір алгоритмів та методів в сфері штучного інтелекту, які застосовуються для того, щоб машина або програмний продукт «навчався» за рахунок різних стратегій аналізу великого набору даних, пошуку закономірностей і тд.[6], [7].

Тобто це один з розділів штучного інтелекту, головною метою якого є повна або часткова автоматизація процесу пошуку розв’язків різноманітних задач з аналітики, прогнозування, прийняття рішень тощо. На сьогоднішній

день ML дуже активно розвивається і його застосування можна зустріти в безлічі різних сферах: від банківської справи, маркетингу, бізнес менеджменту до медицини, освіти, промисловості і багато іншого. Сучасний інформаційний простір є дуже великим і потребує потужних механізмів для аналізу великого обсягу даних, що призвело до появи нових напрямків досліджень в цій сфері.

Згідно з джерелом [7], головними складовими машинного навчання вважаються дані, ознаки та власне сам алгоритм.

В цьому контексті дані – вхідна інформація, навчальна вибірка даних, яка потім буде використовуватися для навчання та тестування моделі. Ця вибірка має бути достатнього обсягу, що забезпечить більш ефективне навчання моделі. Дані можуть збиратися та позначатися вручну, або автоматично – це цілком залежить від конкретної задачі, потреб та можливостей розробника. Організація повного набору даних може бути досить довготривалим та коштовним процесом, проте від якості та повноти даних залежатиме точність роботи моделі.

Ознаки можна визначити як ключові особливості даних, їх структуру, зв’язки, залежності, що потім можна використати для визначення стратегії навчання. Сюди відносять саме ті особливості, від яких напряму залежатиме вихідний результат і які будуть застосовуватися для покращення алгоритму, визначення залежностей в самих даних, що в подальшому призведе до покращення точності роботи алгоритму.

Під алгоритмом розуміється власне сам принцип навчання, його побудова та функціонування. Від обраної стратегії побудови алгоритму навчання залежатиме його точність, швидкість та якість отриманих результатів.

1.3.2 Класифікація методів машинного навчання

Класифікація методів машинного навчання може відбуватися за багатьма різними ознаками. За типом навчання розрізняють:

- Навчання з учителем;
- Навчання без учителя;
- Навчання з підкріпленням;
- Напівкероване навчання.

За типом застосування виділяють наступні види:

- Класичне навчання;
- Глибинне навчання;

Класичне навчання застосовує традиційні алгоритми в сфері статистики та роботи з даними, які біли розроблені більше п’ятдесяти років тому. Даний вид доречно застосовувати для задач кластеризації, класифікації, регресії, прогнозування, сегментація тощо – тобто для стандартних задач роботи з даними.

Глибинне навчання, в свою чергу, є досить новим та сучасним підходом, оскільки тут для навчання та розв’язку задач застосовуються нейронні мережі. Типовими прикладами задач, що розв’язуються методами глибоко навчання і нейронних мереж, будуть задачі генерації, розпізнавання, робота з зображеннями, відео, аудіо, мовою, текстом, а також задачі перекладу, прийняття рішень і багато іншого.

Схему обрання доречного виду машинного навчання зображено на Рис. 1.3.

Рисунок 1.3 - Види машинного навчання

Надалі кожен з даних видів навчання буде розглянуто більш детальніше.

1.3.3 Навчання з учителем

Навчання з учителем, або кероване навчання, будується на принципі порівняння вихідних даних нейронної мережі з готовими розв’язками [3]. Воно потребує наявності певного набору прикладів, або позначених даних, що потім використовується для пошуку вагових коефіцієнтів. Класичним прикладом застосування навчання з учителем буде нейронна мережа, що оброблює зображення і виявляє, чи є на фотографії кіт. Для навчання такої мережі необхідно буде подати на вхід велику вибірку фотографій, на кожній з яких буде відмічено присутність на ній кота. Кероване навчання широко застосовується для задач класифікації, розпізнавання образів, обробки звуків

та відео, роботи з рукописним текстом, жестами, мімікою, тощо. Також з його допомогою розв’язуються задачі наближення функції.

1.3.4 Навчання без учителя

Головною проблемою навчання з учителем є потреба у досить великому набору позначених даних, проте, в деяких практичних задачах обробка та підготовка великої навчальної вибірки може видатися надто повільною, складною і коштовною. Навчання без учителя, або ж некероване навчання [3], в свою чергу, не потребує позначених даних взагалі. В таких задачах на вхід подається навчальна вибірка та функція витрат, яка має зменшити область пошуку вагових коефіцієнтів та допомогти визначити залежність між вхідними та очікуваним вихідними даними. Цей тип навчання доречно використовувати в випадку, коли в нас є певні знання про предметну область моделі і є припущення щодо її структури, параметрів, тощо. Типовими прикладами таких задач будуть проблеми кластеризації, певного оцінювання та фільтрування.

1.3.5 Напівкероване навчання

Напівкероване навчання є досить новим, але дуже перспективним видом машинного навчання, оскільки воно виступає як певний симбіоз керованого і некерованого навчання. Основними передумовами його появи стало те, що класичне навчання з учителем вимагає наявності великої вибірки розмічених даних, які часом дуже важко дістати. Більш того, процес маркування даних, тобто присвоєння кожному екземпляру вибірки якоїсь

окремої мітки, може виконуватися вручну або автоматично. Позначення вручну є надійним, але вимагає роботу людини, а отже займатиме багато часу і буде досить коштовним. Автоматичне позначення, в свою чергу, вимагає наявності певного алгоритму класифікації, що для ряду задач є недоступним. Таким чином, пошук та підготовка навчальної вибірки в деяких проектах може займати суттєву частину часу та бюджету, а отже й потребує певної оптимізації.

Напівкероване навчання потребує лише невеликої вибірки позначених даних та великий набір непозначених, які значно легше добути. Існує багато підходів до проектування моделі напівкерованого навчання. Найбільш розповсюджений – генеративний метод буде більш детально розглядатися у наступних пунктах даної роботи.

1.3.6 Навчання з підкріпленням

Цей тип навчання має трошки інший підхід до формулювання задачі, аніж попередні два. Тут не використовується вибірка даних, а є інформаційний агент, що діє в певному середовище і отримує інформацію під час безпосередньої взаємодії з середовищем. Для регулювання дій агенту вводяться «штрафи» та «винагороди» за правильні та неправильні дії, і основною метою задачі є визначення оптимального набору дій, що забезпечуватиме максимальне значення винагород, або мінімізацію сумарних витрат. Даний тип доречно застосовувати для задач, що передбачають відсутність даних про середовище, або наявності великої кількості можливих станів та факторів впливу, що просто неможливо описати під час моделювання.

1.3.7 Класичне навчання

Класичне навчання [6] будується на традиційних принципах та статистичних алгоритмах і застосовується для розв’язання різноманітних задач з прийняття рішень на основі існуючої вибірки даних. Класичне навчання з учителем вимагає існування набору розмічених даних і його найчастіше застосовують для задач регресії та класифікації. Навчання без учителя потребує лише набору даних (без міток) та допомагає розв’язувати задачі узагальнення, кластеризації та пошуку правил.

Класифікація є однією з найбільш розповсюджених задач, до яких застосовується машинне навчання, оскільки вона передбачає поділ даних на певні категорії, або класи, за окремими ознаками. На практиці подібні задачі можуть застосовуватися будь-де, оскільки вони допомагають швидко та ефективно обробити та впорядкувати великі набори даних. Це може бути класифікація електронних листів та їх поділ за рівнем значущості, за сферою впливу, та змістом, тощо. Також це може бути класифікація продуктів, їх зображень, характеристик і тд. Класичними прикладами також можуть слугувати рекомендації в браузері (до цього моменту запити користувача мають класифікуватися для виділення найбільш актуальних посилань), рекламні листи, що також вимагають аналізу та поділу даних клієнтів та різні групи згідно з їх покупками, розробки для аналізу медичних даних, що мають поділити ряд даних про пацієнта на різні групи згідно з відхиленнями від норми тих чи інших показників, або ж відсутності відхилення.

Задачі регресії допомагають спрогнозувати певне значення або показник відповідно до існуючої вибірки даних та заданих ознак. Це може бути прогнозування рівня завантаженості доріг в залежності від часу, погодних умов, пори року і тд., або ж прогнозування стану ринку та рівня попиту на товар через певний час, що значно допомагає бізнес менеджерам

та аналітикам розробити стратегію та план виробництва. Також даний тип навчання можна застосовувати в медицині для прогнозування загального рівня поширення певних видів хвороб серед населення, або ж для передбачення процесу протікання хвороби та одужання конкретного пацієнта.

Під кластеризацією розуміється поділ заданої вибірки об’єктів на окремі кластери, що суттєво відрізняються один від одного. Головною відмінністю даного метода від класифікації буде те, що в задачах класифікації групи поділу, тобто класи, та їх ознаки нам вже завідома відомі і ми намагаємося навчити машину на вибірки про групованих прикладів розподіляти дані на ці класі. В задачах ж кластеризації групи поділу та ознаки невідомі і алгоритм має сам відшукати зв’язки та параметри, за якими можна здійснити поділ даних на кластери.

Задачі пошуку правил передбачають пошук патернів, тобто певних закономірностей, в заданому наборі даних. Яскравим прикладом практичного застосування даного типу задач може бути аналіз запитів у браузері, аналіз переглядів у соціальних мережах та на стрімінгових платформах, розгляд та пошук певних закономірностей у поведінці клієнтів на веб-сторінках з подальшим застосуванням результатів для реклами та рекомендацій.

Отже, класичне навчання є універсальним і широко розповсюдженим типом машинного навчання, що має великий спектр сфер застосування та показує досить високі результати своєї роботи у різних задачах, що й зробило його настільки популярним сьогодні.

1.3.8 Глибинне навчання

Глибинне (глибоке) навчання [8] з’явилось всупереч класичному, оскільки останнє потребує конструювання складних моделей та систем, що в ряді задач потребує врахування великої кількості характеристик та факторів впливу. Більш того, таку модель складно зробити універсальною, оскільки параметри і ознаки можуть варіюватися в різних задачах, та навіть в межах однієї задачі, що потребуватиме постійних змін самої моделі у разі необхідності внести певні зміни в поставлену задачу.

Глибинне машинне навчання [8] – тип машинного навчання, що використовує багатошарові штучні нейронні мережі для обробки даних та розв’язання поставлених задач. Воно може застосовувати різні види навчання та різні типи нейронних мереж. Детальніше про нейронні мережі розглянуто у пункті 1.2.

Даний тип машинного навчання є відносно новим, але він вже дуже добре себе зарекомендував як альтернатива класичному у таким задачах, як розпізнавання мови, звуків, рукописного тексту, зображень високої якості та з невеликими відмінностями. Сьогодні глибинне машинне навчання активно розвивається та має широкі перспективи для подальших досліджень та вдосконалень.

1.4 Генеративно-змагальні мережі та їх застосування

Генеративно-змагальні мережі, або Generative Adversarial Networks (надалі - GAN), є дуже молодою галуззю досліджень в області нейронних мереж та штучного інтелекту. Вперше вони згадувалися у статті Іана

Гудфеллоуа (англ. Ian Goodfellow) у 2014 році [9], а більш детально ця тема була розкрита у спільній роботі його та його колег [10] у тому ж 2014.

Актуальність своїх досліджень вони обґрунтовували рядом проблем, які виникали під час використання моделей глибокого навчання, а саме: основною перевагою глибокого навчання вважається відкриття багатих ієрархічних моделей, які представлятимуть ймовірнісні розподіли за різноманітними типами даних, які досліджуються засобами штучного інтелекту, тобто відео, аудіо, мова, зображення, символи, сигнали тощо. На момент написання статті автори вважали найбільш вражаючим досягненням в області глибокого навчання з дискримінаційними моделями, зазвичай з такими, що зіставляють багатовимірний насичений сенсорний вхід із міткою класу. Головною причиною успіху даного алгоритму вони вважають те, що він базується на принципах зворотного поширення та принципах вилучення, що використовують кусково-лінійні одиниці. Глибокі генеративні моделі не мали особливо великого впливу на розвиток та дослідження в цій сфері, оскільки мали труднощі з апроксимацією багатьох важкорозв’язних імовірнісних обчислень, які власне виникають при оцінці максимальної ймовірності і пов’язаних з нею стратегіях, а також через складність використання переваг кусково-лінійних одиниць у генеративному контексті.

Визначення генеративно-змагальних мереж наводить у своїй статті журналу «Комп’ютерні засоби, мережі та системи» Самолюк Т. А. [11]. Згідно з нею, генеративно-змагальними мережами називаються засоби генеративного моделювання, що застосовують різноманітні методи глибокого навчання, як от згорткові нейронні мережі, глибокі згорткові нейронні мережі, тощо. А генеративним моделюванням, відповідно [11], буде називатися деяке некероване навчальне завдання в машинному навчанні, що передбачає автоматичне вивчення та виявлення певних закономірностей вхідних даних так, щоб модель мала можливість використовуватися для

генерації (створення або виведення) нових моделей, що можна було б одержати з оригінального набору даних.

У своїй роботі 2016 року [12] І. Гудфеллоу наводить наступні аргументи щодо актуальності досліджень у цій сфері:

- Навчання та приклади з генеративних моделей можуть слугувати перевіркою здатності представляти та маніпулювати високовимірними ймовірнісними розподілами, які, в свою чергу, є важливими об’єктами в широкому спектрі областей прикладної математики та інженерії.
- Дуже часто генеративні мережі інтегруються в алгоритми машинного навчання. Наприклад, алгоритми навчання з підкріпленням можна поділити на дві категорії: алгоритми на основі моделі та алгоритми без моделі, що потребуватимуть генеративну модель.
- Третім пунктом зазначаються генеративні моделі часових рядів. Даний тип можна застосовувати для моделювання можливого майбутнього, наприклад, для планування, прогнозування, тощо.
- У концепті алгоритмів навчання з підкріпленням генеративні мережі можуть застосовуватися для навчання в уявному середовищі, тобто помилкові дії агента під час навчання не будуть завдавати йому реальної шкоди. Більш того, генеративні можна застосовувати ще й для зворотного навчання з підкріпленням, оскільки їх можна навчати з відсутніми даними та робити прогнози щодо неповних, пошкоджених або відсутніх вхідних даних.
- Схожий підхід можна застосувати й у напівкерованому навчанні, де більша частина навчальної вибірки не є промаркованою.
- Також генеративні моделі, зокрема GAN, дозволяють алгоритмам машинного навчання працювати з багатьма виходами. Справа в тому, що існують моделі, в яких один вхід відповідатиме декільком

виходам, всі з яких є правильними та прийнятними, наприклад, прогнозування наступного кадру у відео, прогнозування наступної дії. Традиційні методи навчання моделей, як-от мінімізація середньоквадратичної помилки між бажаним результатом і моделлю прогнозованого виходу, не можуть навчати моделі, що можуть створювати кілька різних правильних відповідей.

– І останній пункт, ряд практичних задач дійсно вимагає генерації вибірки деякого розподілу, оскільки її реальне отримання є складним, коштовним або навіть неможливим. Наприклад, якщо модель оброблює конфіденційні дані, то отримання великого набору даних для навчання буде вкрай проблематично, тому в даному випадку легше побудувати модель генерації вибірки на основі невеликого набору даних. Іншим прикладом може слугувати робота із зображеннями, а саме: автоматичне покращення якості зображення, оскільки задля цього модель вимагає наявності більшої кількості інформації, ніж було задано на вході; створення витворів мистецтва, тобто генерація зображення на основі словесного опису або простого ескізу; трансформація зображення у зображення, як-от перетворення знімків на карти, перетворення графічного плану дизайну інтер’єру на дійсне зображення приміщення, тощо.

Генеративно-змагальні мережі сьогодні є досить потужним і сучасним інструментом, що застосовується в багатьох задачах як в чистому вигляді, так і інтегрований у алгоритм машинного навчання.

GAN в чистому вигляді частіше за все застосовуються для генерації нових об’єктів та моделей. Це можуть бути символи або логотипи компаній, дизайни рекламних кампаній. Таким чином, корпорації, що мають постійно оновлювати маркетингову стратегію, можуть зекономити кошти на дизайнерах та маркетологах. Інші приклади застосування – дизайн нових моделей одягу, взуття, меблів та інших товарів. Звичайно, подібні моделі не

будуть новітніми та унікальними – їх скоріш можна буде назвати певною модифікацією, побудованою на основі попередніх моделей. Проте, для деяких компаній, що фокусуються на масовому споживанні, унікальність та екстраординарність моделі не є виключно важливими показниками, якими компанії можуть знехтувати задля економії ресурсів. Аналогічно, GAN широко застосовується у сфері анімації та розробки комп'ютерних ігор – вони використовуються для генерації нових персонажів, середовищ, об'єктів, карт і так далі.

Інші сфера, в якій генеративно-змагальні мережі набули широкої популярності, - це обробка та редагування зображень. Це може бути певне перетворення, як-от автоматичне старіння обличчя, перетворення літньої фотографії на зимову, чорно-білу на кольорову, нічну на денну і так далі. Також GAN може застосовуватися для покращення якості та роздільної здатності фотографії, їх редагування, зміни стилю та дизайну, перетворення ескізу малюнку на картину чи зображення, перетворення авіазнімків місцевості на карти та схеми і багато іншого. Усі вище перераховані задачі потребують не тільки класичної обробки зображень, а ще й генерації нових об'єктів, частин зображення, тобто нової інформації на основі певних даних, що власне і потребує застосування генеративно-змагальних мереж.

1.5 Висновки до першого розділу

У першому розділі було розглянуто основні проблеми та перспективи розвитку галузі штучного інтелекту в Україні, а також було проведено аналітичну роботу щодо теоретичних відомостей в області штучних нейронних мереж, машинного навчання та методів їх класифікації. Також було обґрунтовано мету даної дипломної роботи та актуальність обраної теми досліджень.

РОЗДІЛ 2. ГЕНЕРАТИВНО-ЗМАГАЛЬНІ МЕРЕЖІ

2.1 Топологія генеративно-змагальної мережі

У даному розділі буде розглянуто основні принципи будови та роботи генеративно-змагальних мереж, їх топологію, архітектуру, області застосування та інтеграції у алгоритми навчання, а також методи реалізації мовою програмування Python. Принцип роботи та основи архітектури генеративно-змагальних мереж детально описав у своїй роботі І. Гудфеллоу [12].

2.1.1 Генеративно-змагальні мережі, побудовані за принципом максимальної правдоподібності

Для спрощення дослідження авторами було обрано моделі, що базуються на принципі максимальної правдоподібності [12]. Основна ідея цього принципу полягає в тому, що задана модель забезпечує оцінку розподілу ймовірностей, що є параметризованим деякою змінною θ. Тоді ми вважатимемо правдоподібністю ймовірність того, що модель визначається навчальними даними $\prod_{i=1}^{m} p_{model}\left(x^{(i)};\ \theta\right)$ для набору даних, що містить m навчальних прикладів $x^{(i)}$.

