

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту**

«На правах рукопису»
УДК 004.032.26+519.711](043.3)

До захисту допущено:
В.о. завідувача кафедри
_____ Ірина ДЖИГИРЕЙ
«__» _____ 2024 р.

**Магістерська дисертація
на здобуття ступеня магістра
за освітньо-професійною програмою «Системи і методи штучного
інтелекту»
зі спеціальності 122 «Комп'ютерні науки»
на тему: «Структурно-параметричний синтез генеративних нейронних
мереж»**

Виконав:
студент II курсу, групи КІ-21мп
Гречухов Микита Іванович

Науковий керівник:
проф. кафедри ШІ, д.т.н., професор,
Синєглазов Віктор Михайлович

Рецензент:
доц. кафедри ММСА, к.ф.-м.н., доцент
Шубенкова Ірина Анатоліївна

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів без
відповідних посилань
Студент: _____

Київ – 2024

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

Рівень вищої освіти – другий (магістерський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В.о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___» _____ 2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Гречухову Микиті Івановичу

1. Тема дисертації «Структурно-параметричний синтез генеративних нейронних мереж», науковий керівник дисертації Синєглазов Віктор Михайлович, д.т.н., професор, затверджені наказом по університету від «08» листопада 2023 р. №5200-с
2. Термін подання студентом дисертації 08.01.24
3. Об'єкт дослідження: структурно-параметричний синтез генеративних нейронних мереж
4. Вихідні дані: модель генеративно змагальної мережі із покращенням
5. Перелік завдань, які потрібно розробити:
 - 1) Здійснити огляд технічної літератури за темою роботи;
 - 2) Дослідити актуальність обраної теми;
 - 3) Ознайомитися із існуючими методами та моделями побудови генеративно-змагальних нейронних мереж;

- 4) Здійснити порівняльний аналіз наявних методів, виявити переваги та недоліки;
- 5) Розробити та реалізувати програмний продукт, що використовує апарат методів машинного навчання та вирішує задачу генерації даних;
- 6) Провести експеримент, що засвідчує працеспроможність запропонованої моделі, сформулювати інтерактивний звіт задачі аналізу, виконати аналіз результатів;
- 7) Розробити концептуальні висновки;
- 8) Підготувати пояснювальну записку;

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: презентація.

7. Дата видачі завдання 30.08.2023.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Вивчення літератури за темою роботи.	01.09.2023 - 25.09.2023	Виконано
2	Підготовка першого розділу	25.09.2023 - 08.10.2023	Виконано
3	Підготовка другого розділу	09.10.2023 - 23.10.2023	Виконано
4	Розробка програмного продукту.	23.10.2023 - 30.11.2023	Виконано
5	Підготовка третього розділу.	01.12.2023 - 17.12.2023	Виконано
6	Підготовка четвертого розділу.	18.12.2023 - 21.12.2023	Виконано
7	Підготовка частини стартап-проєкту.	22.12.2023 - 31.12.2023	Виконано
8	Концептуальні висновки. Перспективи розвитку отриманих рішень.	01.01.2024 - 02.01.2024	Виконано
9	Оформлення пояснювальної записки	02.01.2024 - 07.01.2024	Виконано

Студент

Микита ГРЕЧУХОВ

Науковий керівник

Віктор СИНЄГЛАЗОВ

РЕФЕРАТ

Магістерська дисертація: 100 с., 19 табл., 30 рис., 21 посилання, додаток.

СТРУКТУРНО-ПАРАМЕТРИЧНИЙ СИНТЕЗ, ГЕНЕТИЧНИЙ АЛГОРИТМ, ГЕНЕРАТИВНО-ЗМАГАЛЬНІ МЕРЕЖІ, ГІБРИДНИЙ АЛГОРИТМ НАВЧАННЯ, ГЕНЕРАТОР, ДИСКРИМІНАТОР.

Тема є актуальною для дослідження через великий попит на генерацію різних типів даних.

Об'єктом дослідження є структурно-параметричний синтез генеративних нейронних мереж.

Предметом дослідження є генеративно-змагальні мережі, та способи їх покращення.

Мета роботи – розробити генеративно-змагальну мережу із покращенням ефективності навчання і, як результат, пришвидшенням пошуку оптимального рішення у порівнянні із звичайною GAN моделлю.

У роботі розглянуто, описано та використано різні методи та архітектури генеративно-змагальних мереж.

У результаті побудовано модель GAN із застосуванням генетичного алгоритму, що розв'язує поставлену в роботі задачу.

ABSTRACT

Master's thesis: 100 pages, 19 tables, 30 figures, 21 sources, appendix.

STRUCTURAL-PARAMETRIC SYNTHESIS, GENETIC ALGORITHM,
GENERATIVE-ADVERSARIAL NETWORKS, HYBRID LEARNING
ALGORITHM, GENERATOR, DISCRIMINATOR.

The topic is relevant for research due to the great demand for the generation of various types of data.

The object of the research is the structural-parametric synthesis of generative neural networks.

The subject of research is generative-competitive networks and methods of their improvement.

The purpose of the work is to develop a generative-competitive network with improved learning efficiency and, as a result, speeding up the search for the optimal solution compared to the usual GAN model.

Various methods and architectures of generative-competitive networks are considered, described and used in the work.

As a result, a GAN model was built using a genetic algorithm that solves the problem set in the paper.

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1 ШТУЧНИЙ ІНТЕЛЕКТ І ЙОГО РОЗВИТОК У СВІТІ.....	10
1.1 Штучний інтелект – світовий тренд.....	10
1.2 Розвиток штучного інтелекту в Україні.....	11
1.3 ШІ та його складові.....	13
1.4 Коротка історія і сучасний вигляд предметної області.....	17
Висновки до розділу 1.....	18
РОЗДІЛ 2 ДОСЛІДЖЕННЯ ОБЛАСТІ ЗАДАЧІ СТРУКТУРНО-ПАРАМЕТРИЧНОГО СИНТЕЗУ ТА СУЧАСНИХ МОДЕЛЕЙ ГЕНЕРАТИВНО-ЗМАГАЛЬНИХ МЕРЕЖ.....	19
2.1 Класична модель GAN.....	19
2.2 Опис існуючих моделей GAN.....	22
2.2.1 DCGAN.....	23
2.2.2 WGAN.....	24
2.2.3 WGAN-GP.....	26
2.2.4 SGAN.....	27
2.2.5 CGAN.....	29
2.2.6 Складений або прогресивний GAN.....	30
2.3 Алгоритм навчання GAN.....	31
2.4 Проблеми що виникають при навчанні GAN.....	33
2.4.1 Згортання режиму (Mode Collapse).....	33
2.4.1.1 Катастрофічне забування.....	34
2.4.1.2 Перенавчання дискримінатора.....	37
2.4.2 Рівновага Неша.....	38
2.4.3 Зникаючий градієнт.....	40
2.5 Метрики що застосовуються до GAN.....	41
2.5.1 Inception Score (IS).....	42

2.5.2 Fréchet Inception Distance (FID).....	44
2.5.3 Точність, повнота та оцінка F1.....	45
Висновки до розділу 2.....	47
РОЗДІЛ 3 ГІБРИДНИЙ АЛГОРИТМ МАШИННОГО НАВЧАННЯ	
ГЕНЕРАТИВНО-ЗМАГАЛЬНОЇ МЕРЕЖІ.....	49
3.1 Огляд робіт з машинного навчання генеративно-змагальних мереж.....	49
3.2 Постановка задачі.....	51
3.3 Структура хромосоми генетичного алгоритму.....	54
3.4 Вибір функції втрат.....	57
3.6 Визначення гіперпараметрів.....	58
3.5 Структура гібридного алгоритму машинного навчання генеративно-змагальної мережі.....	59
3.6 Налаштування гіперпараметрів.....	61
3.7 Аналіз отриманих результатів.....	62
Висновки до розділу 3.....	69
РОЗДІЛ 4 РОЗРОБКА ВЛАСНОГО СТАРТАП ПРОЄКТУ.....	70
4.1 План розробки стартапу та масштабування його на ринок.....	70
4.2 Опис ідеї стартап-проєкту.....	72
4.3 Технологічний аудит ідеї проєкту.....	73
4.4 Аналіз ринкових можливостей запуску стартап-проєкту.....	74
4.5 Розроблення ринкової стратегії стартап-проєкту.....	81
4.6 Розроблення маркетингової програми стартап-проєкту.....	83
Висновки до розділу 4.....	84
ВИСНОВКИ.....	86
ПЕРЕЛІК ПОСИЛАНЬ.....	87
ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО ПРОДУКТУ.....	90

ВСТУП

В сучасному світі існує багато проблем що потребують креативного підходу. Один з них це застосування GAN моделей.

Генеративно-змагальні системи – одна з найуспішніших і широко поширених застосувань технологій машинного навчання. За допомогою цього алгоритму можна створити безліч нових, не схожих одне на одного зображень, відео, аудіо, текстів, або іншої інформації.

Зараз існує багато методів для побудови генеративно-змагальних систем, що допомагають в різних сферах – кіно, музика, реклама. Майже ні один сервіс не існує без цих систем, навіть найбільш примітивних.

GAN моделі займають важливе місце в процесі життєдіяльності багатьох великих компаній. Багато сучасних компаній використовує даний підхід для покращення своєї роботи.

NVIDIA використовує GAN моделі для різноманітних завдань, включаючи генерацію відео, зображень та структур молекул для наукових досліджень. Їхній StyleGAN був використаний для створення реалістичних портретів людей, які не існують.

OpenAI випустили ряд GAN-моделей, включаючи GPT-3, що використовується для генерації тексту та розв'язання різних завдань у сфері обробки природної мови. Цей інструмент застосовується в онлайн-ресурсах та додатках, наприклад, для генерації контенту та підтримки чат-ботів.

Adobe використовує GAN для розвитку та вдосконалення своїх графічних програм, де StyleGAN та інші моделі використовуються для автоматичного редагування зображень, зміни стилю фотографій та покращення візуальних ефектів.

Facebook використовує GAN для покращення автозаповнення фотографій у своєму соціальному медіа. Це допомагає користувачам зекономити час при додаванні підписів до своїх фотографій.

IBM використовує GAN для розробки систем, які генерують синтетичні дані для навчання та тестування моделей штучного інтелекту в області медицини, фінансів та інших галузей.

У виробництві анімаційних фільмів GAN використовуються для створення більш реалістичних персонажів та візуальних ефектів. Disney Research також працює над використанням GAN для автоматичного анімування персонажів.

Tesla використовує GAN для обробки та аналізу відеоданих від автомобільних камер. Це допомагає вдосконалити системи автономного водіння та забезпечити безпечну їзду.

Це лише кілька прикладів компаній, які використовують GAN моделі у своїй діяльності. Застосування GAN може бути знайдено в багатьох інших галузях.

Але завжди є шляхи покращення: наприклад, популярний наразі бот створений компанією OpenAI ChatGPT став сенсацією 2022-2023 року, і продовжує вдосконалюватися.

Тому в своїй магістерській дисертації я обрав тему структурно-параметричний синтез генеративних нейронних мереж, в рамках якої проведу дослідження існуючих методів і запропоную власне покращення.

РОЗДІЛ 1 ШТУЧНИЙ ІНТЕЛЕКТ І ЙОГО РОЗВИТОК У СВІТІ

1.1 Штучний інтелект – світовий тренд

Штучний інтелект (ШІ) визначає сучасну епоху і зазнає шаленої популярності як світовий тренд. Завдяки стрімкому розвитку технологій, ШІ стає ключовим фактором у багатьох сферах нашого життя, починаючи від бізнесу та закінчуючи охороною здоров'я та науковими дослідженнями.

Бізнес-середовище вже відчуває вплив ШІ через впровадження автоматизації та підвищення продуктивності. Системи машинного навчання допомагають аналізувати великі обсяги даних, надаючи компаніям цінний інсайт щодо покращення стратегій та взаємодії з клієнтами.

Промисловість також переживає перетворення завдяки ШІ. Автоматизація виробництва, прогнозування попиту, оптимізація ланцюга постачання – усе це стає реальністю завдяки впровадженню інтелектуальних систем. ШІ допомагає ефективно використовувати ресурси, знижуючи витрати та збільшуючи продуктивність.

Технологічні гіганти вкладають значні зусилля в розвиток ШІ. Від голосових асистентів до систем розпізнавання облич і автономних автомобілів, ШІ стає необхідною складовою у світі новітніх технологій.

У галузі охорони здоров'я ШІ прискорює діагностику та лікування захворювань. Використання алгоритмів машинного навчання для аналізу медичних зображень та прогнозування хвороб дозволяє швидше та ефективніше надавати медичну допомогу.

У сфері освіти ШІ трансформує навчання, надаючи персоналізовані програми та автоматизовані засоби оцінювання. Електронні платформи з машинним навчанням сприяють розвитку навичок майбутнього.

Країни та компанії активно конкурують за лідерство у розробці та впровадженні ШІ. Інвестиції у дослідження та розвиток, навчання спеціалістів у

галузі штучного інтелекту – це лише частина глобальної ШІ-гонки. Успішні ініціативи можуть визначити технологічну перспективу країни.

Сучасні проблеми екології стають більш управліними завдяки ШІ. Використання алгоритмів для моніторингу та аналізу даних допомагає виявляти екологічні зміни та розробляти стратегії їх управління. Впровадження "зелених" технологій стає пріоритетом.

Зростаюча потужність ШІ ставить перед суспільством низку етичних питань. Від автономних систем управління до ефективного розподілу ресурсів, розгляд питань, пов'язаних із безпекою та справедливістю, стає все більш важливим завданням.

Незважаючи на всі позитивні аспекти, існують і виклики, пов'язані з етикою використання ШІ, приватністю та можливістю втрати робочих місць через автоматизацію.

ШІ не лише технологічний тренд, але і суспільний феномен, що визначає наше сприйняття інновацій. Велике питання полягає в тому, як ми будемо використовувати цю потужну технологію в майбутньому і як вона вплине на наш спосіб життя.

1.2 Розвиток штучного інтелекту в Україні

Україна активно адаптується до світових тенденцій та впроваджує технології штучного інтелекту (ШІ) в різних галузях. Наприклад, в медичних центрах країни використовуються нейронні мережі для лікування безпліддя. Одна з клінік впровадила два штучних інтелекти, які працюють значно швидше, ніж людський мозок, та допомагають у глибокій діагностиці ембріонів та прогнозуванні можливості настання вагітності.

Міністерство цифрової трансформації також планує впроваджувати технології штучного інтелекту у державних сервісах для оптимізації та полегшення їхньої роботи.

«Або ти почнеш використовувати штучний інтелект, або він замінить тебе. Багато людей бояться штучного інтелекту і говорять: “А що ж буде там з тисячами держслужбовців? А що ж буде з тисячами співробітників компаній?”. Я вважаю, що навпаки відкривається потужний ринок, потрібно вивчати штучний інтелект. З’явиться багато нових спеціалістів, нових професій. Це тренд, який потрібно очолити, а не тікати від нього», – говорить міністр цифрової трансформації Михайло Федоров.

Штучний інтелект, крім того, використовується для ефективного протистояння російській армії. Деякі бойові системи та ШІ-технології вже успішно застосовуються в Збройних силах України.

У лютому 2022 року стартап Reface оголосив про розробку алгоритму для визначення російських військ за допомогою супутникових знімків. Навчальні дані для нейромережі були зібрані спільно з громадянами України.

З моменту березня 2022 року Міністерство оборони України почало використовувати систему розпізнавання облич Clearview AI. Компанія дала згоду на використання своєї технології для ідентифікації осіб російських військових, протидії дезінформації, визначення загиблих та перевірки людей на контрольно-пропускних пунктах.

За словами міністра цифрової трансформації України Михайла Федорова, система розпізнавання облич дозволила встановити особи російських військових, що брали участь у нападі на цивільних у Київській області. Крім того, українська компанія YouControl спільно з ШІ-підприємством Artellence та з підтримкою СБУ представили додаток "ТиХто", який дозволяє швидко виявляти підозрілі особи.

Штучний інтелект також застосовується в процесі розмінування територій, що були звільнені від окупації в Україні. Наприклад, Сполучені Штати передали

Україні робота Spot від Boston Dynamics, який може безпечно виявляти та усувати мінометні снаряди та касетні боєприпаси.

Розробник інтерактивної онлайн-карти подій на фронті в Україні, Deer State UA, взаємодіє з творцями автоматизованої системи обробки даних Griselda. Ця співпраця спрямована на поліпшення точності та оперативності обробки та оновлення інформації.

1.3 ШІ та його складові

Штучний інтелект (ШІ)—це галузь комп'ютерних наук, що ставить за мету створення програм та систем, які можуть виконувати завдання, які вимагають інтелектуального мислення. Ця область включає різні складові, які спільно працюють для досягнення інтелектуальних функцій. Основні складові ШІ включають:

1. Машинне навчання (МН): Машинне навчання є ключовою складовою ШІ, в якій алгоритми отримують здатність вдосконалювати свою продуктивність на основі даних. Вони вчаться без прямого програмування, використовуючи моделі та алгоритми, які дозволяють системі робити прогнози та приймати рішення на основі досвіду.

2. Обробка природної мови (NLP): Ця складова дозволяє системам розуміти та взаємодіяти з людьми через природну мову. Вона включає в себе розпізнавання мови, розуміння тексту, генерацію мовлення та інші техніки, щоб системи могли працювати з людьми на їхньому рівні мовлення.

3. Комп'ютерне зорове сприйняття (CV): Ця складова забезпечує системам можливість розпізнавати та розуміти візуальну інформацію, подібно тому, як це робить людське зорове сприйняття. Системи CV використовують алгоритми обробки зображень для аналізу та розпізнавання об'єктів, облич, рухів тощо.

4. Логічне програмування та планування: Ця складова відповідає за створення систем, які можуть робити логічні висновки та розробляти стратегії для досягнення певних цілей. Вона використовується для розробки алгоритмів прийняття рішень та планування дій.

5. Робототехніка та автономні системи: Системи ШІ також включають в себе розробку автономних агентів та роботів, які можуть самостійно виконувати завдання, взаємодіяти з оточенням та приймати рішення на основі зібраної інформації.

Класифікатори в сфері штучного інтелекту (ШІ) є моделями або алгоритмами, які використовуються для призначення об'єктів чи даних до визначених класів чи категорій на основі їхніх властивостей або характеристик. Ось деякі типові класифікатори в області штучного інтелекту:

1. Логістична регресія.

Опис: Логістична регресія використовує логістичну функцію для моделювання ймовірності того, що об'єкт належить до певного класу.

Застосування: Використовується для бінарної класифікації.

2. Метод опорних векторів (SVM).

Опис: SVM шукає гіперплощину, яка найкращим чином розділяє дані на класи в просторі властивостей.

Застосування: Використовується для бінарної та мультикласової класифікації.

3. Нейронні мережі.

Опис: Використовують штучні нейрони та шари для навчання на великих наборах даних та вирішення різноманітних задач класифікації.

Застосування: Від бінарної класифікації до складних мультикласових завдань.

4. Древа рішень.

Опис: Представляють собою ієрархічну структуру, яка допомагає вирішувати задачі шляхом послідовного прийняття рішень.

Застосування: Використовується для класифікації та регресії.

5. Випадковий ліс.

Опис: Ансамбль дерев рішень, який використовується для зменшення перенавчання та підвищення точності.

Застосування: Використовується для класифікації та регресії.

6. k-найближчих сусідів (k-NN).

Опис: Класифікує об'єкти на основі класів їхніх k найближчих сусідів у просторі ознак.

Застосування: Використовується для класифікації та регресії.

Ці класифікатори представляють лише деякі з різноманітних методів, які використовуються в області штучного інтелекту для вирішення завдань класифікації. Зазвичай вибір конкретного класифікатора залежить від характеристик даних та конкретних вимог задачі.

При навчанні нейронних мереж використовуються різноманітні алгоритми для налаштування ваг нейронів і досягнення оптимальних параметрів моделі. Основні алгоритми охоплюють.

1. Алгоритм зворотного поширення помилок (Backpropagation).

Опис: Зворотне поширення є найпоширенішим алгоритмом для навчання нейронних мереж. Він базується на методі градієнтного спуску, де ваги оновлюються в напрямку, протилежному градієнту функції втрати.

Застосування: Використовується для багатьох типів нейронних мереж, включаючи звичайні штучні нейронні мережі (ANN) та глибокі нейронні мережі (DNN).

2. Оптимізатори градієнтного спуску

Опис: Різноманітні алгоритми оптимізації, такі як SGD (Stochastic Gradient Descent), Adam, RMSprop, Adagrad тощо, застосовуються для ефективного налаштування ваг та прискорення збіжності.

Застосування: Використовуються в глибоких нейронних мережах для оптимізації функції втрати.

3. Функції активації.

Опис: Функції активації, такі як ReLU (Rectified Linear Unit), Sigmoid, Hyperbolic Tangent (tanh) і інші, дозволяють введеному сигналу проходити через нейрони та надають мережі нелінійність.

Застосування: Використовуються для активації нейронів у різних шарах мережі.