Цей принцип ґрунтується на виборі параметрів для моделі, що максимізує правдоподібність навчальних даних. Для цього автор рекомендує працювати у просторі логарифмів, оскільки в такому представлені приклади будуть додаватися, а не множитися, що в свою чергу значно полегшує розрахунки, оскільки сума спрощує алгебраїчні вирази для похідних

правдоподібності. Таке представлення також значно полегшує програмну реалізацію моделі, оскільки дозволяє уникнути ряду проблем під час обчислень, як-от числова недостатність (надто мала ймовірність), що може бути проблематичною під час роботи з деякими мовами програмування, оскільки більшість стандартних типів даних займають обмежений обсяг пам'яті, а отже й мають певні границі точності.

$$\theta^* = argmax_\theta \prod_{i=1}^{m} p_{model}\left(x^{(i)};\, \theta\right) = argmax_\theta log \prod_{i=1}^{m} p_{model}\left(x^{(i)};\, \theta\right)$$

$$= argmax_\theta \prod_{i=1}^{m} log p_{model}\left(x^{(i)};\, \theta\right) \qquad (2.1)$$

Тут при здійсненні перетворень задіяна властивість логарифму: $argmax_v f(v) = argmax_v log f(v)$ для деякого додатного v, що пояснюється тим, що логарифм є зростаючою функцією і не змінює розташування свого максимуму.

Оцінку максимальної правдоподібності також можна представити як мінімізацію KL-дивергенції між розподілом генерації даних і моделлю.

Дивергенція або розбіжність Кільбака-Лейблера (KL) – це міра, що дозволяє визначити наскільки інформаційна ентропія одного розподілу відрізняється від ентропії іншого розподілу [13].

$$\theta^* = argmin_\theta D_{KL}(p_{data}(x)\big|\big|p_{model}(x;\, \theta)\big) \qquad (2.2)$$

Якби цей параметр вдалося б обчислити точно, то p_{data} належала б до сімейства розподілів $p_{model}(x;\, \theta)$, а модель була б здатна точно відновити p_{data}. В реальності ж, зазвичай ми не маємо прямого доступу до p_{data}, а можемо працювати лише з навчальною вибіркою, що складається з m зразків p_{data}. Ця вибірка використовується загалом для визначення емпіричного розподілу $\hat{p}_{data}$, що розміщує свою масу виключно в наближених до p_{data} m точках. Мінімізація розбіжності KL-дивергенції між $\hat{p}_{data}$ і p_{model}

еквівалентна максимізації логарифмічної правдоподібності навчального набору.

2.1.2 Класифікація моделей, побудованих за принципом максимальної правдоподібності

Моделі, створені за принципом максимальної правдоподібності, можна поділити на ряд підкласів за методами обчислення правдоподібності та її градієнтів, або ж їх апроксимацією. Загалом, виділяють моделі з явною та неявною щільністю.

Для моделей з явною щільністю максимізація правдоподібності відбуватиметься досить просто, оскільки функція щільності підставляється у вираз ймовірності та максимізується. Головною складністю при побудові таких моделей буде власне їх створення з урахуванням всіх особливостей вибірки даних.

Моделі з неявною щільністю ж можна навчити навіть без явного визначення щільності функції. Ці моделі пропонують спосіб навчання за допомогою взаємодії лише з p_{model} або вибірки з неї.

Детальніше про підтипи моделей описано в пункті 2.4 у роботі [12]. Загалом автори виділяють класифікацію, що зображена на Рис.2.1.

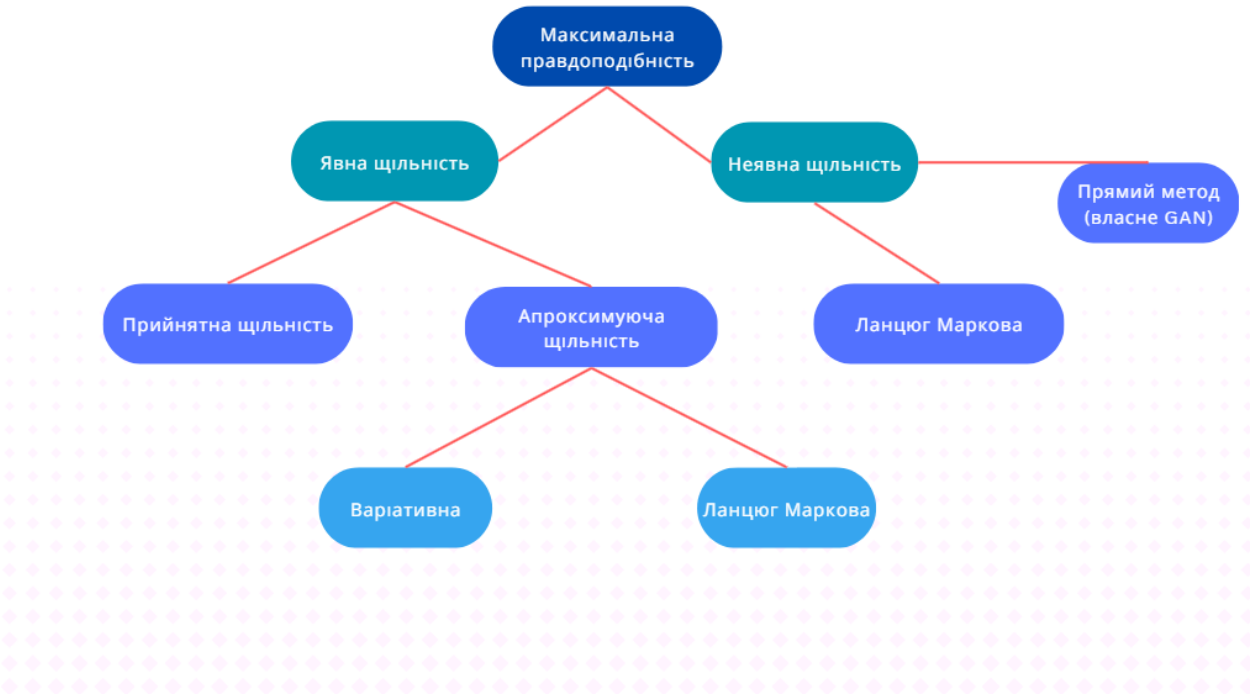


Рисунок 2.1 - Діаграма класифікації моделей

2.1.3 Принцип роботи генеративно-змагальних мереж

Основну ідею генеративно-змагальних мереж можна представити як змагання між двома гравцями: генератором та дискримінатором. Генератор створює зразки, що мають імітувати навчальні дані, а дискримінатор, в свою чергу, перевіряє їх, чи є вони справжніми чи ні. Дискримінатор отримує на вхід дані та поділяє їх на два класи: справжні та підроблені. Таким чином, суть моделі зводиться до того, щоб генератор навчився створювати зразки таким чином, щоб дискримінатор визначав їх як справжні, а для цього вони повинні бути того самого розподілу, що й навчальні дані.

В моделі ці два гравці представлені функціями, кожна з яких є диференційованою як відносно своїх вхідних даних, так і відносно своїх параметрів.

Дискримінатором є деяка функція *D*, яка приймає *x* як вхідні дані та використовує $\theta^{(D)}$ як параметри. Генератор визначається функцією *G*, яка приймає *z* як вхідні дані та використовує $\theta^{(G)}$ як параметри.

І генератор, і дискримінатор мають свої функції витрат. Дискримінатор хоче мінімізувати $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ і він повинен робити це, лише контролюючи $\theta^{(D)}$. Генератор бажає мінімізувати $J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right)$ і повинен робити це, лише контролюючи $\theta^{(G)}$. Оскільки функція вартості кожного гравця залежить від параметрів іншого гравця, які він не контролює, то дану модель легше описати як гру, а не як задачу оптимізації. Розв'язком задачі оптимізації буде (локальний) мінімум, точка в просторі параметрів, де всі сусідні точки мають більшу або однакову вартість. В контексті гри розв'язком буде вважатися рівновага Неша (локальні диференційні рівноваги Неша [14]). Тож, в даному випадку рівновагою Неша буде деякий кортеж $\left(\theta^{(D)}, \theta^{(G)}\right)$, тобто локальний мінімум $J^{(D)}$ відносно $\theta^{(D)}$ і локальний мінімум $J^{(G)}$ відносно $\theta^{(G)}$.

Генератором є деяка диференційована функція *G*. Коли *z* вибирається з деякого простого попередньо заданого розподілу, *G(z)* дає вибірку *x* взяту з p_{model}. Як правило, для представлення *G* використовуються глибокі нейронні мережі. Наприклад, можна розділити *z* на два вектори $z^{(1)}$ і $z^{(2)}$, потім подати $z^{(1)}$ як вхідні дані для першого рівня нейронної мережі та додати $z^{(2)}$ до останнього рівня нейронної мережі. Якщо $z^{(2)}$ - гаусівський, то можна зробити висновок, що *x* – має умовний гаусівський розподіл при заданому $z^{(1)}$. Інша розповсюджена стратегія - застосування домішки або мультиплікативного шуму до прихованих шарів нейронної мережі. Загалом ми бачимо, що обмежень щодо дизайну мережі генератора дуже мало.

Навчальний процес даного типу моделей складається з серії одночасних SGD (Stochastic Gradient Descent). На кожному кроці обираються два невеликих набори: набір значень x із вибірки даних та набір значень z, взятих із попередніх даних. Потім виконуються два кроки градієнтного спуску одночасно: один оновлює $\theta^{(D)}$ для мінімізації $J^{(D)}$, а інший оновлює $\theta^{(G)}$ для зменшення $J^{(G)}$. В обох випадках можна використовувати будь-який алгоритм оптимізації на основі градієнтного спуску.

У своїй роботі автори розглянули та порівняли наступні можливі функції вартості:

- Функція вартості лише дискримінатора;
- Мінімакс (додаємо залежність функції від генератора);
- Евристична, ненасичена гра;
- Гра максимальної правдоподібності;
- Їх більш детальний аналіз було наведено у пунктах 3.2.1 – 3.2.6 [12].

2.2 Структурно-параметричний синтез генеративно-змагальних мереж. Постановка завдання

У цьому розділі буде проаналізовано структурно-параметричний синтез генеративно-змагальних мереж, принципу їх роботи. Також буде проведено аналіз існуючих засобів реалізації та проведено експерименти.

2.2.1 Постановка завдання структурно-параметричного синтезу генеративно-змагальних мереж

Струтурно-параметричний синтез нейронної мережі – етап розробки, під час якого відбувається формування певних топологічних зв’язків, тобто обирається її структура, тип зв’язків, типи нейронів, їх кількість та побудова, алгоритм оптимізації, функцій вартості, функція втрат, тощо. Також в цей етап відбувається навчання моделі, тобто пошук вагових коефіцієнтів та параметрів функції активації, при яких мережа буде працювати найбільш ефективно і давати найменший відсоток похибки.

Головним завданням структурно-параметричного синтезу генеративно-змагальних мереж буде:

1. Визначення типу нейронної мережі генератора та дискримінатора;
2. У випадку побудови багатошарових НМ – визначення кількості шарів та кількості нейронів на кожному з них. Якщо нейронів буде недостатньо, то НМ не буде мати достатніх апроксимаційних здібностей задля розв’язку поставленої практичної задачі, в той час як надто велика кількість нейронів може призвести до появи проблеми перенавчання, що може стати причиною втрати моделлю своїх здібностей;
3. Визначення типів функцій активації та алгоритму оптимізації.
4. Визначення принципу навчання НМ;
5. Власне, сам процес навчання, що дозволить визначити такі значення вагових коефіцієнтів, при яких модель буде надавати найбільш оптимальні та точні результати.

Варто зазначити, що на даний момент не існує стандартних методів проведення структурно-параметричного синтезу нейронних мереж. В деяких випадках структура моделі керується постановкою практичної задачі, тому

вона визначається аналітично і буде залежати від показників багатьох зовнішніх і внутрішніх факторів, фізичних властивостей, тощо.

Для деяких практичних задач це неможливо, тому вони потребують певного математичного апарату для пошуку вище згаданих параметрів. Це можуть бути задачі оптимізації, що дозволятимуть розробникам знайти оптимальний варіант значень параметрів, що дозволить максимізувати складність моделі при прийнятній складності, а також дозволить моделі бути максимально стійкою відносно можливих змін у вибірці даних.

Також деякі розробники застосовують експериментальний метод для проведення структурно-параметричного синтезу моделі, що полягає у тестуванні найбільш потенціальних стратегій та параметрів на певному наборі даних та порівнянні результатів. В залежності від типу задачі розробники зможуть прийняти рішення щодо пріоритетності тих чи інших метрик і зробити рішення відносно обраних параметрів. В деяких випадках, як-от в задачах медичного напрямку, точність моделі має найвищий пріоритет, тому розробники будуть обирати ті параметри, що мінімізують похибку, навіть за умови високих показників складності, часу навчання та розмірів навчальної вибірки, в той час як, наприклад, для моделі генерації зображень похибка може бути дещо більшою, проте швидкість роботи моделі має бути мінімальною.

Таким чином, в результаті структурно-параметричного синтезу НМ розробники мають повне уявлення про структуру конкретної моделі, її параметри, топологію, принцип навчання і так далі, що буде керувати подальший процес планування роботи, розробки, навчання і тестування.

2.2.2 Постановка завдання

У наступних пунктах цього розділу буде здійснено огляд прикладу програмної реалізації генеративно-змагальної мережі, створеним за допомогою мови програмування Python. Як джерело інформації було взяту статтю «GAN: how to build your first model» [19] платформи «Real Python». У цьому розділі буде розглянуто алгоритми побудови генеративно-змагальних мереж, їх теоретичне підґрунтя, а також практичну реалізацію двох задач: задачу генерування пар (x, y), де $y = sinx(x)$ на проміжку $[0; 2\pi]$ та задачу генерування зображень рукописних цифр.

Таким чином, завдання буде полягати в наступному: аналіз вище згаданих алгоритмів та засобів практичної реалізації, а також практична реалізація та проведення експериментів, аналіз результатів.

2.3 Огляд методів побудови генеративно-змагальних мереж

Для реалізації алгоритму генеративно-змагальних мереж було обрано мову програмування Python з використанням бібліотек PyTorch [21], torchvision, math [22] та mathplotlib в он-лайн середовищі розробки Colab [28]. В цьому пункті буде більш детально розглянуто дані інструменти, їх особливості та переваги, а також буде здійснено огляд інших методів побудови генеративно-змагальних мереж та буде обґрунтовано вибір вище згаданих інструментів.

2.3.1 Мова програмування Python

Python – одна з найпопулярніших сьогодні мов програмування [20]. Вона широко застосовується майже в усіх сферах IT індустрії і не тільки. Особливо перспективною вона виявилася в задачах розробки програмних додатків, мобільних додатків, у сфері Data Science, тобто для роботи з даними, в аналітиці, в задачах штучного інтелекту та машинного навчання.

Розроблена в 1990 році голландським програмістом Гвідо ван Россумом мова програмування Python є багатоцільовою і багатоплатформовою, а лаконічний та інтуїтивно зрозумілий синтаксис робить її легкою для навчання та розуміння, що і стало одною з причин такої широкої популярності даної мови.

Головними перевагами Python є швидкість та універсальність. Її можна застосовувати для розв'язання дуже широкого спектру різноманітних задач на будь-який платформах та операційних системах. Також дана мова є досить легкою і гнучкою, а велика кількість готових і безкоштовних бібліотек та фреймворків дозволяє розробникам створювати програми для розв'язку складних задач написавши лише декілька рядків. Така доступність і зрозумілість робить мову програмування Python однією з найбільш широко вживаних в контексті задач штучного інтелекту та машинного навчання.

Головним недоліком даної мови програмування в контексті роботи з даними можна вважати те, що Python є інтерпретованою мовою, що приводить до досить низької швидкості роботи програм. В деяких випадках це компенсується лаконічним синтаксисом і малими розмірами самого коду, що дозволяє зекономити пам'ять та час на компіляції. Проте, в задачах, що вимагають обробки великого обсягу даних, Python може бути значно повільнішим, ніж компільовані мови програмування, як-от C++ чи Golang.

Незважаючи на недоліки, в якості основного інструменту розробки в даній дипломній роботі було обрано саме мову програмування Python, оскільки вона є лаконічною, доступною, гнучкою та розповсюдженою, що робить її легкою для навчання і дослідження.

2.3.2 Бібліотеки PyTorch та math

Як було зазначено раніше, однією з переваг мови програмування Python є наявність великої кількості готових бібліотек, що допомагають розв’язувати різноманітні задачі в багатьох сферах. В задачах штучного інтелекту та машинного навчання найбільш популярними інструментами є бібліотеки PyTorch та math.

PyTorch – досить сучасна та лаконічна бібліотека машинного навчання, розробку якої започаткували працівники компанії Facebook ще у 2012 році [21]. Незважаючи на те, що дана бібліотека працює досить низькорівнево, вона містить окремі модулі, що дозволяють зручно працювати з даними технологіями, наприклад torch.nn для побудови моделей нейронних мереж, torch.utils.data для асинхронної роботи з даними, torchvision для швидкої та зручної роботи із зображеннями. Ця бібліотека є дуже популярною серед розробників, оскільки вона є швидкою і ефективною, надає різні рівні абстракції, проте не обмежує користувача в якійсь певній парадигмі і є надзвичайно гнучкою, що дозволяє використовувати її для розв’язку великої кількості різноманітних задач.

Math є однією з найвідоміших та найбільш важливих бібліотек мови програмування Python [22]. Вона надає широкий математичний функціонал, а саме: основні математичні функції, константи та інші операції по роботі з числами, змінними, функціями і тд.

Отже, ці основні бібліотеки будуть головним інструментом під час роботи над практичною реалізацією алгоритму.

2.3.3 Середовище розробки Colab

Правильно обране середовище розробки може значно пришвидшити і покращити роботу, оскільки середовище також є важливою складової набору інструментів. Зазвичай алгоритми машинного навчання потребують досить великої потужності, що не завжди можна реалізувати на особистому комп’ютері. Більш того, робота на своїй машині вимагає ряду спеціальних налаштунків, що інколи також досить складно організувати. Саме тому для практичної реалізації даного алгоритму було обрано он-лайн середовище розробки Colab.

Colabarotory, або Colab, є зручним інструментом, що дозволяє писати та запускати Python код у браузері, не потребуючи при цьому ніяких додаткових завантажень або конфігурацій. Код можна створювати у файлах-блокнотах і одразу ж запускати його, а зручний інтерфейс дозволяє ефективно працювати, будувати графіки та оформлювати блокнот відповідно до вимог розробника [28].

2.3.4 Огляд інших методів побудови генеративно-змагальних мереж

В залежності від постановки задачі та вподобань розробників та аналітиків для побудови генеративно-змагальних мереж можуть використовуватися й інші методи та інструменти.