4. Функції втрати (Loss Functions).

Опис: Функції втрати визначають, наскільки добре модель вирішує завдання. Для задач класифікації використовуються, наприклад, кросс-ентропія або середньоквадратична помилка для регресії.

Застосування: Використовуються для визначення ефективності нейронної мережі під час навчання.

5. Функції регуляризації.

Опис: Функції регуляризації, такі як L1 та L2 регуляризація, допомагають уникнути перенавчання та підвищити стійкість моделі.

Застосування: Використовуються для контролю величини ваг та запобігання перенавчанню.

6. Dropout.

Опис: Dropout – це техніка, яка випадковим чином вимикає нейрони під час тренування, що сприяє створенню більш стійких та ефективних моделей.

Застосування: Використовується для зменшення перенавчання.

Ці алгоритми та техніки взаємодіють для ефективного навчання нейронних мереж і досягнення оптимальних результатів у вирішенні різноманітних завдань.

У цій роботі об'єктом дослідження є генеративні нейронні мережі. Найактуальнішим на сьогодні є використання GAN (Generative Adversarial Network), генеративно-змагальні мережі, вид архітектури що складається з декількох нейромереж що конкурують одна з одною. Тому, вважаю доцільним розглянути історію та сучасний вигляд предметної області пов'язаної із GAN.

1.4 Коротка історія і сучасний вигляд предметної області

Історія розвитку GAN (Generative Adversarial Network) моделей включає в себе декілька ключових подій та етапів.

У 2014 році Іан Гудфеллоу, Юссеф Бенжіо та інші дослідники опублікували статтю "Generative Adversarial Nets," в якій представили концепцію GAN. Ця концепція включає дві нейронні мережі, генератор та дискримінатор, які змагаються одна з одною. Генератор створює синтетичні дані, а дискримінатор намагається відрізнити їх від реальних даних. Процес навчання GAN призводить до покращення генератора та дискримінатора.

У 2015 році були представлені різноманітні модифікації GAN, включаючи Conditional GAN (cGAN), який дозволяє керувати генерацією за допомогою умови, і деякі інші вдосконалені архітектури, які розширюють можливості GAN.

У 2016 році DCGAN (Deep Convolutional GAN) був представлений як архітектура, оптимізована для генерації зображень за допомогою глибоких згорткових мереж. Це допомогло зробити генеровані зображення більш реалістичними.

У 2017 році були представлені різні інші важливі модифікації GAN, включаючи WGAN (Wasserstein GAN), який вводить новий підхід до функцій витрати, який сприяє стабільному навчанню GAN.

У 2018 році було представлено StyleGAN, що дозволило генерувати вражаючі реалістичні портрети людей, а також розроблено велику кількість інших GAN-архітектур та моделей.

В останні роки було представлено численні вдосконалені версії GAN та нові застосування в різних галузях, включаючи медицину, сферу мистецтва, автономні автомобілі, генетику, фінанси та багато інших.

З інноваціями та вдосконаленням GAN моделей і їх застосуванням продовжуються, і це лише початок історії цього захоплюючого напрямку в

глибокому навчанні. GAN залишаються активною областю досліджень та розробок в сучасному машинному навчанні.

Висновки до розділу 1

У висновку можна зазначити, що історія розвитку генеративно-змагальних мереж (GAN) свідчить про вражаючі досягнення у сфері глибокого навчання та генерації даних. Від їхнього виникнення вони здобули величезну популярність завдяки своїй унікальній здатності генерувати реалістичні дані шляхом конкурентної взаємодії двох мереж.

Застосування GAN охоплює широкий спектр галузей, включаючи перетворення зображень, синтез даних, генерацію музики та багато інших. Вони виявилися надзвичайно корисними у вирішенні завдань, для яких потрібні генеративні рішення, та дали поштовх для розвитку нових методів у глибокому навчанні.

У сучасному світі важливість GAN зростає, оскільки вони допомагають вирішувати складні завдання в таких сферах, як комп'ютерний зір, медична діагностика, творчі індустрії та багато інших. Їхній внесок в розвиток штучного інтелекту та глибокого навчання робить їх ключовим інструментом для інновацій у різних галузях, а також цікавим об'єктом для дослідження.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ОБЛАСТІ ЗАДАЧІ СТРУКТУРНО-ПАРАМЕТРИЧНОГО СИНТЕЗУ ТА СУЧАСНИХ МОДЕЛЕЙ ГЕНЕРАТИВНО-ЗМАГАЛЬНИХ МЕРЕЖ

У цьому розділі будуть розглянуті компоненти за допомогою яких створено покращення і досягнута мета роботи.

2.1 Класична модель GAN

Генеративно-змагальна мережа (GAN) – це архітектура глибокого навчання. Вона вчить дві нейронні мережі конкурувати один з одним і генерувати більш реалістичні нові дані із заданого для навчання набору. Наприклад, можна створювати зображення або музику з відповідних вхідних даних. GAN називають змагальною, тому що вона навчає дві різні мережі та протиставляє їх одна одній. Одна з них генерує нові дані, а друга намагається передбачити, чи отримані вихідні дані є оригінальними. Інакше кажучи, мережа дискримінатор визначає, чи є згенеровані дані підробленими чи реальними. Генератор створює нові покращені версії підроблених даних доти, доки мережа дискримінатор не перестане відрізняти підробку від оригіналу.

Генератор є найважливішим блоком в архітектурі GAN, і розуміння його ролі та структури має важливе значення для розуміння процесу навчання GAN. Архітектура генератора складається з трьох компонентів: латентного простору, генератора та секції генерації зображення. Генератор отримує вектор з латентного простору та створює зв'язок між латентним простором і виходом. Потім ми створюємо нейронну мережу, яка перетворює вхід (латентний простір) на вихід (згенеровані дані наближені до реальних).

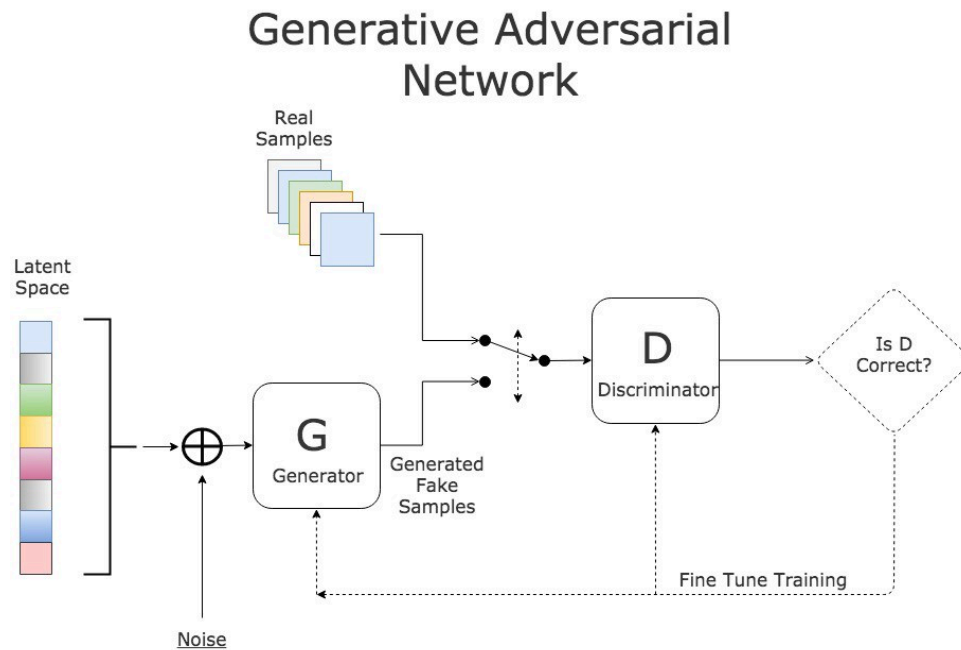


Рисунок 2.1 – Архітектура класичної GAN моделі [20]

Генератор в GAN отримує вхідні дані від вектора випадкового шуму та генерує вихідні дані, схожі на реальні дані, щоб створити нові зразки даних, наближені до реального розподілу даних. Отже, щоб розробити успішну модель GAN, важливо правильно визначити архітектуру генератора, що включає кілька кроків:

1. Вхідний рівень: Генератор приймає випадковий вектор шуму як вхідний сигнал, який зазвичай складається з гаусового або рівномірного розподілу.
2. Приховані шари: Генератор використовує серію прихованих шарів для перетворення вектора шуму на вихід, що нагадує справжні дані. Для реалізації цих прихованих рівнів використовуються різні архітектури нейронної мережі, включаючи повністю зв'язані шари, розгорткові шари або повторювані шари.
3. Вихідний рівень: останній рівень генератора створює вихідні дані, тоді як кількість нейронів у цьому шарі має відповідати розмірам реальних даних.
4. Функції активації: функція активації, яка використовується на вихідному

рівні, залежить від типу згенерованих даних. Наприклад, сигмоїдна функція активації зазвичай використовується для отримання вихідних значень від 0 до 1 для даних зображення.

5. Функція втрат. Бінарна кросентропія зазвичай використовується для вимірювання різниці між згенерованим результатом і реальними даними під час навчання.
6. Оптимізатор: генератор навчається за допомогою оптимізатора, такого як стохастичний градієнтний спуск (SGD) або Adam.
7. Гіперпараметри: для навчання генератора необхідно визначити кілька гіперпараметрів, наприклад швидкість навчання, розмір партії, кількість епох і розмір прихованих шарів.

У GAN дискримінатор – це нейронна мережа, яка відповідає за класифікацію даних, згенерованих генератором, як справжніх так і підроблених.

Дискримінатор в архітектурі GAN служить глибокою нейронною мережею, яка розрізняє реальні та підроблені зображення, генеруючи скалярне значення від 0 до 1, що вказує на ймовірність того, що вхідні дані є справжніми.

Дискримінатор є ключовим компонентом архітектури GAN, оскільки він оцінює автентичність як реальних, так і згенерованих зображень і поступово вчиться розрізняти їх, дозволяючи генератору генерувати нові, раніше недоступні дані з латентного простору. Генератор вчиться мінімізації логарифму втрат на виході дискримінатора для згенерованих зразків, прагнучи генерувати реалістичні зображення, мінімізуючи різницю між згенерованими та реальними навчальними даними.

Щоб визначити архітектуру дискримінатора в GAN, спочатку потрібно визначити вхідні та вихідні розміри дискримінатора. Вхідний розмір – це розмір зображення або даних, які дискримінатор збирається класифікувати, тоді як вихідний розмір зазвичай є бінарним виходом, що вказує на те, чи є вхід справжнім чи фальшивим. Після визначення вхідних і вихідних розмірів можна приступати до проєктування архітектури нейронної мережі дискримінатора. Дискримінатор зазвичай використовує шари згорткової нейронної мережі (CNN)

для вилучення ознак із вхідних даних. Ці ознаки потім подаються на повністю пов'язаний рівень (fully connected layer), який виводить двійкову класифікацію справжніх чи підроблених даних. Кількість шарів CNN і розмір фільтрів, що використовуються в цих шарах, є гіперпараметрами, які можна налаштувати на основі розміру та складності вхідних даних. Загалом, кількість шарів CNN може варіюватися від кількох до десятків залежно від складності набору даних.

Крім того, можуть бути додані шари пакетної нормалізації між шарами CNN для підвищення стабільності та продуктивності дискримінатора. Шари дропауту також можуть бути використані для запобігання перенавчання дискримінатора. Функція втрат дискримінатора в GAN зазвичай є двійковою крос-ентропійною втратою, яка вимірює різницю між прогнозованими та фактичними вихідними мітками.

Рис. 2.2 ілюструє приклад топології генератора та дискримінатора для задачі генерування зображень:

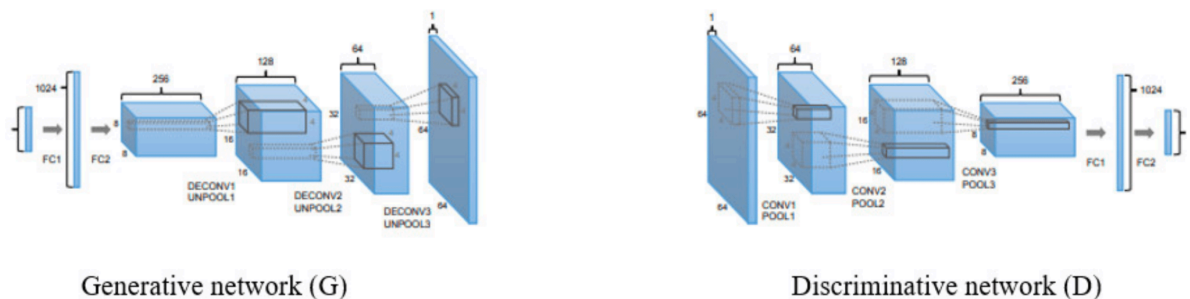


Рисунок 2.2 – Топологія генератора (G) та дискримінатора (D) моделі GAN [18]

2.2 Опис існуючих моделей GAN

За останні 8 років з моменту появи першої моделі GAN з'явилося багато нових архітектур, таких як: DCGAN, WGAN, WGAN-GP, SGAN, CGAN тощо.

Кожна з архітектур має свої переваги та недолік, що будуть наведені у цьому підрозділі.

2.2.1 DCGAN

DCGAN, або глибока генеративна мережа з декількома шарами (Deep Convolutional Generative Adversarial Network), є модифікацією звичайних генеративно-змагальних мереж (GAN), спеціально розробленою для генерації реалістичних зображень. Розглянемо основні аспекти DCGAN, порівняння зі звичайними GAN та їх недоліки.

Топологія DCGAN використовує розгорткові шари для генерації та згорткові шари для дискримінації зображень. Основні особливості архітектури включають генератор, який використовує розгорткові шари для генерації зображення з вектора шуму, та дискримінатор, який використовує згорткові шари для класифікації зображень як реальних чи згенерованих.

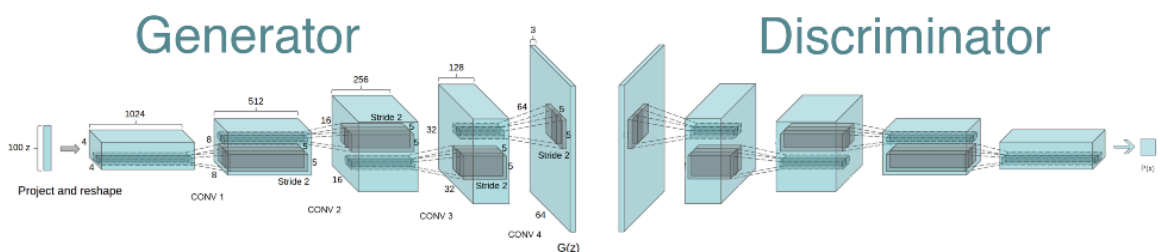


Рисунок 2.3 – Топологія DCGAN [19]

DCGAN має перевагу порівняно зі звичайними GAN, таку як здатність генерувати високоякісні зображення.

Недоліки DCGAN включають чутливість до параметрів, схильність до перенавчання та складність тренування.

2.2.2 WGAN

Wasserstein GAN (WGAN) використовує відстань Вассерштейна, яка має більш плавний градієнт. WGAN навчається незалежно від того, працює генератор чи ні. Діаграма нижче повторює подібний графік значення $D(X)$ як для GAN, так і для WGAN. Для GAN (червона лінія) вона заповнюється областями зі зменшуючими або зростаючими градієнтами. Для WGAN (блакитна лінія) градієнт скрізь плавніший і навчається краще, навіть якщо генератор не створює якісних зображень.

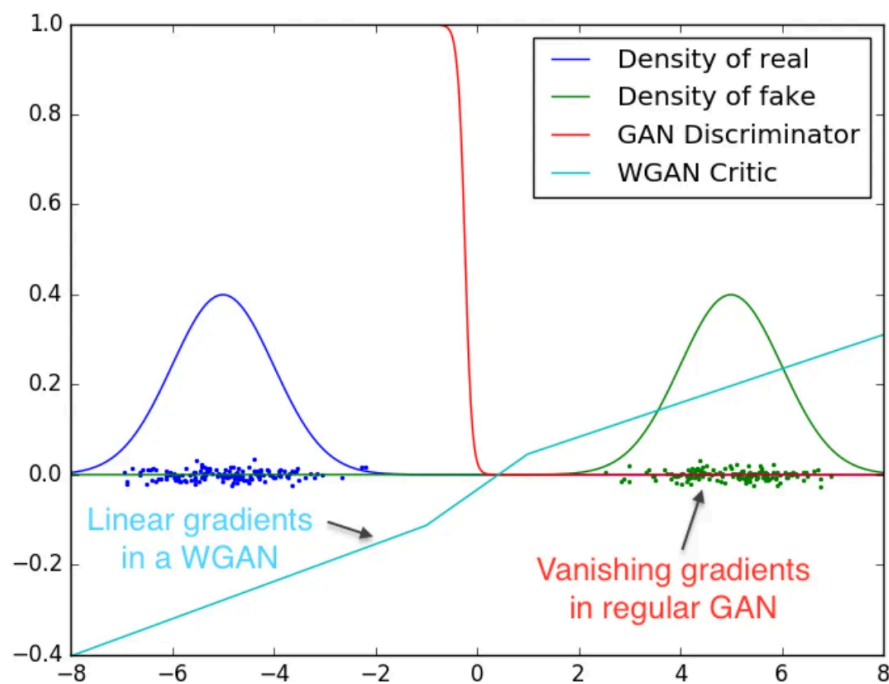


Рисунок 2.4 – Порівняльний графік градієнтів для GAN та WGAN [17]

Однак рівняння для відстані Вассерштейна є дуже складним. Використовуючи подвійність Канторовича-Рубінштейна, можна спростити розрахунок до

$$W(P_r, P_\theta) = \sup_{\|f\|_{\text{Lipschitz}} \leq 1} \left(\mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{z \sim P_\theta} [f(z)] \right),$$

де P_r, P_θ множина реальних даних та множина випадкових чисел відповідно, а $\sup$ є найменшою верхньою межею, f є 1-ліпшицевою функцією, що дотримується цього обмеження.

$$|f(x_1) - f(x_2)| \leq \|x_1 - x_2\|,$$

де x_1 та x_2 є реальними зображеннями. де $\|x_1 - x_2\|$ —це норма (відстань) між векторами x та y . Це означає, що величина зміни функції не перевищує величину зміни її вхідних аргументів, помножену на 1.

Отже, щоб обчислити відстань Вассерштейна, нам просто потрібно знайти 1-ліпшицеву функцію. Як і інші проблеми глибокого навчання, можна побудувати глибоку мережу. Ця мережа дуже схожа на дискримінатор D , тільки без сигмоїдної функції та виводить скалярну оцінку, а не ймовірність. Цей бал можна інтерпретувати як реальність вхідних зображень. У навчанні з підкріпленням ми називаємо це функцією значення, яка вимірює, наскільки добрим є стан (вхід). Назва “дискримінатор” змінюється на “критик”, щоб відобразити його нову роль. Конструкція мережі майже така сама, за винятком того, що критик немає вихідної сигмоїдної функції.

Однак важливої речі все ще не вистачає. Функція f повинна бути 1-ліпшицевою, щоб гарантувати коректність обчислення відстані Вассерштейна. З метою посилення цього обмеження, WGAN використовує простий механізм відсікання. Максимальне значення ваги в f , тобто ваги дискримінатора, обмежується за допомогою відсікання, що встановлює максимальну межу, контрольовану гіперпараметром c .

$$w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w),$$

$$w \leftarrow \text{clip}(w, -c, c).$$

Топологія моделей дискримінатора та генератора для WGAN залежить від типу даних, що використовується при навчанні. Наприклад для зображень можна використовувати згорткові неймережі ідентичні до топології DCGAN.

2.2.3 WGAN-GP

WGAN-GP використовує градієнтний штраф замість обмеження ваги, щоб забезпечити обмеження Ліпшица. Диференційована функція f є 1-ліпшицевою тоді і тільки тоді, коли вона всюди має градієнти з нормою не більше 1. Зокрема, Додаток А в документі WGAN-GP це підтверджує [12]. Точки, інтерпольовані між реальними та згенерованими даними, повинні мати градієнтну норму 1 для f . Тож замість застосування відсікання WGAN-GP штрафує модель, якщо норма градієнта відхиляється від цільового значення норми 1.

$$L = \underbrace{\mathbb{E}_{\tilde{x} \sim \mathbb{P}_g} [D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_r} [D(x)]}_{\text{Original critic loss}} + \lambda \underbrace{\mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} [(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Our gradient penalty}}.$$

де $\hat{x}$ вибірка з $x^{\sim}$ і $x^{\sim}$ з t рівномірно розподілені між 0 і 1,

$$\hat{x} = tx^{\sim} + (1 - t)x \quad t \in [0; 1].$$

λ задана і дорівнює 10. Точка x , яка використовується для обчислення норми градієнта, є будь-якою точкою, взятою між $\mathbb{P}_g$ і $\mathbb{P}_r$.