З точки зору мови програмування, все ж таки Python залишається найбільш розповсюдженою та широко вживаною. Проте, якщо розробник вважає за потрібним працювати на більш низькому рівні, він може застосовувати C++ [23] або C [24]. Звичайно, це буде вимагати більше часу та досвіду роботи, оскільки інтерфейс даних мов програмування не дуже зручний для роботи з нейронними мережами, проте такий підхід дозволить значно пришвидшити процес навчання і може зробити модель більш точною і потужною.

Якщо ж зупинитися на мові програмування Python, то в ній існує багато зручних бібліотек та фреймворків, що дозволяють легко створювати моделі та тестувати нейронні мережі. Окрім згаданої вище бібліотеки Pytorch, існують також бібліотеки keras, scikit-learn та tensorflow.

Keras [25] – це фреймворк для побудови моделей глибинного навчання. Він створений на основні платформи машинного навчання TensorFlow [26] і був розроблений з акцентом на експериментування та проектування моделей. Він досить потужний і гнучкий, проте набагато складніший ніж Pytorch, оскільки працює на більш низькому когнітивному рівні, чим самим вимагає більше часу та досліджень для проектування моделі.

TensorFlow [26] – комплексна платформа, що дозволяє будуванти та розгортанти моделей машинного навчання. Вона бралася за основу написання більшості бібліотек машинного навчання у Python, тому вважається фундаментальною. Деякі розробники віддають перевагу використанню лише даної бібліотеки, оскільки вона не обмежує їх у певних зафіксованих структурах даних і дає більший простір для моделювання та експериментів, проте даний підхід вимагає більше часу та уваги до компонентів нейронної мережі, їх складових та будови.

Scikit-learn [27] – ще одна бібліотека Python для машинного навчання, побудована на пакетах numpy та scipy. Вона часто використовується дата

аналітиками та інженерами, оскільки допомагає будувати зручні моделі з точки зору взаємодії з даними, їх представленням, модифікацій та аналізу. Вона не така гнучка як Tensorflow і keras, проте з перспективи аналітики є чудовим інструментом для роботи з даними та швидкого розв’язку великої кількості задач.

Таким чином, бібліотека Pytorch була обрана саме через її інтуїтивну зрозумілість та простоту, а також через гнучкість моделювання та експериментування, яку вона надає розробникам. Проте, варто зазначити, що вона чудово співпрацює з бібліотекою TensorFlow, що можна використати при укладенні моделі та під час проведення експериментів з її структурою та параметрами.

2.4 Алгоритм структурно-параметричного синтезу генеративно-змагальних мереж

Алгоритм структурно-параметричного синтезу генеративно-змагальних мереж було детально розглянуто у статті [19]. Згідно з нею, задля програмної реалізації моделі даного типу нейронних мереж необхідно спершу розробити дві її складові: генератор та дискримінатор.

Головною задачею генератора буде оцінка розподілу ймовірностей реальної, тобто правильної, вибірки для подальшої генерації аналогічних об’єктів з таким самим розподілом. Дискримінатор, в свою чергу, має оцінювати ймовірність «реальності» зразку, тобто ймовірність його походження з заданої дійсної вибірки, а не з генерованої. Таким чином, дискримінатор вказує степінь схожості згенерованого зразка на очікувані.

Для більш детального розгляду алгоритму авторами статті [19] було обрано наступний приклад: створення генеративно-змагальної мережі, яка

буде генерувати пари (x, y) , де $x \in [0; 2\pi]$ та $y = \sin(x)$. Графік даного набору даних зображено на Рис. 2.2.

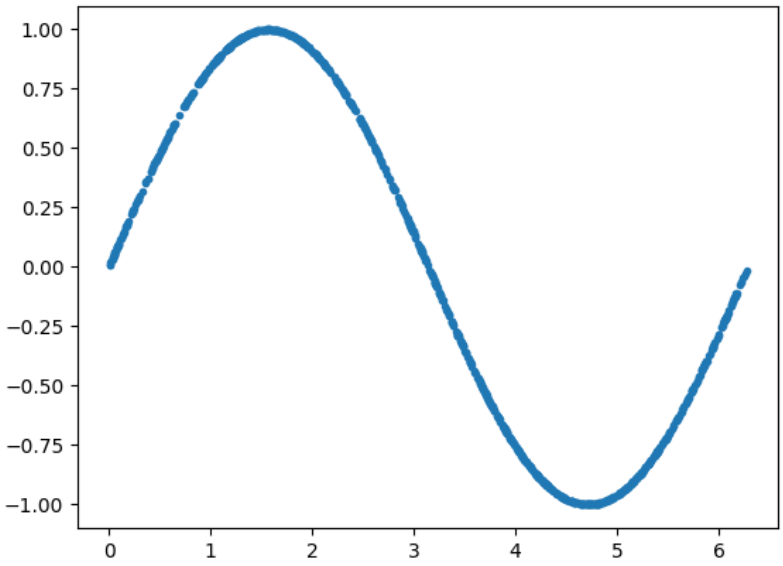


Рисунок 2.2 - Графік заданого набору даних

Робота мережі відбуватиметься наступним чином: на вхід до генератора поступатиме випадкові дані, а він, в свою чергу, матиме згенерувати дані, схожі на очікувані. В даному прикладі на вхід будуть подаватися випадкові пари (x_1, y_1), а генератор повинен згенерувати пари з тим самим розподілом, що й (x, y).

Структура генератора може бути довільною: це може бути згорткова нейронна мережа, багатошарова нейронна мережа, або ж будь-який інший тип, який розробник вважатиме найбільш ефективним.

На вхід до дискримінатора подаються дані або з навчальної вибірки, або зі зразків, створених генератором і головним його завдання буде оцінка «реальності» зразку. Таким чином, результатом роботи дискримінатора після обробки одного зразку буде скаляр з проміжку від 0 до 1, де 1 відповідатиме дійсним даним, а 0 – штучним.

Навчання моделі в даному прикладі буде будуватися за принципом мінімаксу, тобто дискримінатор буде прагнути мінімізувати похибку при

оцінці ймовірності належності зразку до дійсної вибірки, а генератор буде прагнути максимізації ймовірності того, що дискримінатор зробить помилку.

Піж час навчання параметри дискримінатора оновлюються задля мінімізації функції втрат. Після того, як параметри дискримінатора оновились, відбувається навчання генератора, що ставить за мету максимізацію похибки дискримінатора. При цьому процесі параметри дискримінатора не змінюються. Таким чином, при покращенні роботи генератора ймовірність «реальності» зразків буде прямувати до 1. Схеми алгоритму навчання подано на Рис. 2.3 та Рис. 2.4 [19]:

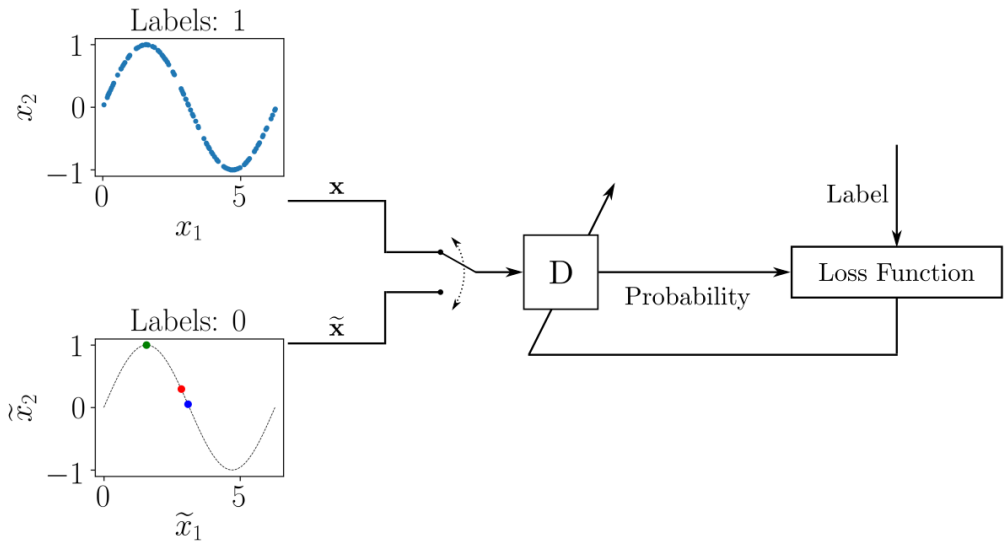


Рисунок 2.3 - Схема навчання дискримінатора [19]

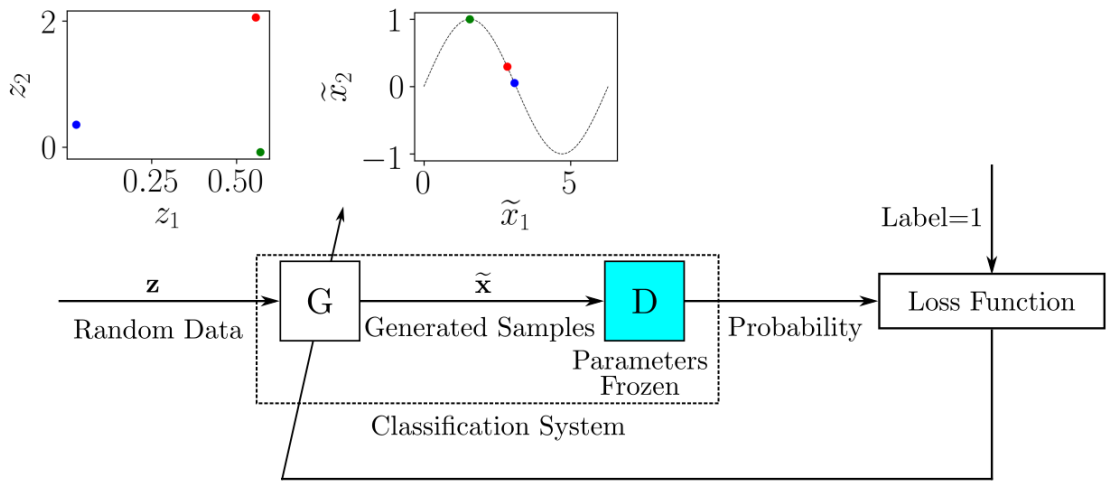


Рисунок 2.4 - Схема навчання генератора [19]

2.5 Перевірка алгоритму на практиці

В цьому розділі буде розглянуто приклад програмної реалізації [19] розв'язку двох задач:

1. Генерація пар (x, y) , де $x \in [0; 2\pi]$ та $y = \sin(x)$ (Пункт 2.5.1).
2. Генерація зображень цифр, що написані рукописним шрифтом (Пункт 2.5.2).

Обидва розв'язки були реалізовані мовою програмування Python в середовищі розробки Colab.

2.5.1 Реалізація задачі 1

Приклад програмної реалізації було детально описано у статті [19]. В ній для проектування моделі було застосовано бібліотеку PyTorch та її

модуль nn (нейронні мережі), для обчислення числа π та функції sin – бібліотеку math, для побудови графіків - бібліотеку mathplotlip.

В якості навчальної вибірки було згенеровано масив з 1024 парами (x, y) , де $x \in [0; 2\pi]$ та $y = \sin(x)$. Графік навчальної вибірки зображено на Рис. 2.5.

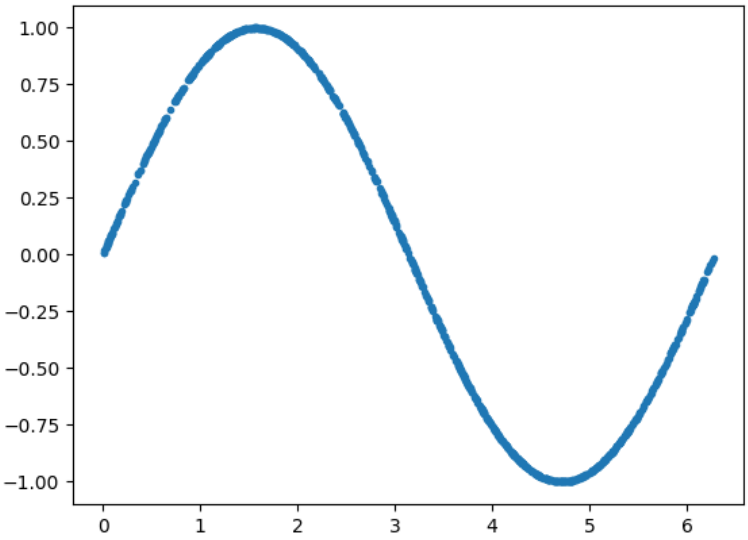


Рисунок 2.5 - Графік навчальної вибірки

Для кращої організації навчальної вибірки було створено об’єкт torch.utils.data.DataLoader, що перемішував та групував зразки по 32 екземпляри.

Дискримінатор та генератор реалізовувалися як класи, побудовані на основі nn.Module з бібліотеки PyTorch. Дискримінатор в даному випадку – це модель з двовимірним входом та одновимірним виходом, що буде отримувати пари даних з навчальної вибірки, або з генератора, та буде повертати ймовірність реальності даної пари. В нашому випадку дана модель буде мати вхідний шар з 2 нейронів, перший прихований шар з 256 нейронів та з функцією активації ReLu, другий та третій приховані шари будуть складатися з 128 і 64 нейронів відповідно з функцією активації ReLu, а вихідний шар буде одновимірним з функцією активації сигмоїда. Після

першого, другого та третього прихованих шарів застосовується dropout у розмірі 0,3, щоб уникнути overfitting.

Клас генератора будується аналогічним чином. В даному випадку це буде модель з двовимірним входом, де на вхід будуть подаватися випадкові пари чисел, та двовимірним виходом, де в результаті будуть подаватися пари чисел, що претендують на схожість з оригінальною вибіркою. Вона також буде мати два приховані шари з 16 і 32 нейронами відповідно та з функцією активації ReLu.

Надалі необхідно задати параметри навчання. В даному прикладі кількість ітерацій буде взята як 300, швидкість навчання (learning rate) визначено як 0.001, а функцією втрат буде бінарна крос-ентропійна функція (BCELoss).

Автори рекомендують використовувати бінарну крос-ентропійну функцію, оскільки вона допомагає розв’язати задачу бінарної класифікації, що доречно для навчання дискримінатора, а також її зручно використовувати для навчання генератора, оскільки він передає свій вихід до дискримінатора, який забезпечує бінарний спостережний вихід. Оптимізація вагів буде реалізована за допомогою функції torch.optim, де в даному випадку використовується алгоритм Adam.

В решті решт, саме навчання реалізовується як цикл, в якому навчальна вибірка подається на вхід до наших моделей і ваги оновлюються задля мінімізації функції втрат [19].

Після завершення навчання мережа здатна генерувати пари. Графік результатів зображено на Рис. 2.6. На ньому можна побачити, що більшість з точок знаходяться на кривій синусоїда, отже, вони відповідають шуканому розподілу. Для $x \geq 5{,}5$ результати дещо погіршуються, проте похибка є все ж таки не дуже великою. Отже, мережа працює досить непогано. Код даної реалізації можна знайти за посиланням у середовищі Colab [28].

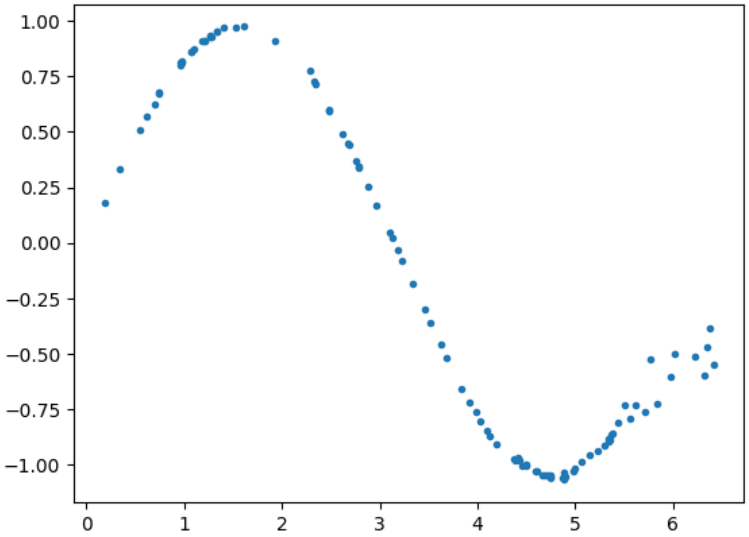


Рисунок 2.6 - Графік згенерованої вибірки

2.5.2 Реалізація задачі 2

У даній задачі буде розглянуто практичну реалізацію побудови мережі, що генерує зображення цифр рукописним шрифтом [19]. Для цього буде використовуватися модуль torchvision з бібліотеки PyTorch, а також набір даних з MNIST. Такі набори даних складають з пікселів 28x28, що утворюють чорно-білі зображення цифр від 0 до 9 рукописним шрифтом. Таким чином, замість того, щоб вручну створювати навчальну вибірку, вона буде згенерована бібліотекою. Приклад зображень з навчальної вибірки зображено на Рис. 2.7.

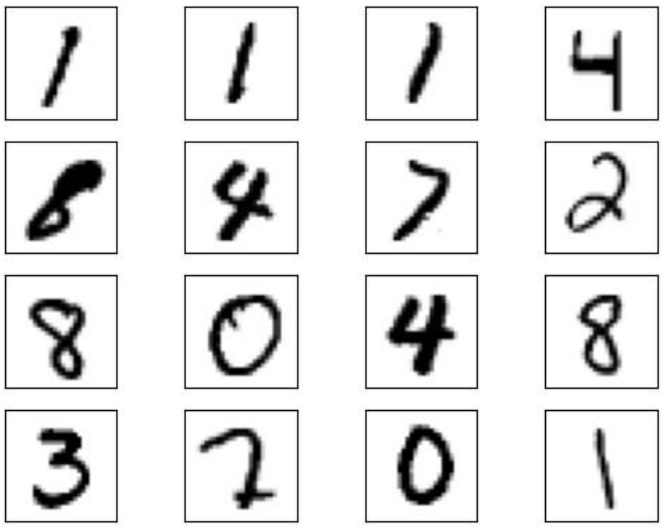


Рисунок 2.7 - Приклади з навчальної вибірки

Моделі генератора і дискримінатора будуються схожим чином, як і в задачі 1. В даному прикладі дискримінатором буде багатошарова нейронна мережа, що отримує на вхід піксельне зображення 28 на 28 і повертає на вихід ймовірність того, що зображення реальне. Задля полегшення проектування зображення буде подано як вектор з 784 елементів.

Генератор в даному випадку повертатиме більш складний тип даних, тому необхідно збільшити розмірність даних, що подаються на вхід. В нашому випадку це буде 100-вимірний зразок, який після обробки буде перетворюватися на вектор з 784 коефіцієнтів. Ще одна відмінність від класу генератора з попереднього прикладу – тут вихідний шар матиме функцію активації як тангенс гіперболічний, оскільки всі результуючі коефіцієнти мають бути в інтервалі від -1 до 1, щоб відповідати нормалізованій вибірці.