Пакетна нормалізація не застосовується для критика (дискримінатора). Пакетна нормалізація створює кореляції між зразками в одного батчу. Це

впливає на ефективність градієнтного штрафу, що підтверджено експериментами.

Задумом чи ні, деякі нові функції втрат додають градієнтний штраф до функції втрат. Деякі ґрунтуються виключно на емпіричних спостереженнях, що моделі поведуться неправильно, коли градієнт збільшується. Однак градієнтне покарання ускладнює обчислення, що може бути небажаним, але воно створює зображення вищої якості.

Топологія моделей дискримінатора та генератора для WGAN залежить від типу даних, що використовується при навчанні. Наприклад для зображень можна використовувати згорткові нейромережі ідентичні до топології DCGAN.

2.2.4 SGAN

Розглянемо альтернативний тренувальний процес, називається SGAN, у якому кілька змагальних «локальних» пар мереж тренуються незалежно, щоб проти них можна було навчити «глобальну» контрольну пару мереж. Мета полягає в тому, щоб тренувати глобальну пару з відповідним суперником ансамблю для кращих виступів у термінах покриття режиму. Цей підхід спрямований на підвищення шанси на те, що навчання не припиниться для глобальної пари, запобігаючи тому, щоб вони потрапили в пастку незадовільного локального мінімуму або зіткнулися з коливаннями, які часто спостерігаються на практиці. до гарантуючи останнє, глобальна пара ніколи не впливає на локальну одну.

Таким чином, правила навчання SGAN такі: глобальний генератор і дискримінатор навчаються за допомогою локальних дискримінаторів та генераторів відповідно, тоді як локальні мережі тренуються з їх фіксованим локальним противником.

Експериментальні результати як для тестів, так і для реальних проблем демонструють, що цей підхід перевершує стандартне навчання з точки зору кращого пом'якшення режиму згортання, стабільності під час конвергенції та, як не дивно, збільшує також швидкість конвергенції.

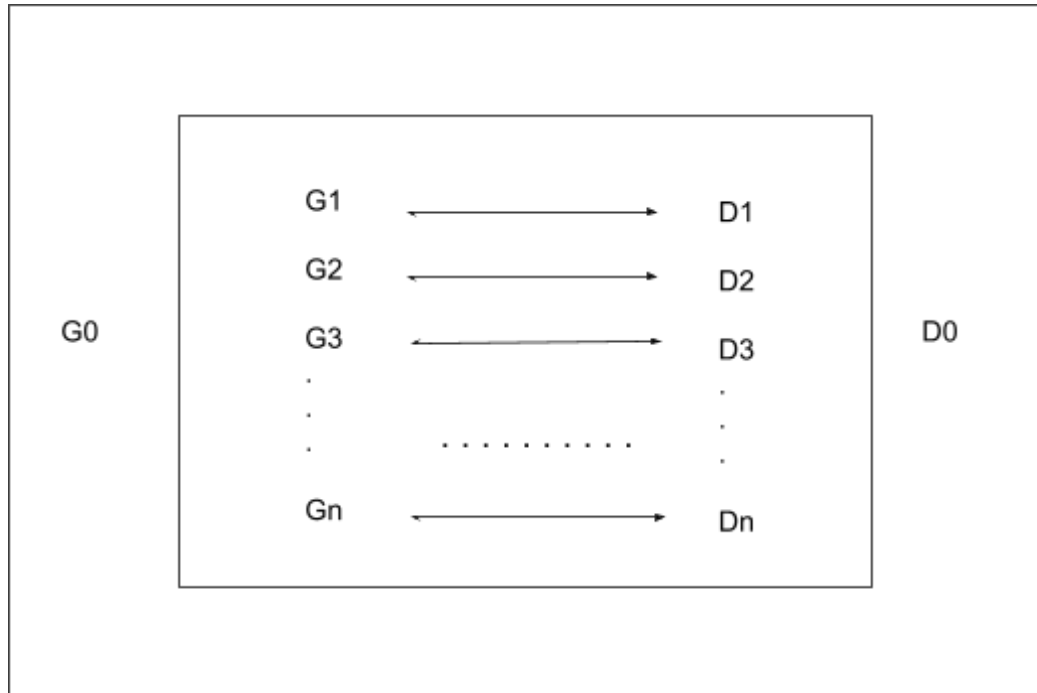


Рисунок 2.5 – Концептуальна ілюстрація SGAN. Є $n + 1$ пари, з яких пара (G_0, D_0) не тренується безпосередньо. D_0 тренується з G_i , $i=1, \dots, n$ і G_0 навчається з D_i , $i=1, \dots, n$, як показано пунктирними прямокутниками

N -пар складається з $G = \{G_1, \dots, G_N\}$ – генераторів і $D = \{D_1, \dots, D_N\}$ – дискримінаторів, які навчаються окремо. Глобальна пара (D_0, G_0) тренується за допомогою локальних пар.

Локальні пари тренуються самостійно в стандартному підході для GAN. Дискримінатор D_0 із глобальної пари тренується з кожним уже навченим генератором G_i з локальної пари, де $i = \{1, \dots, N\}$. Як D_0 , генератор G_0 із глобальної пари також тренується з кожним дискримінатором D_i з локальної пари, АЛЕ замість використання справжнього D_i кожен D_i копіюється в іншу

мережу D_i^{msg} , яка називається «дискримінатором месенджером», а G_0 тренується з D_i^{msg} . На кожній ітерації D_i^{msg} створюється повторно для навчання D_0 . Цей алгоритм дуже добре піддається розгалуженню

Топологія моделей дискримінатора та генератора для SGAN залежить від типу даних, що використовується при навчанні. Наприклад для зображень можна використовувати згорткові нейромережі ідентичні до топології DCGAN.

2.2.5 CGAN

Conditional GAN (CGAN) бере інформацію про мітки і зразки і дає генератору додаткову інформацію для створення зображень. На рис. 2.6 нижче і генератор, і дискримінатор приймають мітки як входні дані. Це створює кращі зображення, але вимагає маркування навчальних зразків.

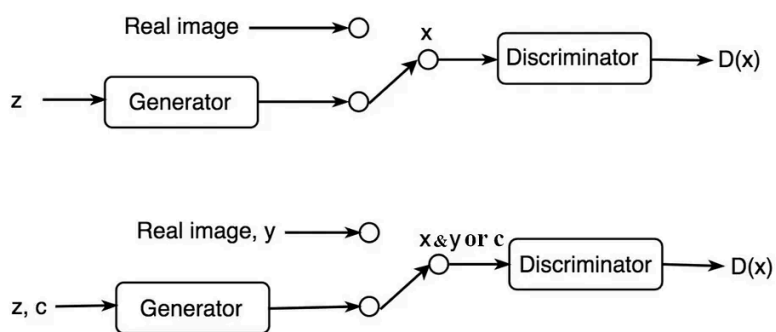


Рисунок 2.6 – Схема роботи генератора і дискримінатора в CGAN [17]

Топологія моделей дискримінатора та генератора для CGAN залежить від типу даних, що використовується при навчанні. Наприклад для зображень можна використовувати згорткові нейромережі ідентичні до топології DCGAN.

2.2.6 Складений або прогресивний GAN

В іншому дизайні GAN прогресивний GAN підвищує або зменшує роздільну здатність зображень на 2 у кожному підшарі. На етапі 1 навчаємо генератор створювати зображення 4×4 . Після завершення навчання додаємо рівень підвищення дискретизації, щоб подвоїти вихідну роздільну здатність до 8×8 . Зробивши це 9 разів, зрештою буде створено зображення розміром 1024×1024 .

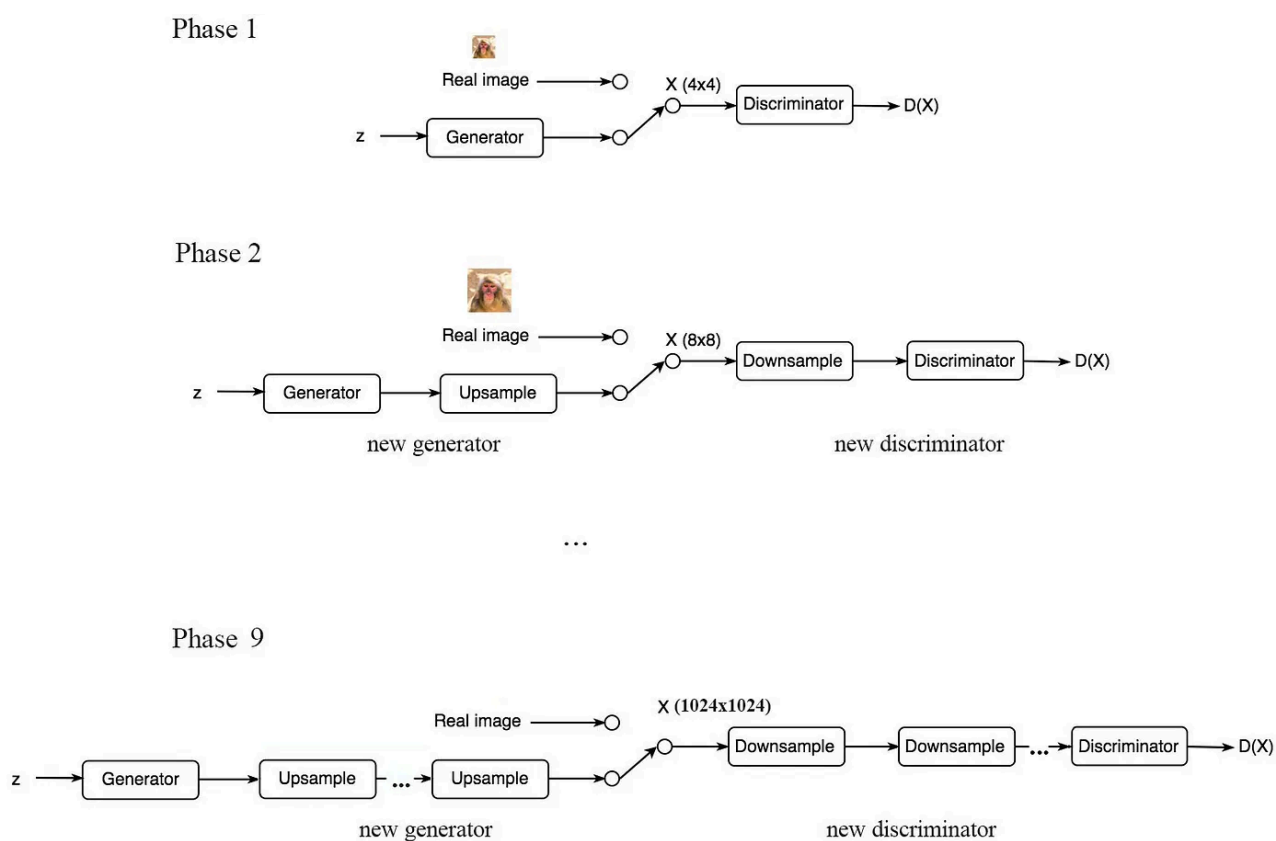


Рисунок 2.7 – Схема роботи progressive GAN [17]

2.3 Алгоритм навчання GAN

Генеративно-змагальна мережа складається з двох нейронних мереж –

генеруючої та дискримінуючої. Обидві мережі тренуються у змагальній грі, де одна намагається генерувати нові дані, а інша – вгадати, чи вони є підробленими чи реальними.

Генеруюча мережа намагається максимально збільшити ймовірність помилки мережі, що дискримінує, а та у свою чергу намагається звести шанс до мінімуму. Повторюючи навчання щоразу, генеруюча і дискримінуюча мережі розвиваються і постійно протистоять одна одній, доки досягнуть стану рівноваги (рівноваги Неша). У цьому стані мережа, що дискримінує, більше не може розпізнавати синтезовані дані. На цьому тренувальний процес закінчено.

Генератор приймає випадковий вектор z , що розподілений за нормальним законом, як вхідні дані та генерує вихідне зображення $G(z)$. Дискримінатор приймає або згенероване зображення $G(z)$, або реальне зображення x як вхідні дані та генерує вихідні дані $D(x)$ або $D(G(z))$.

Під час навчання дискримінатора ваги та зсуви генератора заморожені. Дискримінатор штрафується, коли він передбачає реальне зображення як підроблене або підроблене зображення як справжнє. Вагові коефіцієнти та зміщення дискримінатора оновлюються через зворотне поширення від втрати дискримінатора.

Так само під час навчання генератора ваги та зміщення дискримінатора заморожуються. Мета генератора – обдурити дискримінатор. Отже, генератор штрафується, якщо дискримінатор може визначити, що зображення, створене генератором, є підробленим зображенням. У цьому випадку втрата повертається через мережу генератора, щоб оновити її ваги та зміщення.

Концептуально GAN можна розглядати як мінімаксну гру між моделлю генератора та моделлю дискримінатора. Обидві моделі навчаються одночасно, де одна модель намагається мінімізувати втрати, а інша намагається максимізувати втрати. Нижче наведена формула мінімаксної гри генератора та дискримінатора:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Дискримінатор повертає значення $D(x)$, що вказує ймовірність того, що x є реальним зображенням. Наша мета—максимізувати ймовірність правильно розпізнати реальні зображення, а відтворені вважати штучно створеними. Іншими словами, дискримінатор прагне до максимальної ймовірності коректного класифікування спостережуваних даних. Для вимірювання втрат використовується бінарна крос-ентропія, яка є стандартним підходом у багатьох методах глибокого навчання.

$$\max_D V_D = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] - \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

З боку генератора, його цільова функція хоче, щоб модель генерувала зображення з максимально можливим значенням $D(G(z))$, щоб обдурити дискримінатор.

$$\min_G V_G = - \mathbb{E}_{z \sim p_z(z)} [\log D(G(z))]$$

Після визначення обох цільових функцій, вони об'єднуються і оптимізуються разом за допомогою алгоритму градієнтного спуску. Спочатку фіксуємо параметри генератора та проводимо ітерацію навчання дискримінатора, використовуючи як реальні, так і згенеровані зображення. Після цього переключаємося, оптимізуємо параметри дискримінатора та навчаємо генератор протягом ще одного кроку. Процес навчання проводиться по черзі для обох мереж до тих пір, поки генератор не створить зображення задовільної якості. Нижче наведено огляд потоку даних і градієнтів, які використовуються для зворотного поширення.

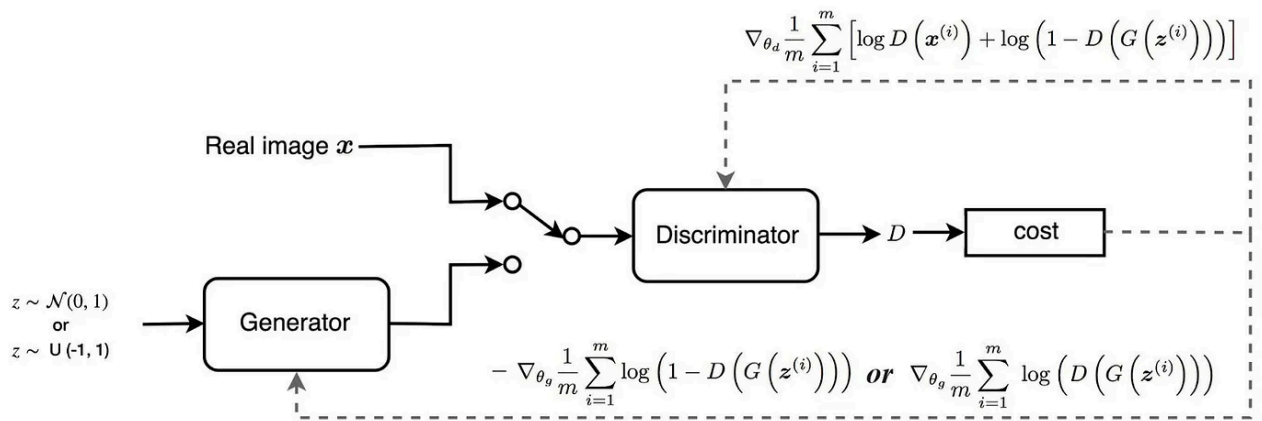


Рисунок 2.8 – Схема навчання GAN [17]

2.4 Проблеми що виникають при навчанні GAN

2.4.1 Згортання режиму (Mode Collapse)

Розповсюдження реальних даних є мультирежимним (складається із розповсюджень декількох режимів даних). Наприклад, у MNIST існує 10 основних режимів від цифри «0» до цифри «9». Наведені нижче зразки генеруються двома різними GAN. Верхній рядок створює всі 10 режимів, тоді як другий рядок створює лише один режим (цифра «6»). Ця проблема називається згортанням режиму, коли генерується лише кілька режимів даних.

Є кілька причин, чому згортання режиму може статися в GAN. Однією з причин є катастрофічне забування, коли знання, засвоєні в попередньому завданні, руйнуються під час навчання в поточному завданні. Іншою причиною колапсу режиму є перенавчання дискримінатора, що призводить до зникнення втрат генератора.

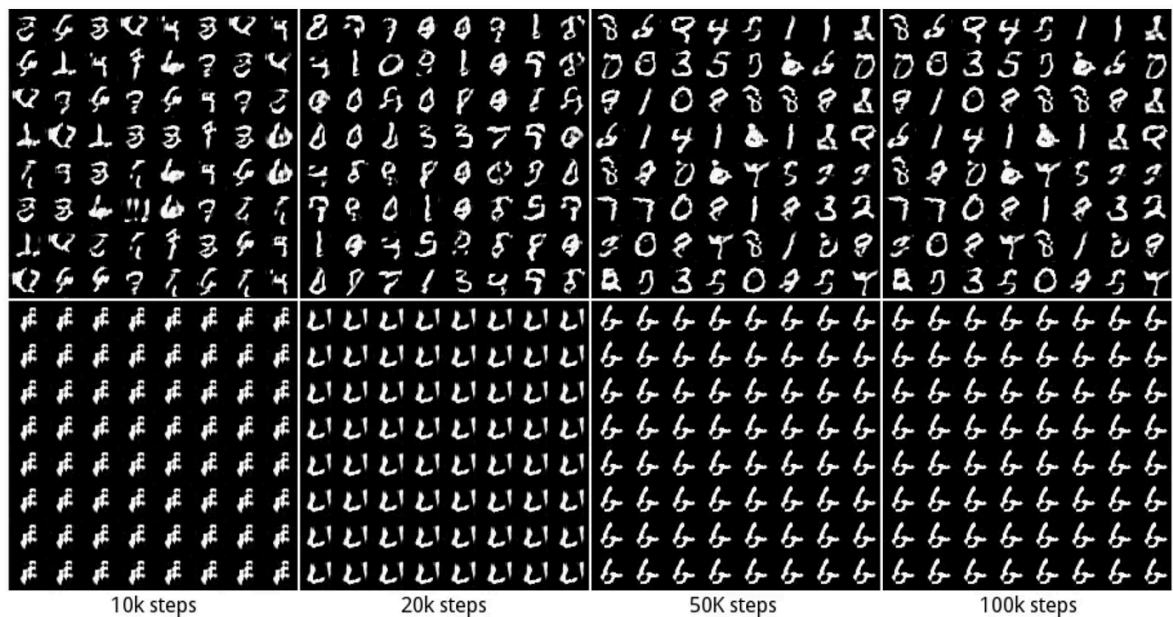


Рисунок 2.9 – Приклад згортання режиму на різних ітераціях навчання моделі GAN [16]

2.4.1.1 Катастрофічне забування

Катастрофічне забування відноситься до явища, за якого модель, навчена виконанню конкретного завдання, забуває знання, які вона отримала під час вивчення нового завдання. Давайте розглянемо GAN, що складається з генератора G і дискримінатора D . Під час t -го кроку навчання; генератор створює вибірки з розподілом p_G , який наближає справжній розподіл даних p_R . На кожному кроці навчання дискримінатор навчається розрізняти реальні дані p_R і згенеровані вибірки p_G . Однак, оскільки p_G змінюється з кожним кроком, дискримінатор повинен адаптуватися до нових завдань класифікації [?].

Проаналізуємо катастрофічне забування, навчивши GAN на простих синтетичних 2D-даних. Набір даних складається з шести розподілів Гауса, розташованих по колу, як показано на рис. 2.10. Сині точки даних

представляють реальні дані, тоді як помаранчеві точки даних представляють згенеровані зразки. Завдання моделі – відтворити розподіл синіх точок даних. Стрілки вказують напрямок зменшення втрат генератора ($-d(\text{Loss}_G) / dx$). Коли втрати генератора зменшуються, вихід дискримінатора $D(x)$ збільшується. Таким чином, можна інтерпретувати стрілки як вказівку напрямку збільшення впевненості дискримінатора в ідентифікації справжніх точок даних. Варто зауважити, що оцінка дискримінатора збільшується, коли рух відбувається вздовж стрілок пропорційно довжині стрілки.