Навчання моделі відбуватиметься так само, як і в задачі 1, єдина відмінність – кількість ітерацій зменшиться до 50 задля економії часу. Реалізація циклу навчання також будується аналогічно. Час кожної ітерації навчання – близько 2.5 хвилин, а навчання за всією вибіркою в даному експерименті зайняло трохи більше двох годин.

Після завершення навчання дана нейронна мережа здатна генерувати зображення як на Рис. 2.8. Деякі цифри вийшли трошки «зашумленими», проте навіть їх можна зрозуміти, в той час як більшість зразків вийшли досить реалістичними. Отже, експеримент можна вважати вдалим. Вихідний код даного прикладу можна побачити за посиланням [29].

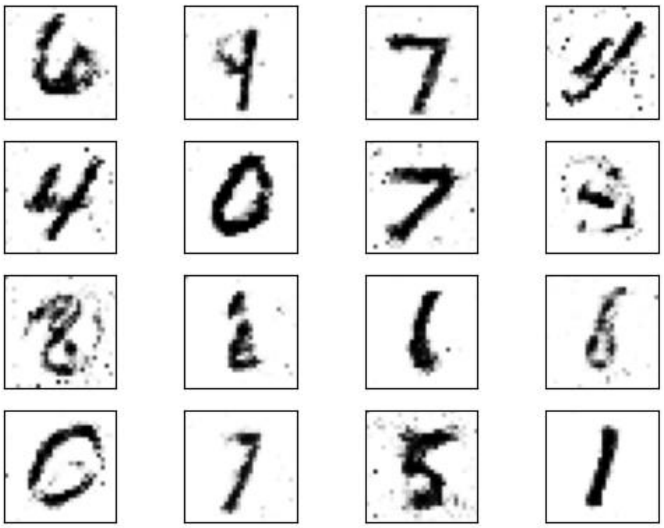


Рисунок 2.8 - Результати, згенеровані нейронною мережею

2.5.3 Висновок

Отже, в даному розділі було розглянуто практичне застосування даного алгоритму для розв'язання двох задач: генерування пар чисел (x, y) , де $x \in [0; 2\pi]$ та $y = \sin(x)$ та генерація зображень цифр від 0 до 9 рукописним шрифтом. Програмна реалізація відбувалася мовою програмування Python в середовищі розробки Colab згідно з методичними рекомендаціями та інструкцією з статті [19]. Навчання мережі для задачі 1 зайнято близько 4 хвилин, а для задачі 2 – більше 2 годин. Така різниця в часі пов'язана зі складністю даних – в задачі 2 це зображення, що складаються з 784 пікселів, а в задачі 1 – пари чисел. В результаті, обидві нейронні мережі показали

результати досить високої точності, що свідчить про те, що алгоритм було реалізовано правильно.

2.6 Висновки до другого розділу

У Розділі 2 було проведено структурно-параметричний синтез генеративно-змагальних мереж, було досліджено та описано їх структуру, топологію, будову та принцип функціонування. Була проведена робота з теоретичними джерелами та науковими роботами.

Також було проведено аналіз існуючих методів та інструментів практичної побудови генеративно-змагальних мереж, було оцінено їх основні переваги та недоліки та було обрано основні інструменти для подальшої роботи.

У даному розділі також було створено алгоритм роботи генеративно-змагальної мережі, який було реалізовано на практиці задля розв’язку двох експериментальних задач. Було одержано гарні результати та зроблено висновки.

РОЗДІЛ 3. МОДИФІКАЦІЯ МЕТОДІВ СТРУКТУРНО-ПАРАМЕТРИЧНОГО СИНТЕЗУ ГЕНЕРАТИВНО-ЗМАГАЛЬНИХ МЕРЕЖ. ПРАКТИЧНА РЕАЛІЗАЦІЯ

В даному розділі буде проведено аналіз структурно-параметричного синтезу генеративно-змагальних мереж та запропоновано і описано алгоритм інтеграції генеративно-змагальних мереж у напівкероване навчання.

Також буде проведено огляд розробленої програмної реалізації розв’язку задачі напівкерованого навчання з використанням генеративного методу. Буде здійснено порівняльний аналіз основних показників роботи розробленої моделі та звичайної моделі керованого навчання для тієї ж самої задачі, зроблено відповідні висновки. Також буде проведено огляд існуючих проектів, стартапів та технологій, які застосовують генеративно-змагальні мережі та/або напівкероване навчання.

3.1 Аналіз запропонованого алгоритму структурно-параметричного синтезу генеративно-змагальних мереж

Експериментальні задачі з Розділу 2 дали змогу побачити приклад роботи стандартної генеративно-змагальної мережі, оцінити її основні якості, особливості побудови, переваги та недоліки. Також дослідження на тему функціональних характеристик та потенційних стратегій вдосконалення генеративно-змагальних мереж були детально описані у статтях Джонатана Ху (Jonathan Hui) “Why it is so hard to train GAN” та “Ways to improve GAN performance” на он-лайн платформах “Medium” та “Towardsdatascience” [25][26].

Основними недоліками можна виділити наступні пункти:

1. Навчання моделі займає дуже багато часу (всього 50 епох у задачі 2 вимагали близько 2 годин роботи), що для більш складних задач, що вимагатимуть великої кількості епох задля покращення якості і точності, може стати досить суттєвою технічною проблемою, оскільки потребуватиме час та ресурси;
2. Колапс режимів (mode collapse) - тобто генератор «зациклюється» на лише окремих класах, або режимах, і перестає навчатися далі для всієї вибірки [25];
3. Модель може стати дуже нестабільною;
4. Дисбаланс між генератором і дискримінатором – оскільки мережа "змагальна", то якщо один з гравців є суттєво сильнішим, то другий буде завжди йому програвати і перестане навчатися;
5. Висока чутливість до гіперпараметрів.

Таким чином, задля того, щоб підвисити точність роботи моделі та уникнути вище перерахованих потенційних недоліків у своїх роботах [25][26] автор розглянув ряд можливих модифікацій, застосування яких може призвести до покращення роботи моделі.

В статті [26] було зазначено наступні стратегії:

1. Заміна функції витрат на ту, що буде краще відповідати оптимізаційним вимогам конкретної задачі.
2. Введення системи штрафів для функції витрат, що забезпечить більш чітке дотримання обмежень.
3. Уникання виникнення явищ overconfidence та overfitting.
4. Пошук кращих способів оптимізації моделі.
5. Додавання міток.

Незважаючи на те, що стаття [26] була написана ще в 2018 році, дані методи оптимізації й досі залишаються одними з найбільш ефективними та

широко розповсюдженими, проте автор звертає увагу на активні дослідження в області генеративно-змагальних мереж, що спричиняють появу все більшої кількості нових методів, алгоритмів та модифікацій.

3.1.1 Балансування генератора і дискримінатора

Генератор і дискримінатор завжди перебувають в стані змагання, тому суттєвий дисбаланс між ними може вивести з ладу всю модель. Таким чином, вдале балансування цих складових моделі може значно покращити її роботу, підвищити ефективність навчання, покращити точність та додати стабільності. Проте, методи балансування не завжди є простими, оскільки різні дослідники мають різні погляди на ефективність тих чи інших методів. Дехто рекомендує контролювати фіксоване співвідношення між кількістю ітерацій градієнтного спуску на генераторі та дискримінаторі, дехто вважає потрібним застосування динамічних метрик.

З іншої точки зору, ряд дослідників вважають непотрібними такі дії, оскільки якщо дискримінатор добре навчився, то він буде давати гарні відповіді генератору, тим самим навчаючи його. Більш того, частіш за все генератор відстає від дискримінатора, а балансування може бути надто складною задачею. Альтернативним рішенням може слугувати пошук такої функції витрат, що не має близького до нуля градієнта у випадках коли генератор працює погано. [26]

3.1.2 Функції вартості

Існують багато дискусій серед досліджень стосовно ефективності та доречності використання тої чи іншої функції вартості, проте можна точно сказати, що кожна з них досить непогано себе зарекомендувала у певних задачах, тому вибір функції вартості має ґрунтуватися на цільовій задачі та вимогах до неї.

3.1.3 Кількість шарів та їх представлення

Зміна кількості шарів генератора та дискримінатора може суттєво вплинути на швидкість та якість роботи моделі. Для покращення роботи можна спробувати застосувати стратегію збільшення кількості нейронів на кожному шарі та зміну функції активації з урахуванням потреб шару.

Також варто зазначити можливий перехід до згорткових шарів, що може значно збільшити точність роботи моделі.

3.2 Алгоритм генеративно-змагальних мереж в задачах напівкерованого навчання

У попередніх пунктах було зазначено основні передумови появи напівкерованого навчання, а саме: недоступність великої вибірки позначених даних для навчання моделі а також відсутність логічних зв’язків для розв’язування задачі методами некерованого навчання. Однією з основних стратегій напівкерованого навчання є інтеграція генеративно-змагальної

мережі, що здатна навчитися генерувати позначені дані, а також використовувати їх для навчання мережі класифікації. У цьому пункті буде більш детально розглянуто алгоритм інтеграції генеративно-змагальної мережі у модель напівкерованого навчання.

3.2.1 Основні принципи напівкерованого навчання

Основні принципи напівкерованого навчання, а також інтеграція у нього генеративно-змагальних мереж були детально описані у роботі «*Good semi-supervised learning that requires a bad gan*» 2017 року [15], а також у роботі «*Semi-supervised learning with generative adversarial networks*» 2016 року [16].

Сучасні алгоритми глибокого навчання, як правило, вимагають досить великої кількості позначених даних, оскільки, щоб створити більш узагальнену та широко застосовану модель, необхідно натренувати її на великій кількості різноманітних прикладів, що, в свою чергу, іноді є дещо проблематичним, бо є коштовним, займає багато часу і ресурсів, а в деякий практичних випадках є й зовсім неможливим.

Напівкероване навчання є однією з стратегій зменшення необхідної кількості промаркованих даних. Алгоритм потребує вивчення лише невеликої кількості промаркованих прикладів і великого набору непозначених даних, які зазвичай значно легше отримати.

Суть алгоритму напівкерованого навчання полягає в певній генерації промаркованих даних на основі існуючої вибірки, а потім переходу до традиційного навчання з учителем.

Власне для цієї генерації і застосовують розглянуті вище генеративно-змагальні мережі. Можливість їх інтеграції у алгоритми напівкерованого навчання була згадана ще І. Гудфеллоуом у його оригінальній статті 2014 року [9], але першим практичним застосуванням вважається CatGAN (Categorical GAN) [17]. Як правило, наразі моделі навчаються на наборах даних, що містять 50 000 або більше промаркованих екземплярів, але впровадження GAN може допомогти отримати високу продуктивність з дуже малим обсягом даних: від 20 до 8000.

Основна ідея реалізації напівкерованого навчання з урахуванням алгоритмів GAN — перетворення задачі класифікації з n класів на класифікацію з n + 1 класів, де додатковий клас відповідає підробці даних (зображення, відео, тощо). Усі дійсні класи можна підсумувати, щоб отримати ймовірність реальності даних, що дозволяє використовувати класифікатор як дискримінатор в моделі GAN. Навчити алгоритм відрізнити справжні дані від підроблених можна навіть за допомогою непозначених даних, які, як відомо, є реальними, і із зразками з генератора, які завідомо є підробленими. Цей підхід був одночасно розроблений Салімансом та ін. (2016) [18] та Оденом (2016) [16]. Раніше CatGAN використовував дискримінатор класу n, а не n + 1, тож, можна очікувати, що майбутні вдосконалення GAN принесуть до появи ще кращих результатів застосування напівкерованого навчання.

3.2.2 Порівняльний аналіз GAN і SGAN

У Розділі 2 було детально описано класичну архітектуру генеративно-змагальної мережі, проте для використання згенерованих даних під час навчання моделі класифікації необхідна одна важлива умова: результуючі дані мають бути позначеними. Принципи взаємодії генеративно-змагальних

мереж та напівкерованого навчання детально описала Ольга Петрова у своїй статті «Semi-supervised learning with GANs» на платформі «Medium» [32].

З точки зору міток, дискримінатор у класичних GAN можна розглядати як мережу бінарної класифікації, оскільки він отримує на вхід певний вектор даних, а на вихід повертає оцінку цього вектора – «реальний» або «згенерований». Таким чином, якщо перед розробниками стоїть задача згенерувати та класифікувати дані *K* різних класів, то дискримінатор перетворюється на мережу класифікації *K+1* класів, з них - *K* дійсних класів і 1 клас, що відповідатиме згенерованим даним.

З точки зору навчання даний процес можна поділити на 2 частини: кероване навчання класифікації *K* класів (у навчальній вибірці мають бути приклади екземплярів всіх класів) та некероване навчання класифікації 1 класу – тобто результатів роботи генератора.

Алгоритм навчання генератора та його структура не буде суттєво відрізнятися від стандартної GAN, проте для дискримінатора він буде дещо зміненим. На вхід будуть подаватися дані трьох типів:

- Позначена дійсна вибірка для коректної класифікації;
- Непозначена дійсна вибірка, всі елементи якої належать до *K* класів (*K* - відоме) представлених у першій вибірці;
- Згенерована вибірка – результат роботи генератора. [32]

Також до структури дискримінатора додається функція активації softmax [33]. Ця функція перетворює вектор з *K* дійсних значень на вектор *K* дійсних значень, сума яких дорівнює 1. Вхідні дані можуть бути додатними, від’ємними, нульовими, більшими чи меншими за 1 – функція softmax перетворить їх на значення в інтервалі між 0 і 1, оскільки тільки в такому випадку їх можна буде інтерпретувати як ймовірності [33].

Дану функцію іноді називають softargmax, або багато класовою логістичною регресією. Це пояснюється тим, що функція softmax є певним узагальненням логістичної регресії. Варто зазначити, що на практиці дану функцію можна використовувати в класифікаторі тоді і тільки тоді коли всі класи є взаємовиключними [33].

Ця функція є дуже актуальною для багатьох багатошарових нейронних мереж, оскільки дуже часто передостанній шар виводить оцінки, які досить складно масштабувати, що призводить до появи ряду проблем у майбутньому. В таких ситуація функція softmax зможе перетворити результуючі оцінки на нормалізований розподіл ймовірностей, з яких потім можна скласти статистичний аналіз, представити користувачеві, або застосовувати під час подальшої роботи. Саме тому softmax часто виступає саме останнім шаром нейронної мережі [33].

Формула даної функції виглядає наступним чином:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \qquad (3.1),$$

де $\vec{z}$ – вхідний вектор розмірності K;

z_i – значення елементів вхідного вектора (можуть бути будь-якими дійсними числами);

K – кількість класів.

3.3 Опис програмної реалізації алгоритму напівкерованого навчання з використанням генеративно-змагальних мереж

3.3.1 Постановка задачі

За основу даного прикладу було взято задачу 2 з Розділу 2. Проте, якщо в звичайній генеративно-змагальній мережі основною метою була генерація зображень, то в даному випадку це буде вже класифікація об’єктів, тобто розпізнавання зображень.

Таким чином, задачу можна сформулювати наступним чином: побудувати нейронну мережу, що буде розпізнавати зображення рукописних цифр та буде вимагати якомога меншого набору позначених даних для навчання.

3.3.2 Опис структури розробленої мережі

За основу реалізації даної модифікації алгоритму було взято розв’язок задачі 2 з Розділу 2. Проте, з урахуванням перерахованих вище удосконалень було впроваджено ряд змін у структурі задля підвищення ефективності роботи моделі.

Таким чином, модель матиме наступні складові:

– Мережа генератора складатиметься з 4 шарів: вхідний лінійний шар та 3 приховані згорткові шари. На вхід подається випадковий вектор розмірністю 100, а на вихід отримуємо вектор розмірністю 784 (28^2) – відповідно до розмірності зображень. Вибір такої структури обґрунтований тим, що згорткові мережі набагато краще зарекомендували себе в подібного роду задачах. Два згорткові шари мають функцію активації LeakyReLu, яку частіш за все використовують для навчання генеративно-змагальних мереж, про що

згадував у своїх працях Гудфеллоу [14]. Останній шар має функцію активації тангенс гіперболічний. (Рис. 3.1)

– Мережі дискримінатора складатиметься з 3 згорткових шарів та вихідного лінійного. На вхід подається вектор розмірністю 784, а на вихід отримуємо вектор розмірності 10 відповідно до кількості класів. Оскільки в даному випадку мережа генератора також розв'язує задачу класифікації, то вона буде мати дві функції активації: sigmoid буде використовуватися для некерованої частини, тобто розпізнавання, чи є зображення реальним, а softmax буде застосовуватися для керованої частини, тобто задачі класифікації. (Рис. 3.2). Під час проведення експериментів було виявлено, що в даній задачі для некерованої частіше краще застосовувати власну функцію активації замість sigmoid. Вона матиме наступний вигляд (3.2):

$$- \quad y = \frac{\sum_i e^{z_i}}{\sum_i e^{z_i + 1}}, i = 1,2, \dots k \qquad (3.2)$$

де z – вихідний вектор попереднього шару;

k – кількість класів, в даному випадку 10;

y – ймовірність, буде в діапазоні від 0 до 1.

– Функцій втрат також буде дві: бінарна крос-ентропійна функція буде відповідати некерованій частині навчання, в той час як звичайна крос-ентропійна функцій – керованій.
– В якості алгоритму оптимізації було обрано алгоритм Адам.

Вибір даної структури мережі обирався аналітично, виходячи з постановки завдання. Порівняно з лінійними, згорткові шари набагато краще зарекомендували себе під час роботи з задачами класифікації, так само як і алгоритм оптимізації Адам, про що свідчать роботи І. Гудфелоуа та його колег [14][16]. Розмірності вхідних та вихідних векторів обиралися згідно з

постановкою задачі та розмірністю екземплярів дата сету. Частина гіперпараметрів корегувалися експериментально.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                 [-1, 6272]         633,472
   ConvTranspose2d-2          [-1, 128, 14, 14]         262,144
       BatchNorm2d-3          [-1, 128, 14, 14]             256
   ConvTranspose2d-4          [-1, 128, 28, 28]         262,144
       BatchNorm2d-5          [-1, 128, 28, 28]             256
   ConvTranspose2d-6            [-1, 1, 28, 28]             128
================================================================
Total params: 1,158,400
Trainable params: 1,158,400
Non-trainable params: 0
----------------------------------------------------------------
```

Рисунок 3.1 – Архітектура мережі генаратора

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 128, 14, 14]           1,152
       BatchNorm2d-2          [-1, 128, 14, 14]             256
            Conv2d-3            [-1, 128, 7, 7]         147,456
       BatchNorm2d-4            [-1, 128, 7, 7]             256
            Conv2d-5            [-1, 128, 4, 4]         147,456
       BatchNorm2d-6            [-1, 128, 4, 4]             256
            Linear-7                  [-1, 10]          20,490
================================================================
Total params: 317,322
Trainable params: 317,322
Non-trainable params: 0
```

Рисунок 3.2 – Архітектура мережі дискримінатора

3.3.3 Опис програмної реалізації

За основу реалізації було також взято розв’язок задачі 2, включаючи всі основні інструменти, як-от мова програмування Python, бібліотеки Pytorch, math і matplotlib, набір даних MNIST. Принципи використання даних

інструментів для побудови програмних продуктів детально описав Tryambak Kaushik у своїй статті «Semi-supervised learning with GAN» [34]. Покрокова реалізація буде здійснюватися наступним чином:

Крок 1. Завантаження датасету

Спершу необхідно завантажити вихідний датасет. В даній задачі ми використовуємо відкритий та загальнодоступний набір даних MNIST, що складається за набору зображень рукописних цифр 28 на 28 пікселів у чорно-білій кольоровій гамі та позначень, що відповідають зображеним цифрам. MNIST є досить гнучким, тому за допомогою функцій бібліотеки Pytorch ми можемо нормалізувати набір, змінювати розмір, конвертувати у тензор і так далі.