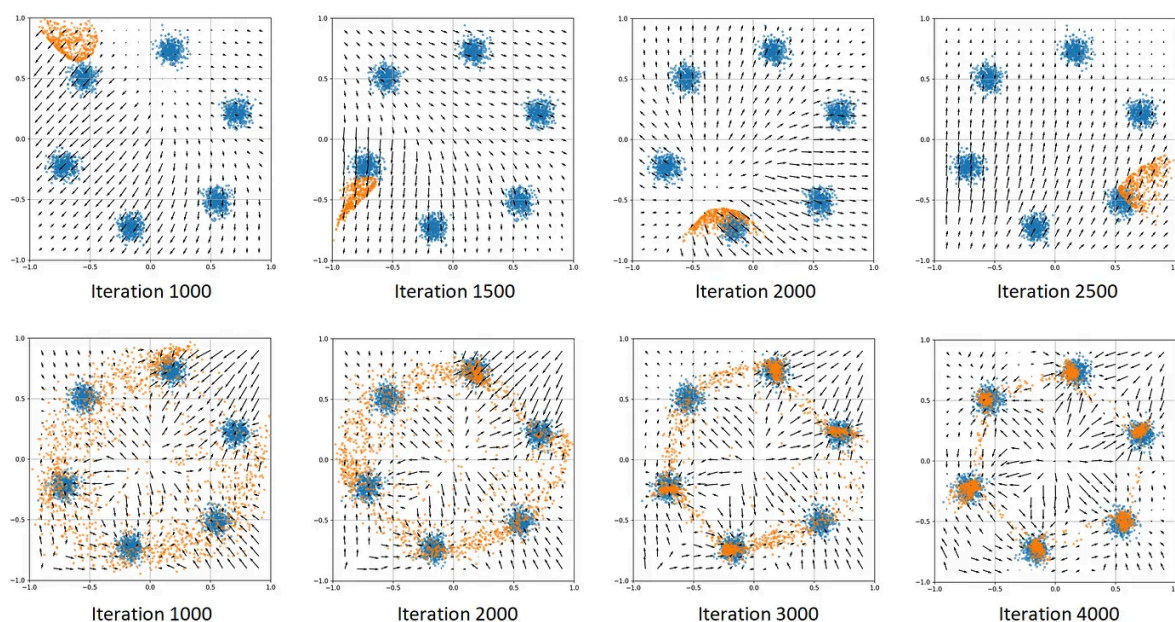


Рисунок 2.10 – Навчання GAN на синтетичних даних [21]

Дивлячись на поведінку дискримінатора, зображеного у верхній частині рис. 2.10, стає очевидним, що він дає вищі оцінки точкам, які знаходяться далі від згенерованих зразків, і не звертає уваги на реальні точки даних. Це відбувається тому, що згенеровані зразки розташовані в невеликій області, і, отже, стає легко розрізнити справжній і згенерований розподіли, просто надавши низькі оцінки створеним зразкам і високі оцінки іншим регіонам. Крім того, через монотонну природу градієнтів навколо згенерованих точок даних

вони не можуть поширюватися на різні області. Інше спостереження полягає в тому, що напрямок градієнтів може бути протилежним самому собі на різних ітераціях і залежати лише від згенерованих зразків. Це означає, що модель забуває знання, отримані на попередніх кроках, що призводить до катастрофічного забуття. Поведінка навчання без згортання режиму продемонстрована внизу рис. 2.10. Помітна закономірність полягає в тому, що при конвергенції GAN реальні точки даних є локальними максимумами дискримінатора.

Щоб визначити, чи відбувається катастрофічне забування в реальних багатовимірних даних, візуалізуємо поверхню дискримінатора, який був навчений на наборі даних MNIST. Оскільки розподіл зображень у наборі даних є багатовимірним, неможливо візуалізувати його у 2D. Однак можна провести випадкову лінію через зображення та спостерігати, як оцінка дискримінатора змінюється вздовж лінії. Візуалізуємо значення $f(x) = D(x + k*u)$, де u – випадковий спрямований вектор, а k – коефіцієнт зсуву. На рис. 2.11 зображено графік функції f . Більше того, це ілюструє, що генератор видає подібні результати, що вказує на згортання режиму, а оцінки дискримінатора для тих самих зображень змінюються між кроками навчання. Таким чином, GAN демонструють ті ж катастрофічні тенденції забування, що й під час навчання на симетричних 2D-наборах даних.

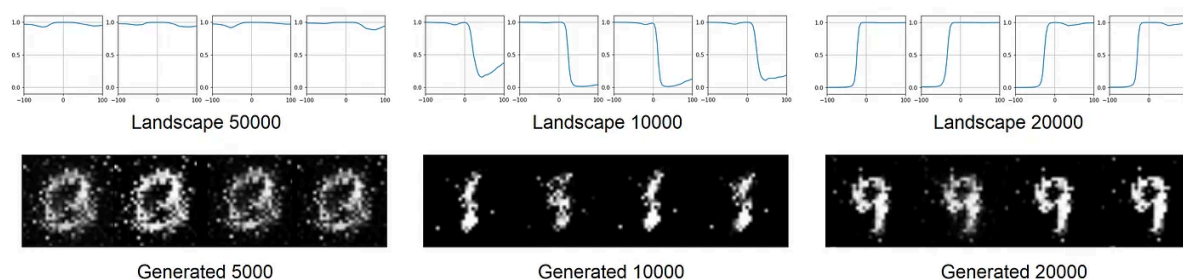


Рисунок 2.11 – Навчання GAN на наборі даних MNIST зі згортанням режиму

2.4.1.2 Перенавчання дискримінатора

Нижче застосовано метод візуалізації до GAN, який не постраждав від згортання режиму та продемонстровано різноманітність та точність результатів, як показано на рис. 2.12. Під час навчання GAN функція $f(k)$ постійно демонструє локальний максимум при $k=0$ для кожної реальної точки даних. Це означає, що дискримінатор піддається впливу локальних максимумів на реальних точках даних. Коли згенерована вибірка потрапляє в зону привабливості реальної точки даних, застосування оновлень градієнта безпосередньо до згенерованої вибірки перемістить її до реальної точки даних. Існування різних атракторів (локальних максимумів) у різних областях простору даних переміщує різні згенеровані вибірки в різних напрямках, розподіляючи їх по всьому простору та ефективно мінімізуючи колапс режиму.

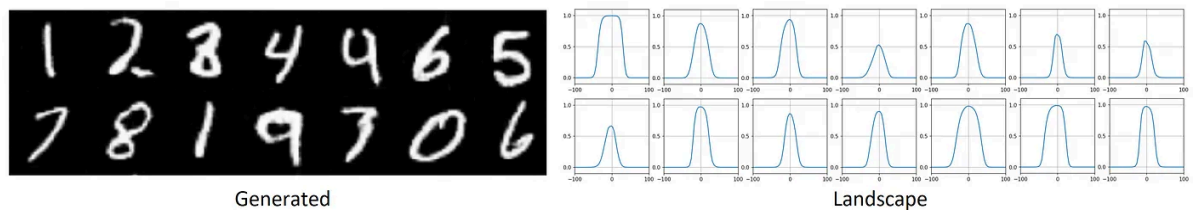


Рисунок 2.12 – Навчання GAN на наборі даних MNIST із хорошою конвергенцією [21]

Максимуми, зображені на рис. 2.13, значно гостріші, ніж на рис. 2.12, через переобладнання дискримінатора на реальних точках даних, що призводить до наближення оцінок сусідніх точок даних до нуля. Це призводить до появи численних плоских областей, де градієнти дискримінатора до точок даних у цих областях є зникаюче малими. У результаті згенеровані зразки, що знаходяться в плоскій області, не можуть рухатися до реальних точок даних через зникаючий градієнт. Таким чином, реальні точки даних, зображені на рис. 2.13, мають

обмежені басейни тяжіння і не можуть ефективно розподіляти згенеровані зразки в просторі. Отже, різноманітність згенерованих вибірок зменшується, що призводить до колапсу режиму. Щоб запобігти цьому, локальні максимуми повинні мати широку форму, що вказує на великий басейн тяжіння, який тягне згенеровані зразки в різні боки.

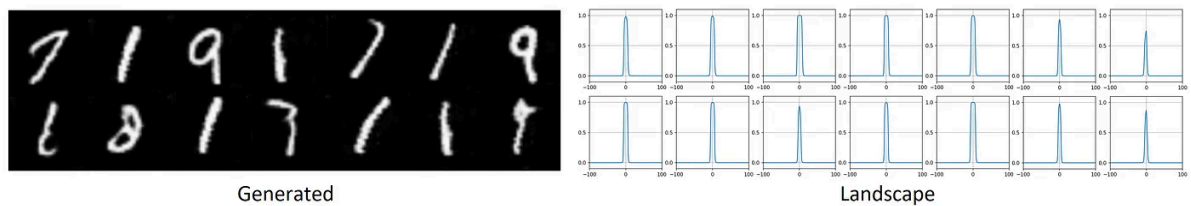


Рисунок 2.13 – Навчання GAN на наборі даних MNIST із домінуванням дискримінатора [21]

2.4.2 Рівновага Неша

GAN базується на некооперативній грі з нульовою сумою. Коротше кажучи, якщо один виграє, інший програє. Гру з нульовою сумою також називають мінімаксом. Ваш опонент хоче максимізувати свої дії, а ваші дії – мінімізувати їх. У теорії ігор модель GAN сходиться, коли дискримінатор і генератор досягають рівноваги Неша. Це оптимальна точка для мінімаксового рівняння нижче. Рівновага Неша відноситься до сценарію, за якого гравці не мають мотивації відхилятися від своєї початкової стратегії. Розглянемо двох гравців А і В, які контролюють значення x і y відповідно. Гравець А хоче максимізувати значення xu , тоді як В хоче його мінімізувати.

$$\min_B \max_A V(D, G) = xu$$

Рівновага за Нешем дорівнює $x=y=0$. Це стан, коли зміна думки одного гравця не покращить результат. Розглянемо, чи можливо легко знайти рівновагу Неша за допомогою градієнтного спуску. Оновлюємо параметри x і y на основі градієнта функції значення V .

$$\Delta x = \alpha \frac{\partial(xy)}{\partial(x)}$$

$$\Delta y = -\alpha \frac{\partial(xy)}{\partial(y)}$$

де α – швидкість навчання. З графіку x , y і xy проти навчальних ітерацій, стає зрозуміло, що розв'язок не збігається.