Для зменшення навантаження та часу навчання було також впроваджено використання групування навчального датасету, тобто створення батчей [34].

Крок 2. Створення моделі генератора

Реалізація класу генератора була зроблена аналогічно до задачі 2, проте задля покращення точності роботи моделі було впроваджено ряд змін. Тепер замість трьох прихованих шарів ReLu будуть згорткові шари, причому перші два включатимуть batch normalization, а третій – ні [34].

Крок 3. Створення моделі дискримінатора

В даній задачі дискримінатор буде виступати й класифікатором також. Тому основною відмінністю від аналогічній реалізації у задачі 2 буде розподіл вхідних даних на 11 класів, а не на 2. Тут також приховані шари було реалізовано як згорткові. Фактично, можна сказати, що дискримінатор буде складатися з двох частин: керованої – класифікація вхідних даних на K класів, екземпляри яких присутні у навчальній вибірці з відповідними мітками, та некерованої – розпізнавання дійсних даних та згенерованих.

Крок 4. Розробка циклу навчання

Процес навчання в даній реалізації буде складатися з 2 вкладених циклів. Внутрішній цикл відповідає за навчання генератора та дискримінатора на усій вибірці даних, яку ми завантажили раніше. Зовнішній цикл повторює навчання відповідно до визначеної кількості ітерацій (епох). Керована та некерована частини навчання реалізовані окремо. Некерована частина схожа на реалізацію навчання у класичній GAN, де до дискримінатора подаються справжні та згенеровані дані. Керована частина використовує лише малу частину навчальної вибірки. Перед початком навчання ваги було задано як випадкові числа з нормальним розподілом з нульовим математичним сподіванням та дисперсією 0.02. Бінарна крос-ентропійна функція втрат використовується щоб обчислити значення втрат генератора та некерованої частини дискримінатора, в той час як звичайно крос-ентропійна функція втрат використовується при роботі лише з керованою частиною дискримінатора. Для обчислення загального значення функції втрат дискримінатора значення його окремих функцій додаються. Алгоритм оптимізації Адам використовується щоб оновити значення вагів генератора та дискримінатора. Тут використовується принцип зворотного поширення для обчислення градієнтів. Щоб уникнути проблеми накопичення градієнтів (gradient accumulation) на кожній ітерації та не перемішувати згруповані дані, на початку кожного циклу дані ініціалізуються заново [34].

Крок 5. Тестування моделі

Задля перевірки правильності роботи моделі та оцінки її характеристик було проведено експеримент на тестовому наборі даних. Результати експерименту, а також обчислені значення втрат та точності було зображено в якості графіків за використанням бібліотеки matplotlib [34].

Програма була розроблена за принципом модулярності, а отже модель може корегуватися за рахунок змін параметрів, що значно спрощує процес дослідження.

3.3.4 Аналіз результатів

Значення заданих гіперпараметрів наведено у Таблиці 3.1:

Змінна	**Опис**	**Значення**
GEN_INPUT	Розмір вхідного вектора генератора	100
EPOCH_NUM	Кількість ітерацій навчання	20
BATCH_SIZE	Розмір батчу датасету	25
IMAGE_SIZE	Розмір зображення (в пікселях)	28
LEARNING_RATE	Рівень навчання	0.002
BETA1	Бета гіперпараметр для алгоритму оптимізації Адам	0.5
NUM_CLASSES	Кількість класів	10

Таблиця 3.1 – вхідні параметри навчання моделі

Всього для навчання моделі знадобилось близько двох годин. Для навчання моделі було застосовано навчальну вибірку з 60000 екземплярів, з яких 6000 екземплярів були з міткою, тобто 10%.

Графік залежності функції втрат від кількості ітерацій зображено на Рис. 3.2.

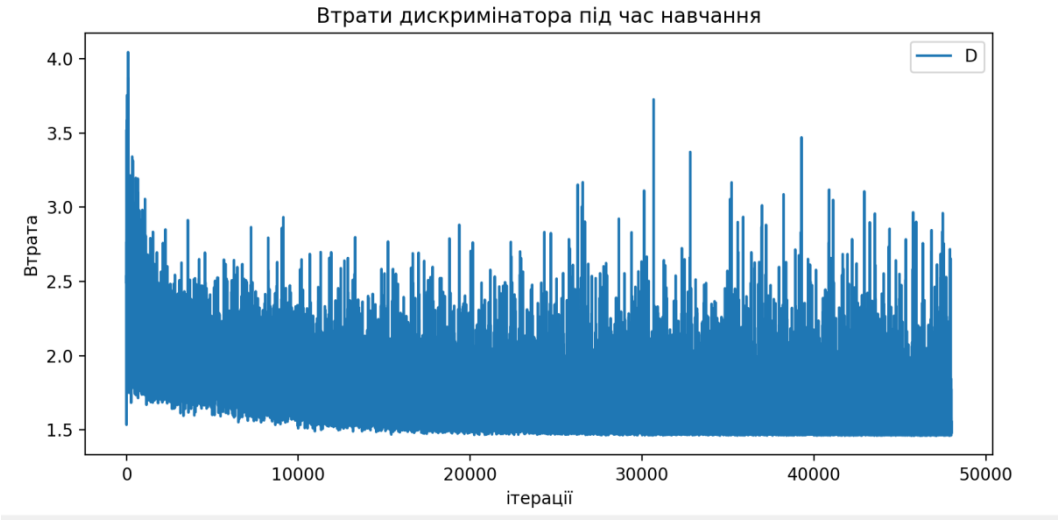


Рисунок 3.2 – Графік функцій втрат дискримінатора

Як ми можемо побачити, для дискримінатора є невеликий стрибок на початку навчання, проте далі показники функції втрат залишаються приблизно в діапазоні від 0 до 2, причому нижня межа є більш стабільною і прямує до нуля. Проте, ми все одно бачимо досить суттєві стрибки. Це можна обґрунтувати тим, що дана мережа є змагальною, отже з покращенням роботи генератора дискримінатор буде отримувати більше втрат.

Приклад зображень з навчальної вибірки наведено на Рис. 3.3.

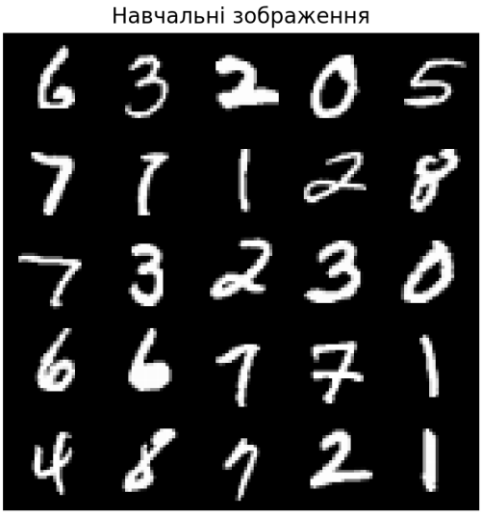


Рисунок 3.3 – Приклад елементів навчальної вибірки

Результати роботи генератора зберігаються в окрему теку. Приклади згенерованих зображень зображено на Рис. 3.4.

Рисунок 3.4 Приклади результатів роботи генератора зі збільшенням кількості ітерацій

Після завершення навчання було проведено тестування на вибірці з 400 тестових наборів. Приклад з тестової вибірки зображено на Рис. 3.5.

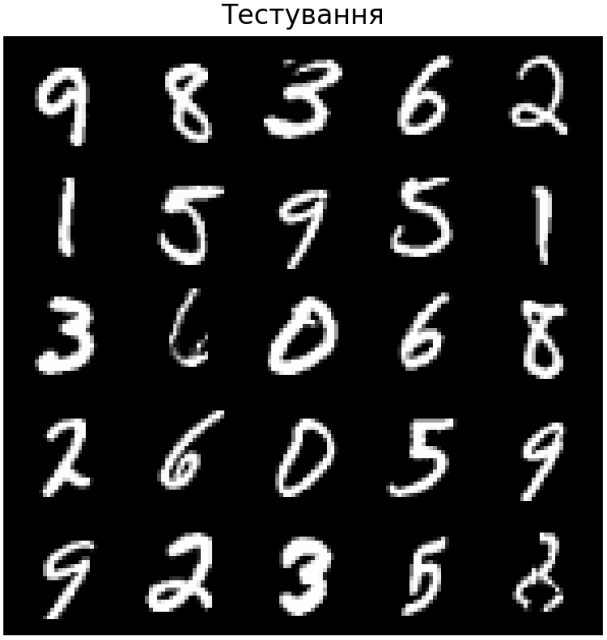


Рисунок 3.5 – Приклад тестової вибірки

Тестування дало наступні результати (Рис. 3.6):

Середнє значення функції втрат: 0.9835.

Точність: 0.9862 (99%).

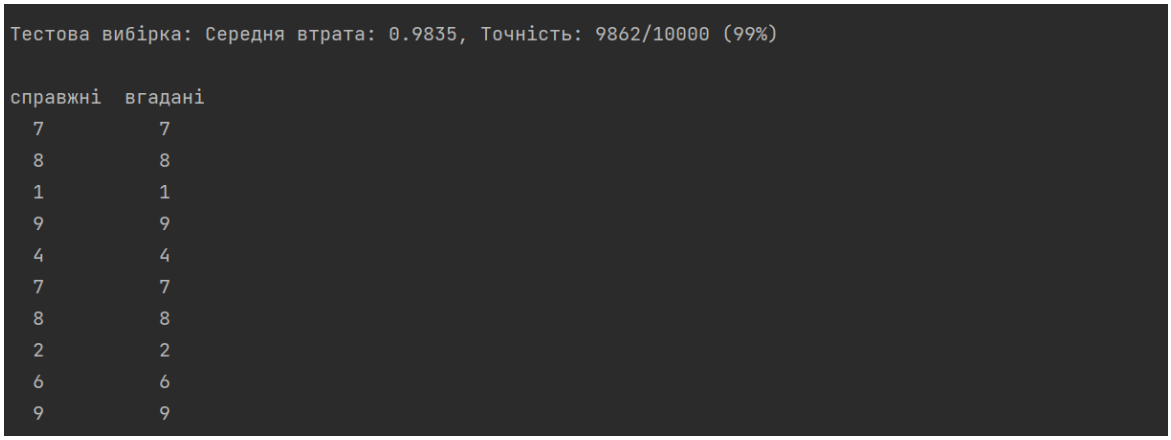


Рисунок 3.6 – Результати тестування

Таким чином, можна стверджувати, що робота моделі дала високі результати і вона є досить ефективною.

3.3.5 Порівняльний аналіз роботи розробленої моделі та звичайної багатошарової нейронної мережі

Для проведення порівняльного аналізу було розроблено звичайну багатошарову нейронну модель для розв’язку тієї ж самої задачі. Ця модель складається з чотирьох шарів: вхідний, два прихованих з функціями активації ReLu та вихідний з функцією активації softmax [33]. Оптимізація здійснюється за допомогою алгоритму Адам, в якості функції втрат було обрано крос-ентропію. Результати роботи даної мережі зображено на Рис. 3.7.

```
Середнє значення функції втрат:   0.14063581824302673
Точність:   0.9750999808311462
```

Рисунок 3.7 – Результати роботи багатошарової нейронної мережі

Порівняльний аналіз основних характеристик роботи двох алгоритмів представлено у Таблиці 3.2.

Характеристика	**Багатошарова нейронна мережа**	**SGAN**
Кількість епох навчання	20	20
Розмір розміченої вибірки	60000	6000
Розмір нерозміченої вибірки	-	60000
Точність	97%	99%
Втрата	0.14	0.9835
Час навчання	~3 хвилини	~2 години

Таблиця 3.2 – Порівняння характеристик багатошарової нейронної мережі та SGAN

З огляду на це можна зробити наступні висновки:

- SGAN безперечно переважає багатошарову нейронну мережу з точки зору необхідної навчальної вибірки, оскільки потребує лише 10% того датасету, що необхідний для навчання звичайної нейронної мережі.
- З іншого боку, навчання SGAN потребує значно більше часу, оскільки відбувається навчання двох його складових: генератора і дискримінатора, що потребує додаткової складності та часу.
- Точність SGAN вище за точність звичайної нейронної мережі.

– Значення функції втрат у звичайної нейронної мережі буде меншим, оскільки SGAN є мережею змагальною, тому навіть при покращенні мережі під час навчання значення функції втрат може залишатися досить високим для генератора і дискримінатора відповідно (в даній таблиці бралося значення саме дискримінатора, оскільки він є основною мережею при даній постановці задачі).

Отже, розроблена модель напівкерованого навчання задовольняє поставленим цілям, а саме: досягти якомога більшої точності при використанні якомога меншої кількості розмічених даних і показала високі результати порівняно зі звичайною багатошаровою нейронною мережею.

3.4 Огляд ідей практичного застосування генеративно-змагальних мереж

Генеративно-змагальні мережі є досить новою технологією, яка зазнала суттєвих вдосконалень протягом останніх 3-5 років. Якщо у 2016 році GAN використовувалися переважно для дослідницьких проектів, то сьогодні вони широко застосовуються в багатьох сферах людської діяльності і показують вражаючі результати. Надалі у цьому пункті буде розглянуто декілька прикладів ідеї застосування генеративно-змагальних мереж та їх практичної реалізації, огляд яких було наведено у статті Олівера Ху [36].

3.4.1 Циклічна генеративно-змагальна мережа (cycle GAN)

Міждоменне перенесення було одним з перших напрямків практичного застосування генеративно-змагальних мереж. Такий тип мереж здатен конвертувати зображення одного типу в інший тип. Наприклад, це може бути представлення фотографії як картини в стилі того чи іншого художника, заміна певних об’єктів зображення на інші, як-от трансформація коні-зебри на Рис. 3.8.

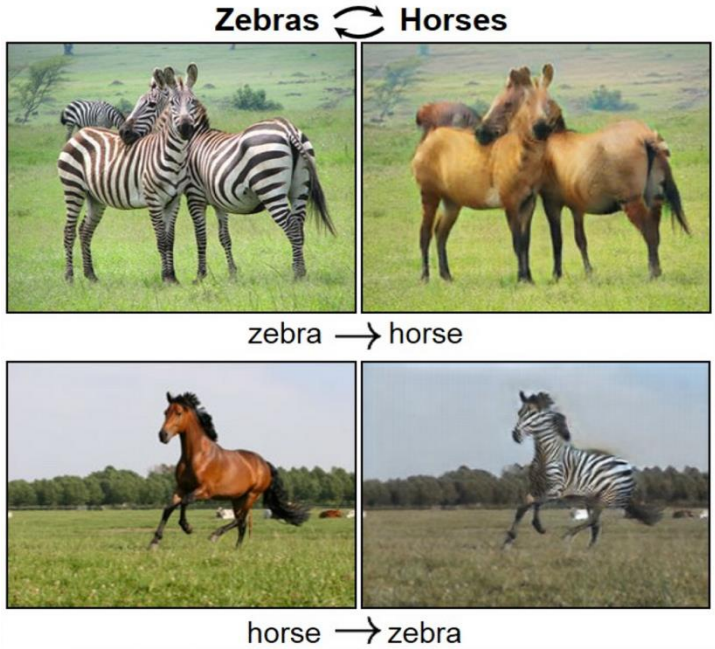


Рисунок 3.8 – Трансформація зображення коні – зебри [36]

Cycle GAN складається також з генератора та дискримінатора. Проте, на вхід до генератора подається не випадковий вектор, а визначені вхідні дані, наприклад, зимова фотографія. До дискримінатора на вхід подаються очікувані змінені фотографії, наприклад літня фотографія, і таким чином генератор навчається перетворювати зображення зими на літо. [36]

3.4.2 Покращення якості зображень

Ще одна область обробки зображень, в якій генеративно-змагальні мережі вже досить добре себе зарекомендували. Покращення якості зображень, підвищення резолюції є важливою задачею сучасних дослідників, оскільки дозволятиме працювати зі старими файлами, документами та фотографіями засобами сучасних платформ та інструментів [36]. Приклад роботи даної мережі зображено на Рис. 3.9.

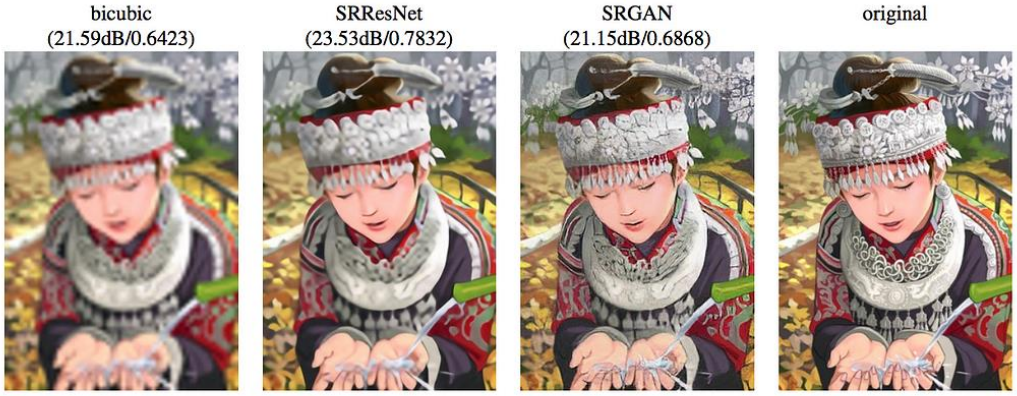


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

Рисунок 3.9 – Приклад роботи мережі з покращення якості зображення [36]

Структура генеративно-змагальної мережі в даному випадку буде досить стандартною: вона складатиметься з багатьох згорткових шарів, batch normalization та функції активації ReLu, або ускладнена ReLu.

3.4.3 Синтез зображень високої якості

Дана задача є оберненою до стандартної задачі сегментації. Тут на вхід мережі подаються семантичні карти, а вона має згенерувати реалістичне

зображення, спираючись на них [36]. Через складність збору навчальної вибірки та її коштовність в даній задачі було застосовано напівкероване навчання. Приклади результатів роботи такої мережі зображено на Рис. 3.10.

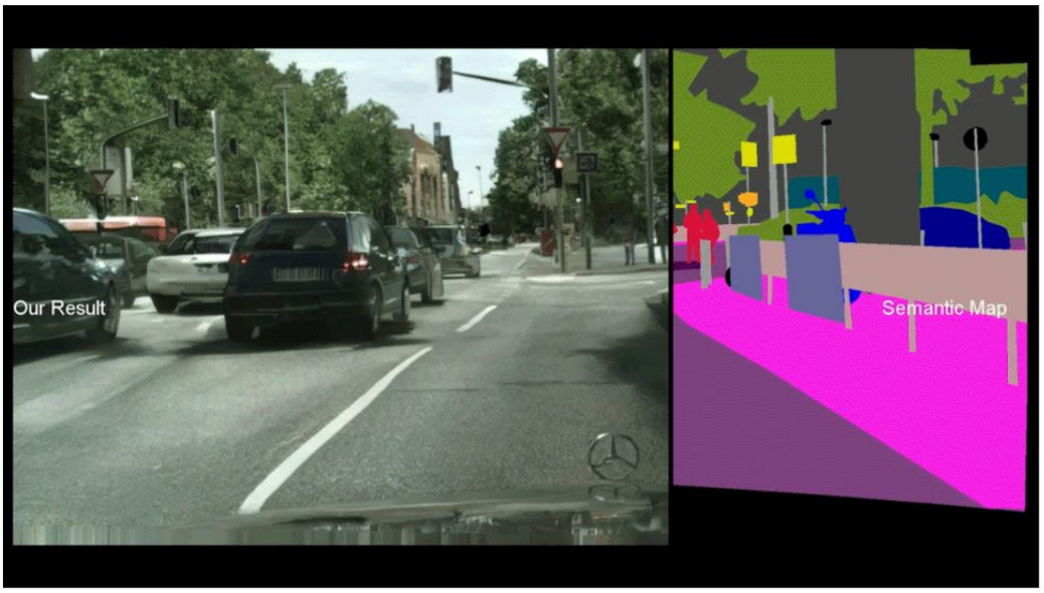


Рисунок 3.10 – Приклад створення зображень з семантичних карт [36]

3.4.4 Відновлення пошкоджених зображень

Основною метою даної мережі є відновлення пошкоджених зображень, або їх фрагментів. На практиці існує багато ситуацій, коли фотографії пошкоджуються фізично, відбувається втрата даних під час сканування обробки, або помилка трапляється вже на електронному носії. В будь-якому випадку, наявність потужного інструменту відновлення даних змогло б суттєво покращити ситуацію та полегшити роботу спеціалістів [36].

Такі GAN навчаються генерувати втрачені фрагменти зображень реалістично і непомітно шляхом обробки цілих зображень в пошкоджених. Приклад роботи такої мережі зображено на Рис. 3.11.

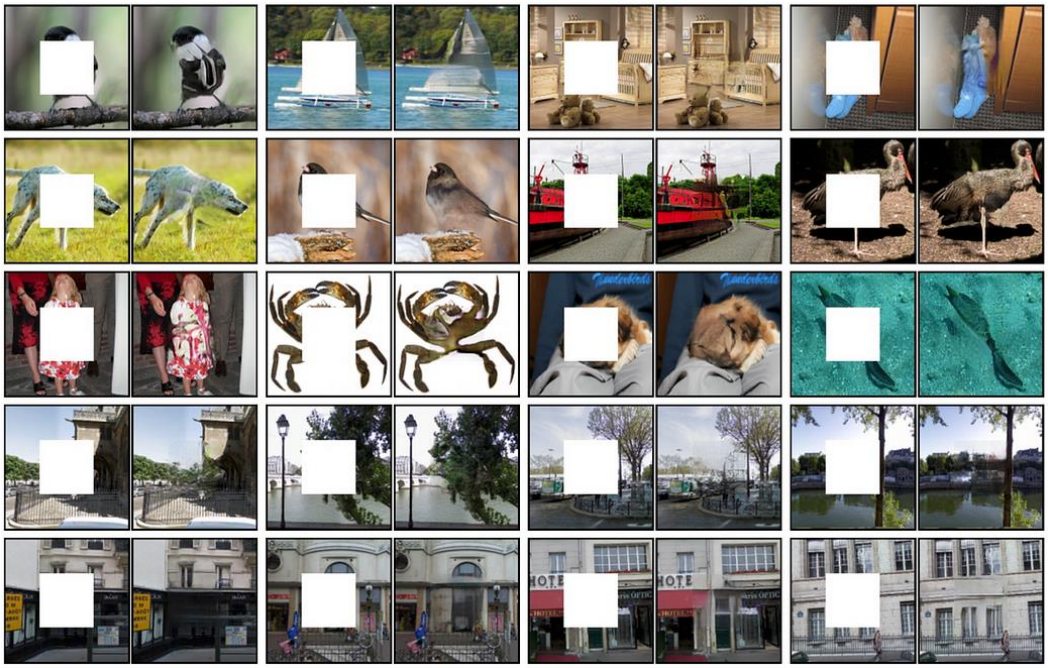


Рисунок 3.11 – Приклад відновлення зображень [36]

3.4.5 Розпізнавання медичних аномалій

Медицина є однією з найбільш перспективних галузей для впроваджень технологій штучного інтелекту. З точки зору генеративно-змагальних мереж, в медицині існує багато задач аналізу зображень, розпізнавання аномалій, пошук захворювань, тощо. Проте, більшість медичних даних є конфіденціальними, отже пошук необхідної навчальної вибірки може видатися надзвичайно складним і коштовним. Тому для таких задач доречно використовувати алгоритм напівкерованого навчання на базі генеративно-змагальних мереж, який був описаний у попередніх. Приклад схеми роботи такої мережі зображено на Рис. 3.12.

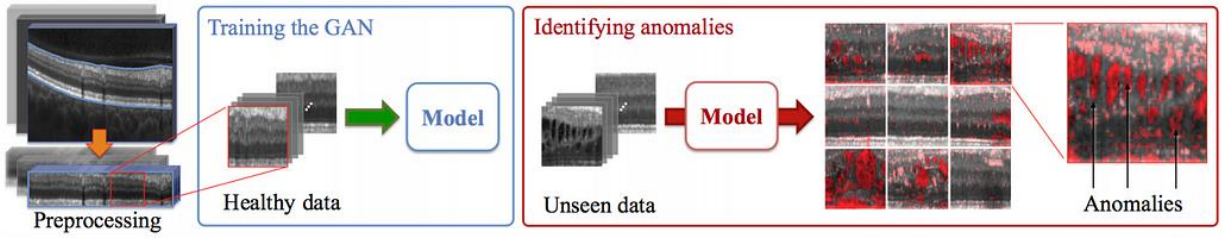


Рисунок 3.12 – Приклад роботи мережі з аналізу медичних знімків [36]

3.5 Висновки до третього розділу

У розділі 3 було розглянуто основні принципи роботи напівкерованого навчання, було обґрунтовано його переваги та було досліджено передумови його появи. Також було проаналізовано джерела та розроблено узагальнення про алгоритм інтеграції генеративно-змагальних мереж у напівкероване навчання. Було проведено порівняльний аналіз GAN і SGAN, на основі якого в було створено програмну реалізацію даного алгоритму на тестовому прикладі.

Також було проведено аналіз характеристик розробленої моделі, проведено порівняльний аналіз SGAN та звичайної багатошарової нейронної мережі, зроблено висновки. Було проведено огляд існуючих реалізацій, ідей або проектів застосування генеративно-змагальних мереж та напівкерованого навчання та проаналізовано перспективи на напрями подальшого розвитку.

РОЗДІЛ 4. ФУКНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка задачі

Метод функціонально-вартісного аналізу, що застосовується в даній дипломній роботі, дозволяє знайти, дослідити та оптимізувати собівартість продукту та його цінність з точки зору потенційних споживачів. Він дозволятиме зменшити час на розробку та оптимізувати основні витрати.

Головною метою функціонально-вартісного аналізу є визначення вартості програмного застосунку або розробленого алгоритмічного метода, його актуальність для ринку та потенційну прибутковість з точки зору монетизації. Даний метод фінансового аналізу дозволятиме оптимізувати такі характеристики як час розробки і тестування, обрані інструменти та методики, тим самим зробивши процес створення та впровадження програмного продукту більш економічно вигідним та оптимальним для розробників.

Функціонально-вартісний аналіз буде застосовано до самого алгоритму класифікації, розробленого на основі методів напівкерованого навчання із застосуванням генеративно-змагальних мереж та його програмної реалізації, побудованої для розв’язку задачі розпізнавання рукописних цифр. Особливість даної реалізації полягає в тому, що вона є досить універсальною і її можна адаптувати під конкретні вимоги користувача.

Процес функціонально-вартісного аналізу буде складатися з:

- Визначення плану розробки програмного продукту;
- Визначення оптимальних інструментів розробки та їх вартості;
- Обчислення кінцевої вартості продукту та потенціал монетизації.

Для визначення компонентів плану розробки та їх подальшого аналізу необхідно спершу визначити основні вимоги до програмного продукту. Вони будуть полягати в наступному:

- Програмний продукт повинен бути мультиплатформленим і працювати на базі основних операційних систем. Також він повинен працювати на звичайних комп’ютерах зі стандартним офісним пакетом конфігурації і не вимагати великого обсягу пам’яті та потужності.
- Код програмного продукту повинен бути читабельним та зрозумілим для інших розробників. Задля цього необхідна наявність документації, коментарів в самому коді та README.md файлу.
- Після завершення навчання модель має швидко обробляти вхідні дані.
- Вартість процесу розробки, часові витрати, вартість інструментів, засобів і т.д. має бути якомога меншою.

4.1.1 Обґрунтування функцій програмного продукту

Для проведення аналізу необхідно визначити задані функції:

F_0 – розробка програмного продукту,

F_1 – мова програмування,

F_2 – метод проектування,

F_3 – середовище розробки,

Тепер для кожної з цих функцій задамо декілька потенційних варіантів значень:

Для вибору мови програмування:

1. C++,
2. Python.

Для методу проектування:

1. Написання всіх функцій вручну,
2. Використання бібліотек.

Для середовища розробки:

1. Python IDLE,
2. Inteliji IDE.

4.1.2 Варіанти реалізації основних функцій

Для визначення потенційних комбінацій необхідно розробити морфологічну карту залежності значень вище перерахованих функцій.

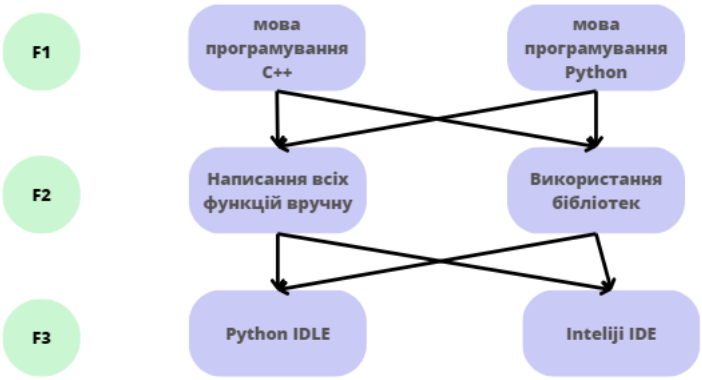


Рисунок - 4.1 Морфологічна карта

На морфологічній карті ми можемо побачити всі можливі комбінації значень функцій F_1, F_2, F_3, що утворює повну множину варіантів реалізації програмного продукту.

Побудуємо та проаналізуємо позитивно-негативну матрицю.

Основні функції	Варіанти	Переваги	Недоліки
F_1	А	Статична типізація, висока швидкість	Синтаксична складність,

		виконання коду.	написання коду займає багато часу
	Б	Доступність бібліотек і фреймворків, універсальність, кросплатформеність, висока швидкість написання коду	Динамічна типізація
F_2	А	Реалізація підлаштовується виключно під завдання, гнучкість.	Розробка і тестування займає багато часу, є висока ймовірність наявності помилок та неоптимального рішення.
	Б	Розробка займає значно менше часу та ресурсів, висока якість коду.	Програмний за стосунок буде більш універсальним, розробка не дуже гнучка.
F_3	А	Легкість завантаження, займає мало пам’яті.	Не зручний інтерфейс, що сповільнює процес

			написання коду.
	Б	Великий спектр інструментів, під світка, пошук багів, git, тощо.	Займає багато пам’яті.

Таблиця 4.1 – Позитивно-негативна матриця

З аналізу позитивно-негативної матриці можна зробити висновок, що деякі варіанти реалізації функцій можна відкинути, оскільки вони не відповідають поставленим вимогам.

Функція F_1:

Швидкість написання коду та гнучкість є важливими параметрами, оскільки розробка та тестування вимагають впровадження великої кількості змін, швидкої переробки, модифікації та адаптації. Саме тому перевага надається варіанту Б.

Функція F_2:

Розроблені бібліотеки мови програмування Python є якісними, надійними, оптимальними і доступними. Їх використання здатне значно підвищити швидкість написання коду та його якість, а отже варіант А написання всіх функцій вруну відкидаємо і надаємо переваги варіанту Б.

Функція F_3:

Середа розробки не є критичним критерієм під час розробки, тому будемо розглядати обидва варіанти А і Б.

Таким чином, маємо наступні варіанти реалізації програмного продукту:

1) $F_{1Б} - F_{2Б} - F_{3А}$

2) $F_{1Б} - F_{2Б} - F_{3Б}$

Для оцінки якості розглянутих функцій обрана окрема система параметрів.

4.2 Обґрунтування системи параметрів програмного продукту

4.2.1 Опис параметрів

Параметри вибору обираються згідно з встановленими вимогами від програмної реалізації, які визначалися раніше в цьому розділі. Надалі ці параметри будуть використовуватися задля вибору коефіцієнта технічного рівня.

Таким чином, було обрано наступні параметри:

X1 – приблизний обсяг програмного коду;

X2 – швидкість роботи мови програмування;

X3– обсяг пам’яті, який необхідний для зберігання даних та проведення обчислень.

Визначимо мінімальні, середні та максимально допустимі значення (Табл. 4.2).

Назва параметру	Умовні позначення	Одиниці виміру	Значення		
			гірші	середні	кращі
Обсяг програмного коду	X1	Кількість рядків	1500	600	400
Швидкість роботи мови програмування	X2	оп/мс	40	70	120
Обсяг пам’яті	X3	Мб	64	32	16

Таблиця 4.2 – Система параметрів

Побудуємо графічні характеристики параметрів.

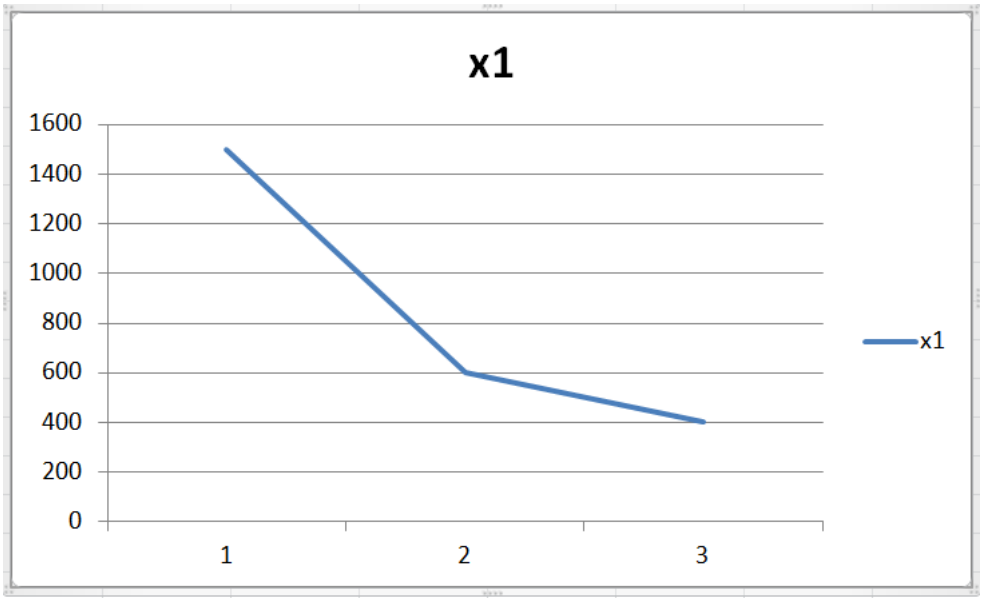


Рисунок 4.2 – X1, обсяг програмного коду

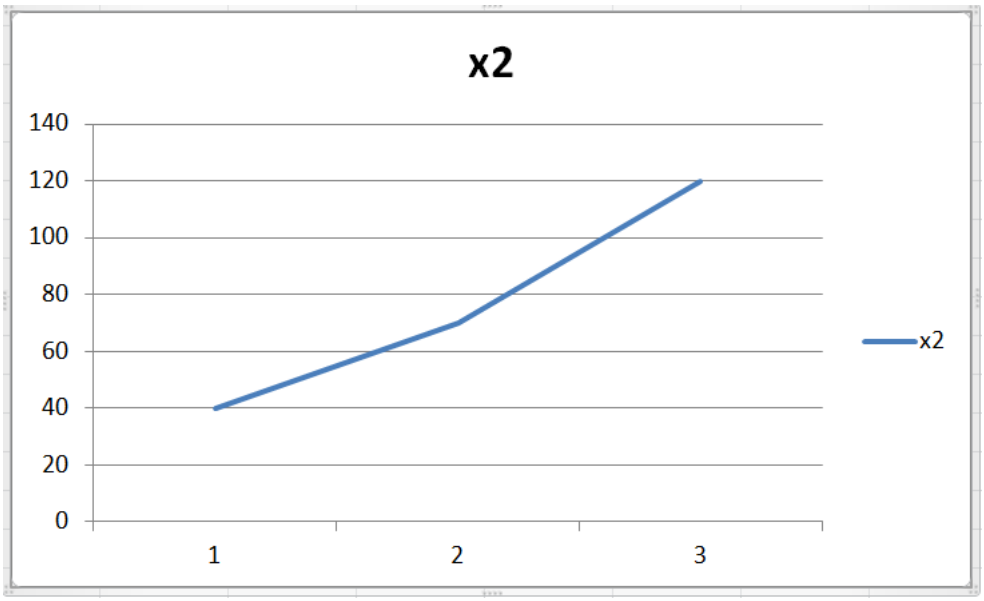


Рисунок 4.3 – X2, швидкість роботи мови програмування

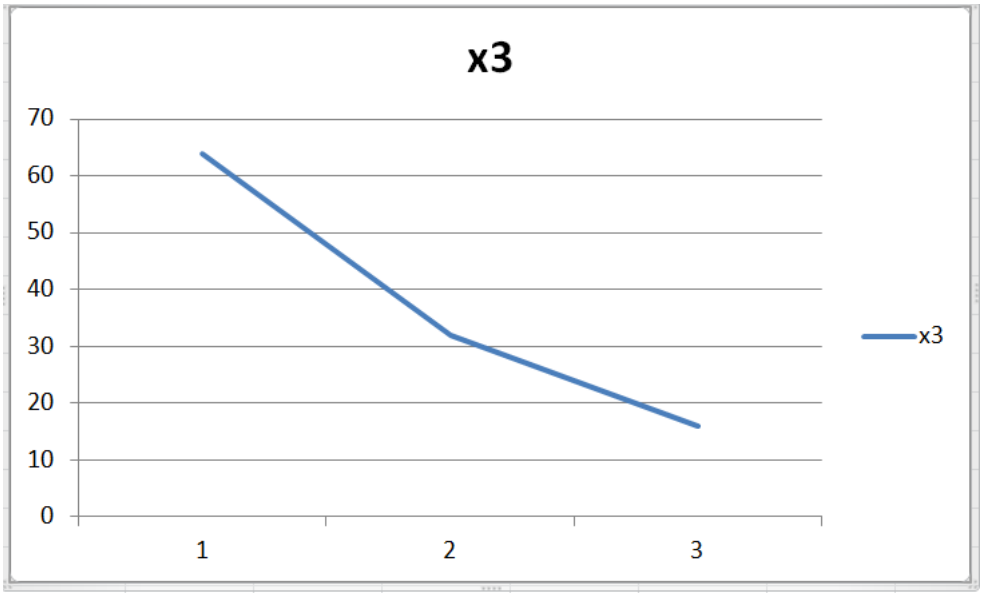


Рисунок 4.4 – X3, обсяг пам’яті

4.2.2 Аналіз експертного оцінювання параметрів

Після проведення аналізу експерти оцінюють наскільки важливим є той чи інший параметр для поставленої задачі, а саме: розробки програмного

застосунку, що буде класифікувати вхідні зображення з максимальної точністю.

За допомогою методу попарного порівняння визначимо ступінь значущості кожного з параметрів.

Визначення коефіцієнтів значимості включає в себе наступні етапи:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Параметр	Ранг параметру по оцінці експерта							Сума рангів, R_i	Відхилення Δ_i	Квадрат відхилення $(\Delta_i)^2$
	1	2	3	4	5	6	7			
X1	3	3	3	2	3	3	3	20	6	36
X2	2	2	2	3	1	2	2	14	0	0
X3	1	1	1	1	2	1	1	8	-6	36
Разом	6	6	6	6	6	6	6	42	0	72

Таблиця 4.3 - Результати ранжування параметрів

Обчислимо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 72}{7^2(3^3 - 3)} = 0{,}734 > W_k = 0{,}67. \qquad (4.1)$$

Знайдений коефіцієнт узгодженості переважає нормативний, а отже його можна вважати достовірним. Тепер проведемо попарне порівняння параметрів.

Параметр	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 та X2	<	<	>	<	<	<	>	<	0.5
X1 та X3	>	>	>	<	<	<	>	>	1.5
X2 та X3	>	>	>	>	<	>	>	>	1.5

Таблиця 4.4 - Попарне порівняння параметрів.

Наведемо розрахунок вагомості параметрів.

Параметри	Параметри X_i			Перший крок		Другий крок		Третій крок		Четвертий крок	
	X1	X2	X3	b_i	K_{bi}	b_i	K_{bi}	b_i	K_{bi}	b_i	K_{bi}
X1	1	0.5	1.5	3	0.333	8	0.333	20.5	0.315	57	0.301
X2	1.5	1	1.5	4	0.444	11.5	0.479	30.25	0.465	92.375	0.488
X3	0.5	0.5	1	2	0.222	4.5	0.188	14.25	0.219	39.625	0.210
Разом				9	1	24	1	65	1	189	1

Таблиця 4.5. Розрахунок вагомості параметрів

Як ми можемо бачити у Таблиці 4.5, різниця значень коефіцієнтів вагомості не більша 2%, а отже більше ітерацій не потрібно.

Визначимо рівень якості кожного з обраних варіантів виконання основних функцій. Абсолютні значення параметрів X1 (Обсяг коду), X3 (об’єм пам’яті) відповідають технічним вимогам умов функціонування даного ПП.

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	Б	X1	600	25	0,301	7,525
F2	Б	X2	60	29	0,488	14,152
F3	А	X3	32	19	0,095	1,805
	Б	X3	20	23	0,115	2,645

Таблиця 4.6. Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 7{,}525 + 14{,}152 + 1{,}805 = 23{,}482 ;$$

$$K_{K2} = 7{,}525 + 14{,}152 + 2{,}645 = 24{,}322.$$

Як видно з розрахунків, кращим є варіант Б, оскільки для нього коефіцієнт технічного рівня має найбільше значення.

4.3 Економічний аналіз варіантів розробки ПП

Для того, щоб визначити загальну вартість програмного продукту необхідно спершу обчислити його трудомісткість.

Всі варіанти включають в себе наступні завдання:

1. Проектування програмного продукту;

2. Розробка програмного забезпечення.

Проте, варто зазначити, що обидва варіанти матимуть різні додаткові завдання:

1. Реалізація власних методів;
2. Розробка за допомогою інтерфейсу бібліотек.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \qquad (4.2)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання бібліотек;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання трудомісткість дорівнює: T_P = 40 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.6. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта

$К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 40 \cdot 1.6 \cdot 0.8 = 51,2 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =26 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$Т_2 = 26 \cdot 0.9 \cdot 0.8 = 18,72 \text{ людино-днів.}$$

Для завдання три (А) використовується алгоритм третьої групи складності, тобто $Т_Р$ =10 людино-днів, $К_{СК}$ = 1, $К_{СТ}$ =0.6:

$$Т_{3а} = 10 \cdot 0.6 = 6 \text{ людино-днів.}$$

Для завдання три (Б) використовується алгоритм другої групи складності, тобто $Т_Р$ =14 людино-днів, $К_{СК}$ = 1, $К_{СТ}$ =0.6:

$$Т_{3б} = 14 \cdot 0.6 = 8,4 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (51,2 + 18,72 + 6) \cdot 8 = 607,36 \text{ людино-годин.}$$

$$Т_{II} = (51,2 + 18,72 + 8,4) \cdot 8 = 626,56 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 16000 грн., один аналітик в області даних з окладом 18000. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} грн., \qquad (4.3)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{16000 + 16000 + 18000}{3 \cdot 21 \cdot 8} = 99,2 \ \ грн. \qquad (4.4)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \qquad (4.5)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 99,2 \cdot 607,36 \cdot 1.2 = 72300,13$ грн.

II. $С_{ЗП} = 99,2 \cdot 626,56 \cdot 1.2 = 74585,70$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 72300,13 \cdot 0.22 = 15906,03$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 74585,70 \cdot 0.22 = 16408,85$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 16000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 16000 \cdot 0,2 = 38400 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 38400 \cdot (1 + 0.2) = 46080 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 46080 \cdot 0,22 = 10137,6 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 15000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 15000 = 5250 \text{ грн.}$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 15000 \cdot 0.08 = 1680 \text{ грн.,}$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.35 =$$

$$= 627,2 \text{ години,}$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 627,2 \cdot 0,2 \cdot 0,3 \cdot 4,87 = 183,27 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 15000 \cdot 0,67 = 10050 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.6)$$

$$С_{ЕКС} = 46080 + 10137,6 + 5250 + 1680 + 183,27 + 10050 = 73380,87 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 73380,87 / 627,2 = 117 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т, \qquad (4.7)$$

I. $С_М = 117 \cdot 607,36 = 71061,12$ грн.

II. $С_М = 117 \cdot 626,56 = 73307,52$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \qquad (4.8)$$

I. $С_Н = 72300,13 \cdot 0,67 = 48441,08$ грн.

II. $С_Н = 74585,7 \cdot 0,67 = 49972,42$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \qquad (4.9)$$

I. $С_{ПП} = 72300,13 + 15906,03 + 71061,12 + 48441,08 = 207708,36$ грн.

II. $С_{ПП} = 74585,7 + 16408,85 + 73307,52 + 49972,42 = 214274,49$ грн.

4.4 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = {К_{Кj}}/{С_{Фj}}, \qquad (4.10)$$

$К_{ТЕР1} = 23,482 / 207708,36 = 11,3 \cdot 10^{-5}$,

$К_{ТЕР2} = 24,322 / 214274,49 = 11,4 \cdot 10^{-5}$.

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2} = 11,4 \cdot 10^{-5}$.

Таким чином можна зробити висновок, що найкращим в даному випадку буде другий випадок. У нього виявився найкращий показник

техніко-економічного рівня якості $К_{ТЕР} = 11,4 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Вибір мови програмування - Python;
- Реалізація алгоритмічної частини відбувається за допомогою бібліотек;
- Використання середовища розробки – Inteliji IDE.

Такий варіант запровадить зручний та ефективний робочий інструмент для розробників у якості середовища та бібліотек і надасть можливість для швидкої та оптимальної реалізації.

4.4 Висновки до четвертого розділу

Отже, в четвертому розділі було здійснено повний функціонально-вартісний аналіз даного програмного продукту а також оцінку його основних функцій та характеристик.

В результаті виконання перерахованих вище дій було оцінено основні функції програмного продукту, знайдено параметри характеризації та основні компоненти.

ВИСНОВКИ

В ході виконання даної дипломної роботи було проведено дослідження в області нейронних мереж та машинного навчання в цілому, а також конкретно в сфері генеративно-змагальних мереж.

У Розділі 1 було проведено аналіз стану розвитку сфери штучного інтелекту в Україні за допомогою роботи з нормативно-правовими документами, а також було представлено теоретичні відомості з теми нейронних мереж, машинного навчання, їх властивостей та класифікації. Було надано основні поняття про генеративно-змагальні мережі.

У Розділі 2 було проведено структурно-параметричний синтез генеративно-змагальних мереж, було досліджено їх побудову, топологію, основні засоби та інструменти будови. Було проаналізовано існуючі мови програмування та бібліотеки, доступні для створення моделей машинного навчання та було обрано бібліотеку Pytorch мови програмування Python як основний інструмент подальших досліджень. За його допомогою було побудовано практичний розв'язок двох експериментальних задач: задачі генерації пар типу (x, y) , де $x \in [0; 2\pi]$ та $y = \sin(x)$, а також задачі генерування рукописних цифр. Обидві моделі було вдало представлені і давали задовільні результати.

У Розділі 3 було проведено огляд основних принципів напівкерованого навчання, було проведено детальний аналіз архітектури алгоритму напівкерованого навчання з використанням генеративно-змагальних мереж. Також було побудовано власну реалізацію розв'язку задачі класифікації, проведено порівняльний аналіз створеної моделі напівкерованого навчання та класичної моделі навчання з учителем, зроблено висновки.

У Розділі 4 було проведено функціонально-вартісний аналіз розробленого програмного продукту, обчислено його вартість та здійснено пошук оптимальних параметрів. Зроблено висновки.

Отже, дана робота є дуже актуальною, оскільки генеративно-змагальні мережі є досить молодою, але надзвичайно перспективною галуззю машинного навчання. Вони вже добре зарекомендували себе для розв’язку задач генерації нових зображень, наборів даних, відео, аудіо, тощо. Вони можуть застосовуватися як самостійно – для створення нових логотипів, дизайнів, покращення якості фотографій, їх модифікації, тощо, так і в співпраці з класичними нейронними мережами, тим самим забезпечуючи їм необхідну навчальну вибірку, як-от в задачах напівкерованого навчання.

Напівкероване навчання дозволяє розробникам уникати проблеми пошуку великої навчальної вибірки, що є надзвичайно актуальним для сучасних проектів. Як показала експериментальна модель, даний метод дозволяє отримати велику точність, при цьому використовуючи близько 10% позначеного набору даних.

Розроблена модель показала високі результати порівняно з класичною моделлю навчання з учителем для розв’язку тієї ж самої задачі: вона дала вищу точність та потребувала набагато меншого набору позначених даних. Саме тому можна з впевненістю сказати, що ця сфера буде ще активно розвиватися і досліджуватися у наступні роки.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Штучний інтелект [Електронний ресурс] // Termin.in.ua – URL: https://termin.in.ua/shtuchnyy-intelekt/.(дата звернення – 8.04.2023)
2. Розпорядження Кабінету Міністрів України від 02.12.2020 № 1556-р «Про схвалення Концепції розвитку штучного інтелекту в Україні» [Електронний ресурс] URL: https://zakon.rada.gov.ua/laws/show/1556-2020-%D1%80#Text (дата звернення – 8.04.2023)
3. Кравченко В. Що таке нейронні мережі та як вони працюють? Класифікація штучних нейромереж [Електронний ресурс] / Віталій Кравченко // LivingFo. – 2022. – URL: https://livingfo.com/shcho-take-nejronni-merezhi-ta-iak-vony-pratsiuiut/ . (дата звернення – 15.04.2023)
4. Штучна нейронна мережа [Електронний ресурс] // Wikipedia – URL: https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B0_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0 . (дата звернення – 16.04.2023)
5. Згорткова нейронна мережа [Електронний ресурс] // Wikipedia – URL: https://uk.wikipedia.org/wiki/%D0%97%D0%B3%D0%BE%D1%80%D1%82%D0%BA%D0%BE%D0%B2%D0%B0_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0 . (дата звернення – 16.04.2023)
6. Класичне машинне навчання: завдання класифікації, узагальнення, кластеризації даних [Електронний ресурс] // Evegreen. – 2019. – URL: https://evergreens.com.ua/ua/articles/classical-machine-learning.html . (дата звернення - 20.04.2023)

7. Що таке машинне навчання [Електронний ресурс] // Evegreen – URL: https://evergreens.com.ua/ua/articles/machine-learning-overview.html . (дата звернення - 20.04.2023)
8. Kholod S. Коротко про глибинне (навчання) [Електронний ресурс] / Sophie Kholod. – 2018. – URL: https://medium.com/@sophiekholod/%D0%BA%D0%BE%D1%80%D0%BE%D1%82%D0%BA%D0%BE-%D0%BF%D1%80%D0%BE-%D0%B3%D0%BB%D0%B8%D0%B1%D0%B8%D0%BD%D0%BD%D0%B5-%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F-4c441d556f7c . (дата звернення – 22.04.2023)
9. Goodfellow I.J., et al. Generative adversarial networks. Universite de Montreal. 2014. 9 p.
10. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y.(2014a). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672–2680).
11. Нейромережі GAN в створенні нових моделей / Т.А. Самолюк // Комп'ютерні засоби, мережі та системи. — 2019. — № 18. — С. 86-90. — Бібліогр.: 7 назв. — укр.
12. Goodfellow, I. (2017). NIPS 2016 tutorial: Generative adversarial networks. arXiv:1701.00160.
13. Дівергенція Кульбака-Лейблера [Електронний ресурс] // Medium. – 2019. – URL: https://congyuzhou.medium.com/%D0%B4%D0%B8%D0%B2%D0%B5%D1%80%D0%B3%D0%B5%D0%BD%D1%86%D0%B8%D1%8F-%D0%BA%D1%83%D0%BB%D1%8C%D0%B1%D0%B0%D0%BA%D0%B0-%D0%BB%D0%B5%D0%B9%D0%B1%D0%BB%D0%B5%D1%80%D0%B0-431fa749fddb . (дата звернення – 25.04.2023)

14. Рівновага Неша [Електронний ресурс] // Wikipedia – URL: https://uk.wikipedia.org/wiki/%D0%A0%D1%96%D0%B2%D0%BD%D0%BE%D0%B2%D0%B0%D0%B3%D0%B0_%D0%9D%D0%B5%D1%88%D0%B0 . (дата звернення – 25.04.2023)
15. Dai, Z., Yang, Z., Yang, F., Cohen, W. W., & Salakhutdinov, R.R (2017). Good semi-supervised learning that requires a bad gan. In Advances in neural information processing systems (pp. 6510–6520)
16. Odena, A. (2016). Semi-supervised learning with generative adversarial networks. arXiv:1606.01583.
17. Springenberg J. T (2015). Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks. arXiv: 1511.06390.
18. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training gans. In Advances in neural information processing systems (pp. 2234–2242).
19. Generative Adversarial Networks: Build Your First Models [Електронний ресурс] // Real Python – URL: https://realpython.com/generative-adversarial-networks/ . (дата звернення – 24.04.2023)
20. Python [Електронний ресурс] // Python – URL: https://www.python.org/ . (дата звернення – 25.04.2023)
21. PyTorch [Електронний ресурс] – URL: https://pytorch.org/docs/stable/torch.html#tensors. (дата звернення – 28.04.2023)
22. Math documentation [Електронний ресурс] – URL: https://docs.python.org/3/library/math.html. (дата звернення – 28.04.2023)
23. C++ Introduction [Електронний ресурс] // W3schools – URL: https://www.w3schools.com/cpp/cpp_intro.asp#:~:text=C%2B%2B%20is%20a%20cross%2Dplatform,over%20system%20resources%20and%20memory. (дата звернення – 13.05.2023)

24. C Tutorial [Електронний ресурс] // W3schools – URL: https://www.w3schools.com/c/index.php . (дата звернення – 13.05.2023)
25. Keras. Documentation [Електронний ресурс] // Keras – URL: https://keras.io/about/. (дата звернення – 13.05.2023)
26. TensorFlow Documentation [Електронний ресурс] // TensorFlow – URL: https://www.tensorflow.org/about. (дата звернення – 13.05.2023)
27. Scikit-learn [Електронний ресурс] // Scikit-learn – URL: https://scikit-learn.org/stable/ . (дата звернення – 13.05.2023)
28. Colab розв'язок задачі sin [Електронний ресурс] // Colab – URL: https://colab.research.google.com/drive/1FxNtJpbf-Tg1AiDTtQ1SAGP5UgBKwgsB?usp=sharing (дата звернення – 29.04.2023)
29. Colab розв'язок задачі handwritten digits [Електронний ресурс] // Colab – URL: https://colab.research.google.com/drive/1sNNq_Wcj_fEPIEfkhf_7KTQhlHSRt7Uh?usp=sharing (дата звернення – 29.04.2023)
30. Hui J. GAN — Why it is so hard to train Generative Adversarial Networks! [Електронний ресурс] / Jonathan Hui // Medium. – 2018. – URL: https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b#4987. (дата звернення – 18.05.2023)
31. Hui J. GAN — Ways to improve GAN performance [Електронний ресурс] / Jonathan Hui // TowardsDataScience. – 2018. – URL: https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b. (дата звернення – 18.05.2023)
32. Petrova O. Semi-Supervised Learning with GANs [Електронний ресурс] / Olga Petrova // Medium. – 2020. – URL: https://medium.com/scaleway-cloud/semi-supervised-learning-with-gans-a-tale-of-cats-and-dogs-3c90acfe91c9. (дата звернення – 18.05.2023)

33. Softmax function [Електронний ресурс] // DeepAI – URL: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer. (дата звернення – 18.05.2023)
34. Kaushik T. Semi-supervised learning with Generative Adversarial Networks [Електронний ресурс] / Tryambak Kaushik – URL: https://www.kdnuggets.com/2020/01/semi-supervised-learning-generative-adversarial-networks.html. (дата звернення – 18.05.2023)
35. van de Wolfshaar J. Semi-supervised learning with GANs [Електронний ресурс] / Jos van de Wolfshaar // Medium. – 2018. – URL: https://medium.com/@jos.vandewolfshaar/semi-supervised-learning-with-gans-23255865d0a4. (дата звернення – 18.05.2023)
36. Hui J. GAN — Some cool applications of GAN [Електронний ресурс] / Jonathan Hui // Medium. – 2018. – URL: https://jonathan-hui.medium.com/gan-some-cool-applications-of-gans-4c9ecca35900. (дата звернення – 20.05.2023)

ДОДАТОК А

Система генерування навчальної вибірки в задачах напівкерованого навчання на основі генеративно-змагальних мереж

Виконала:
Маєвська Катерина
Науковий керівник:
д.н.т. проф. Синєглазов Віктор Михайлович

Вступ

Предмет дослідження

Застосування генеративно-змагальних мереж для розв'язку задач напівкерованого навчання.

Об'єкт дослідження

Об'єктом дослідження було обрано генеративно-змагальні мережі та напівкероване навчання.

Мета дослідження

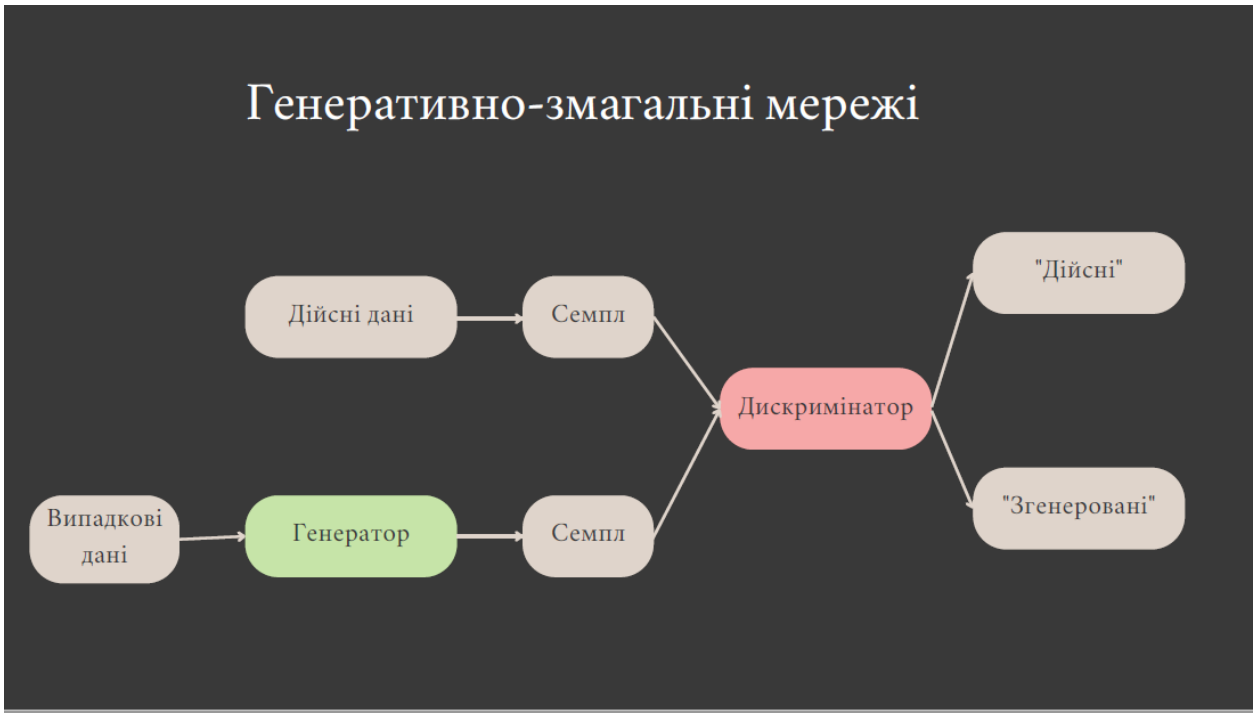
Дослідити існуючі алгоритми генерації навчальної вибірки в задачах напівкерованого навчання за допомогою генеративно-змагальних мереж, оптимізувати та реалізувати.

Актуальність роботи

Застосування алгоритмів напівкерованого навчання дозволить будувати моделі навчання при наявності лише невеликої кількості даних, тобто допоможе розв'язати проблему витратності процесу збору навчальної вибірки та буде корисним у випадку роботи з конфіденційними даними.

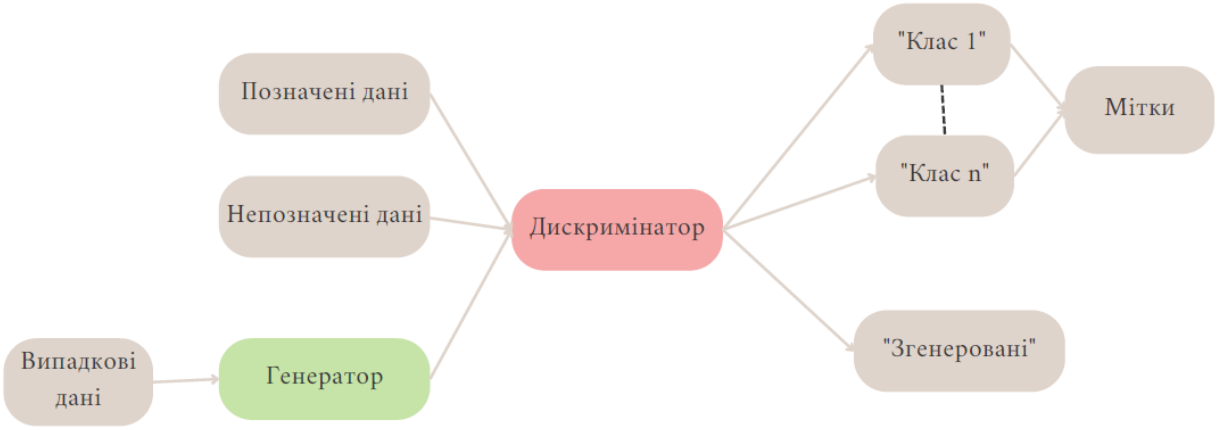
Постановка задачі

- Проаналізувати літературні джерела та зробити огляд існуючих теоретичних відомостей щодо генеративно-змагальних мереж та напівкерованого навчання;
- Розробити та реалізувати модель напівкерованого навчання з використанням генеративно-змагальних мереж. Провести тестування;
- Аналіз результатів. Порівняльний аналіз роботи розробленої моделі та звичайної багатошарової нейронної мережі. Висновки.


Генеративно-змагальні мережі
Дійсні дані
Семпл
"Дійсні"
Дискримінатор
Випадкові дані
Генератор
Семпл
"Згенеровані"


Напівкероване навчання
Кероване навчання
Вся вибірка позначена
Модель
Напівкероване навчання
Невелика частина вибірки позначена
Більша частина вибірки непозначена
Модель
Некероване навчання
Вся вибірка непозначена
Модель

Напівкероване навчання з використанням генеративно-змагальних мереж



Основні відмінності

- **Дискримінатор - мережа класифікації** n+1 **класів**
- **На вхід подаються дані трьох типів:**
  - Позначена дійсна вибірка;
  - Непозначена дійсна вибірка, всі елементи якої належать до К класів (К - відоме) представлених у першій вибірці;
  - Згенерована вибірка – результат роботи генератора.
- **Приховані шари - згорткові**
- **Функція активації** - softmax

Вхідні дані

MNIST

В даній задачі ми використовуємо відкритий та загальнодоступний набір даних MNIST, що складається за набору зображень рукописних цифр 28 на 28 пікселів у чорно-білій кольоровій гамі та позначень, що відповідають зображеним цифрам.

Розмір вибірки

Позначені дані - 6000 зображень
Непозначені дані - 60000 зображень (весь датасет)
Тестова вибірка - 10000 зображень

Архітектура

class Discriminator():

Мережа дискримінатора. Складається з 4 шарів:
Вхідний
Приховані - згорткові, мають функцію активації LeakyReLu
Вихідний - має 2 функції активації:
sigmoid - для задачі розпізнавання згенерованих даних
softmax - для задачі класифікації

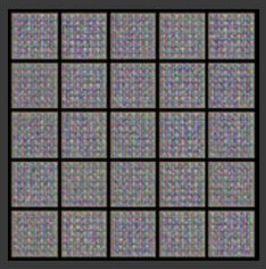
class Generator():

Мережа генератора.
Складається з 4 шарів:
Вхідний - лінійний
Приховані - згорткові, мають функцію активації LeakyReLu
Вихідний - має функцію активації тангенс гіперболічний

Навчання

- 2 вкладених цикли
- Крок навчання генератора і дискримінатора - 1:1
- Навчання генератора відбувається як у стандартній генеративно-змагальній мережі
- Навчання дискримінатора складається з 2 частин:
  - керована - задача класифікації
  - некерована - задача розпізнавання дійсні/недійсні

Аналіз результатів



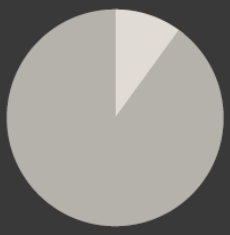
Середнє значення функції втрат: 0.9835
Точність: 0.9862 (99%)

```
Тестова вибірка: Середня втрата: 0.9835, Точність: 9862/10000 (99%)

справжні  вгадані
  7         7
  8         8
  1         1
  9         9
  4         4
  7         7
  8         8
  2         2
```

Порівняльний аналіз

Характеристика	Багатошарова нейронна мережа	Розроблена модель
Кількість епох навчання	20	20
Розмір розміченої вибірки	60000	6000
Розмір нерозміченої вибірки	-	60000
Точність	97%	99%
Втрата	0.14	0.9835
Час навчання	~3 хвилини	~2 години



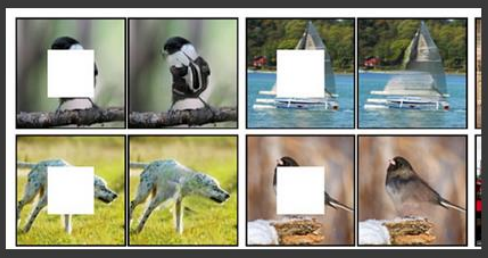
10% позначених даних

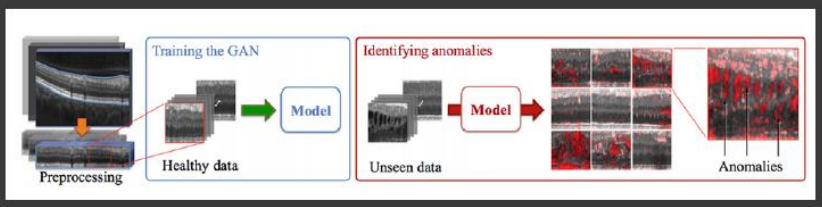
точність 99%

Висновки

- Було проведено аналіз теоретичних відомостей в області штучного інтелекту, штучних нейронних мереж, машинного навчання та їх класифікацій
- Було проведено аналіз будови та принципу роботи генеративно-змагальних мереж
- Було проведено аналіз алгоритмів напівкерованого навчання та інтеграції генеративно-змагальних мереж
- Було розроблено алгоритм напівкерованого навчання на основі генеративно-змагальних мереж для розв'язку задачі розпізнавання рукописних зображень, побудовано програмну реалізацію
- Було побудовано розв'язок задачі за допомогою звичайної багатошарової нейронної мережі, проведено порівняльний аналіз. Розроблена модель показала високі результати.

Перспективні сфери застосування





Подальші дослідження

1. Вдосконалення мережі для зменшення часу навчання та розміру вибірки
2. Робота з кольоровими зображеннями, з літерами
3. Розробка графічного користувацького інтерфейсу

Дякую за увагу!

ДОДАТОК Б

```
from __future__ import print_function
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = "1"
import fileOperations
import random
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import time
from torchsummary import summary

def define_vars():
    """
    визначаємо глобальні змінні як параметри середовища
    """
    try:
        # Кількість каналів зображення. Кольорове - 3, чорно-біле - 1
        os.environ['CHANNEL_NUM'] = '1'

        # Розмірність вхідного вектора генератора
        os.environ['GEN_INPUT'] = '100'

        # Кількість навчальних епох
        os.environ['EPOCHS_NUM'] = '20'

        # Тека датасету
        os.environ['DATAROOT'] = '/content'

        # Розмір навчального батчу
        os.environ['BATCH_SIZE'] = '25'

        # Розмір зображень
        os.environ['IMAGE_SIZE'] = '28'

        # Learning rate
        os.environ['LEARNING_RATE'] = '0.002'

        # Beta1 гіперпараметр для алгоритму Адам
        os.environ['BETA1'] = '0.5'

        # Кількість доступних GPU.
        os.environ['GPU_NUM'] = '1'

        os.environ['NUM_CLASSES'] = '10'

    except Exception as e:
        raise Exception("Error while running define_vars: {}".format(e))



def main():
    try:
        manualSeed = 999
        random.seed(manualSeed)
```

```
        torch.manual_seed(manualSeed)

        # тека датасету
        directory = "imagesMNIST"
        if not os.path.exists(directory):
            os.makedirs(directory)
        define_vars()

        gpu_num = int(os.environ['GPU_NUM'])
        # налаштунки
        device = torch.device("cuda:0" if (torch.cuda.is_available() and
gpu_num > 0)
                              else "cpu")

        train_loader = fileOperations.download_mnist(device)

        netG = fileOperations.Generator(gpu_num).to(device)
        netG.apply(fileOperations.define_weights)
        netD = fileOperations.Discriminator(gpu_num,
num_classes=int(os.environ['NUM_CLASSES'])).to(device)
        netD.apply(fileOperations.define_weights)

        noise = torch.randn(int(os.environ['BATCH_SIZE']),
int(os.environ['GEN_INPUT']), 1, 1, device=device)
        fake = netG(noise)
        noise = torch.randn(int(os.environ['BATCH_SIZE']), 1, 28, 28,
device=device)
        adv, aux = netD(noise)

        # створюємо функції втрат

        criterion_adv = nn.BCELoss()
        criterion_aux = nn.CrossEntropyLoss()

        fixed_noise = torch.randn(int(os.environ['BATCH_SIZE']),
int(os.environ['GEN_INPUT']), 1, 1, device=device)

        # Встановлюємо позначки для дійсних та згенерованих даних
        real_label = 1
        fake_label = 0

        # Встановлюємо оптимізатор Адам
        optimizerD = optim.Adam(netD.parameters(),
lr=float(os.environ['LEARNING_RATE']), betas=(float(os.environ['BETA1']),
0.999))
        optimizerG = optim.Adam(netG.parameters(),
lr=float(os.environ['LEARNING_RATE']), betas=(float(os.environ['BETA1']),
0.999))

        # Створюємо теку для моделей
        os.environ['CHECKPOINT_DIR'] = '/content/MNIST_Check/'
        if not os.path.exists(os.environ['CHECKPOINT_DIR']):
            os.makedirs(os.environ['CHECKPOINT_DIR'])


        os.environ['NUM_EPOCH'] = '20'

        supervised_batch = int(os.environ['BATCH_SIZE']) // 10

        netD.train()
        netG.train()
        t0 = time.time()
        # Цикл навчання
```

```
        ###################
        # print("Start training")
        # G_losses, D_losses  = fileOperations.train_gan(train_loader,
device, netG, netD, supervised_batch, optimizerD, optimizerG,
        #                           criterion_aux, criterion_adv, fixed_noise,
t0)
        #
        # plt.figure(figsize=(10, 5))
        # plt.title("Втрати дискримінатора під час навчання")
        # #plt.plot(G_losses, label="G")
        # plt.plot(D_losses, label="D")
        # plt.xlabel("ітерації")
        # plt.ylabel("Втрата")
        # plt.legend()
        # plt.show()
        # print("Training completed")

        ##################


        # Завантажуємо збережену модель

        checkpoint_dir = os.environ['CHECKPOINT_DIR']
        print(checkpoint_dir)
        GAN_path = os.path.join(checkpoint_dir, 'GAN.pkl')

        if (device):
            checkpoint = torch.load(GAN_path, map_location='cpu')

        netD.load_state_dict(checkpoint['D_state_dict'])
        netG.load_state_dict(checkpoint['G_state_dict'])
       # print(summary(netG, (1, 100, 1)))

        test_loader = fileOperations.load_test_dataset()
        real_batch = next(iter(test_loader))
        plt.figure(figsize=(5, 5))
        plt.axis("off")
        plt.title("Тестування")

plt.imshow(np.transpose(vutils.make_grid(real_batch[0].to(device)[:25],
                                                 padding=2, normalize=True,
                                                 nrow=5).cpu(),
                                (1, 2, 0)))

        netD.eval()
        test_loss = 0
        correct = 0

        with torch.no_grad():
            for imgs, labels in test_loader:
                adv, aux = netD(imgs.to(device))
                test_loss += F.nll_loss(aux, labels.to(device),
reduction='sum').item()
                pred = aux.data.max(1, keepdim=True)[1]
                correct +=
pred.eq(labels.to(device).data.view_as(pred)).sum()

        test_loss /= len(test_loader.dataset)
        test_loss *= (-1)
        # test_loss.append(test_loss)

        print('\nТестова вибірка: Середня втрата: {:.4f}, Точність: {}/{}
({:.0f}%)\n'.format(
```

```
            test_loss, correct, len(test_loader.dataset),
            100. * correct / len(test_loader.dataset)))

        images, labels = next(iter(test_loader))

        no_of_data = min(30, images.shape[0])

        # Виводимо зображення
        plt.figure(figsize=(no_of_data, no_of_data))

plt.imshow(np.transpose(vutils.make_grid(images.to(device)[:no_of_data],
                                         padding=2, normalize=True,
                                         nrow=no_of_data).cpu(),
                                (1, 2, 0)))

        adv, aux = netD(images.to(device))
        _, predicted = torch.max(aux, 1)

        print('справжні  вгадані')
        for j in range(no_of_data):
            print('  {}          {}'.format(labels[j].item(),
predicted[j].item()))

        plt.show()
    except Exception as e:
        raise Exception("Error while running main file: {}".format(e))


if __name__ == '__main__':
    main()
```