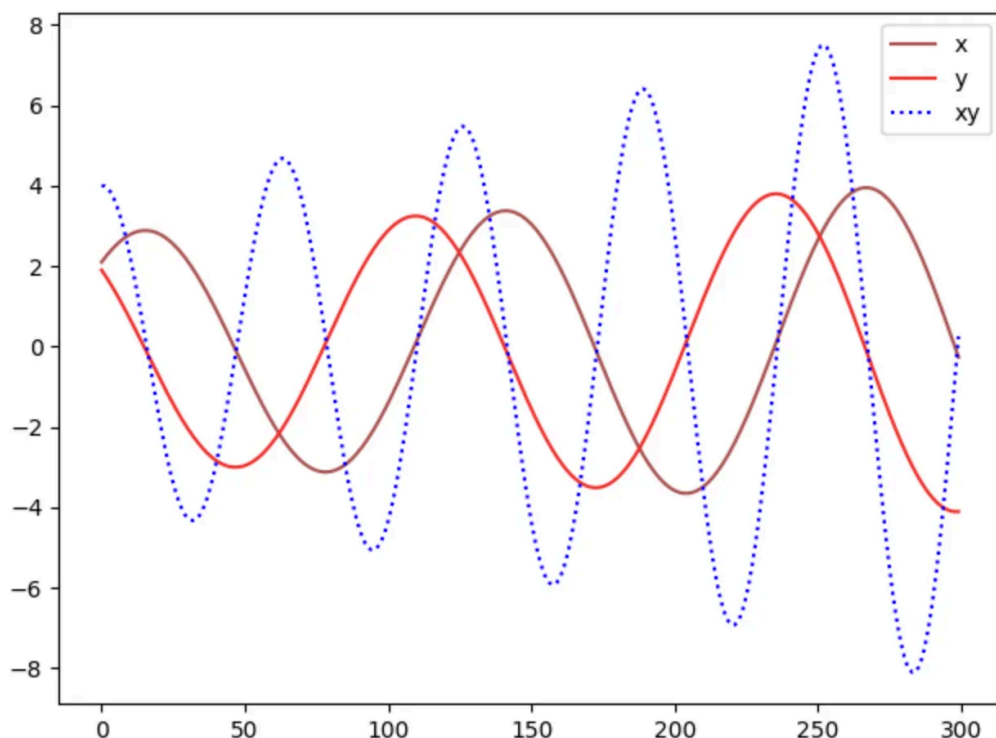


Рисунок 2.14 – Графік x , y і xy проти навчальних ітерацій [16]

Якщо збільшити learning rate або навчати модель довше, стає видно, що параметри x , y нестабільні з великими коливаннями.

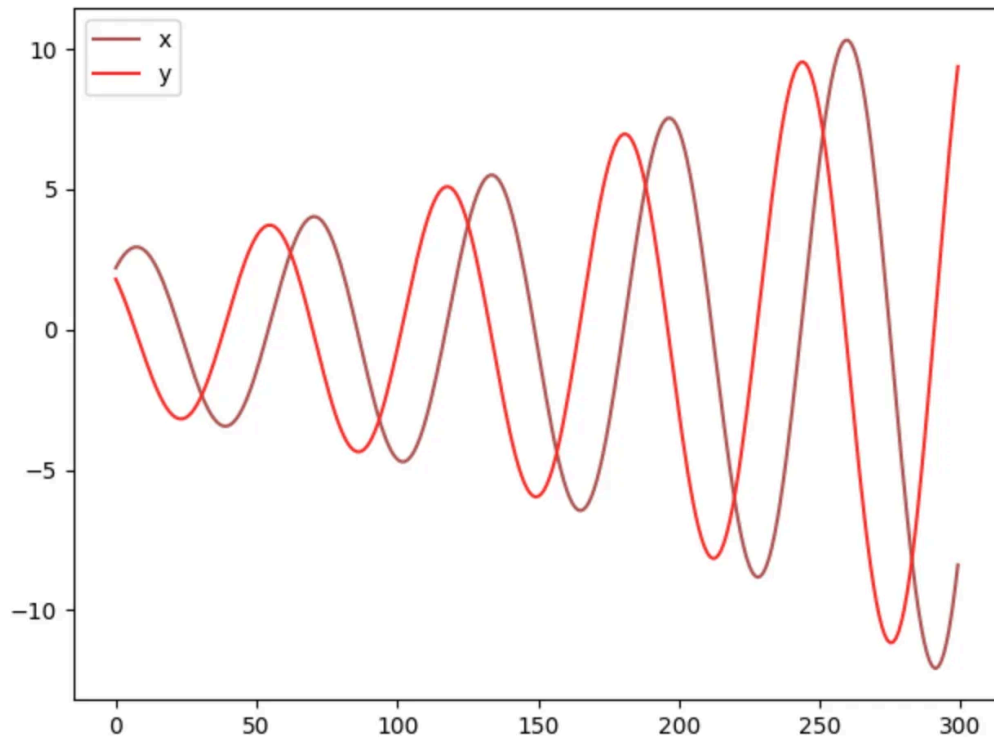


Рисунок 2.15 – Графік функції втрат генератора та дискримінатора при збільшенні швидкості навчання [16].

Це демонструє, що деякі функції втрат не збігаються з градієнтним спуском.

2.4.3 Зникаючий градієнт

Коли дискримінатор оптимальний, цільова функція для генератора:

$$\min_G V(D^*, G) = 2D_{JS}(p_r \| p_g) - 2 \log 2$$

Що відбувається з градієнтом JS-дивергенції, коли розподіл даних q зображень генератора не збігається з основною істинністю p для реальних

зображень. Нижче розглянуто приклад, у якому p і q є розподілом Гауса, а середнє значення p дорівнює нулю. Розглянемо q різними засобами для дослідження градієнта $JS(p, q)$.

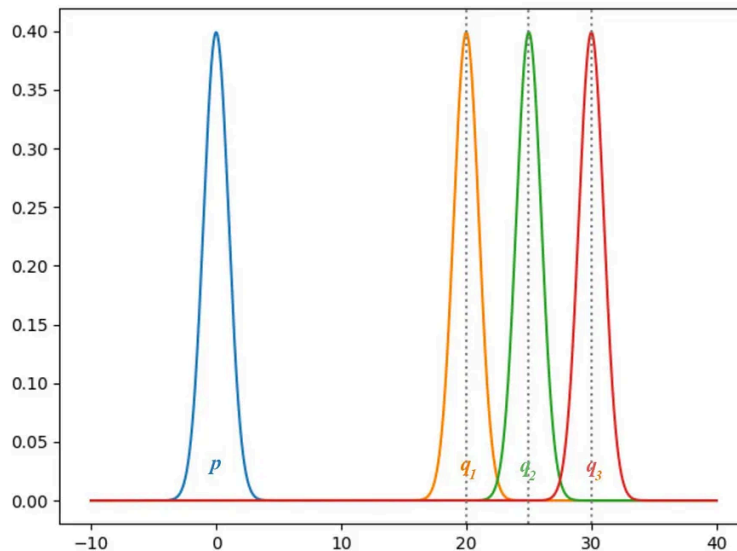


Рисунок 2.16 – Графік розподілів зображень при різних градієнтах. p –реальні зображення, q_1, q_2, q_3 –згенеровані [16]

Як показано нижче, градієнт для JS-розбіжності дорівнює нулю від q_1 до q_3 . Генератор GAN навчатиметься дуже повільно, аж до нуля, коли вартість насичена в цих регіонах. Зокрема, на початку навчання p і q дуже відрізняються, і генератор навчається дуже повільно.

2.5 Метрики що застосовуються до GAN

У GAN цільова функція для генератора та дискримінатора зазвичай вимірює, наскільки добре вони працюють відносно опонента. Наприклад, вимірюємо, наскільки добре генератор обманює дискримінатор. Це не є

хорошим показником для вимірювання якості зображення або його різноманітності. У цьому підрозділі розглянуто Inception Score, Frechet Inception Distance at F1 score.

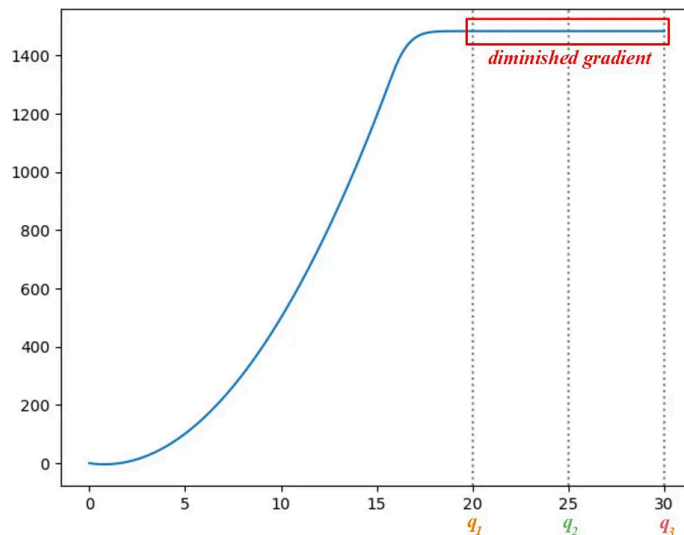


Рисунок 2.17—графік JS-розбіжності $JS(p, q)$ між p і q із середнім значенням q в діапазоні від 0 до 30 [16]

2.5.1 Inception Score (IS)

Inception Score (IS) – це метрика ефективності, яка використовується для оцінки якості згенерованих зображень або синтетичних зображень, створених мережами GAN. Він вимірює, наскільки реалістичними та різноманітними є вихідні зображення.

IS використовує два критерії для вимірювання ефективності GAN:

- якість створених зображень і
- їх різноманітність.

Ентропію можна розглядати як випадковість. Якщо значення випадкової величини x є високопередбачуваним, воно має низьку ентропію. Навпаки, якщо він дуже непередбачуваний, ентропія висока. Наприклад, на рис. нижче маємо два розподіли ймовірностей $p(x)$. p_2 має вищу ентропію, ніж p_1 , тому що p_2 має більш рівномірний розподіл і, отже, менш передбачувано щодо x .

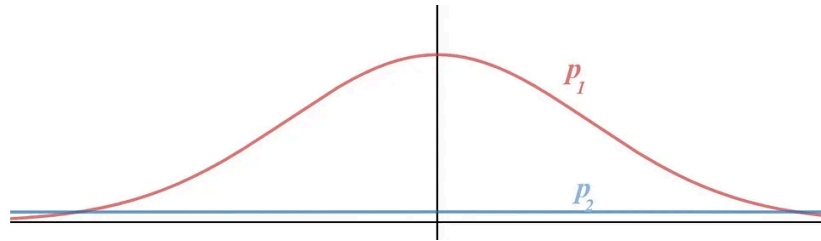


Рисунок 2.18—Схема розподілу двох випадкових величин p_1 та p_2 [14]

У GAN ми прагнемо, щоб умовна ймовірність $P(y|x)$ була високопередбачуваною (низька ентропія). тобто, маючи зображення, ми повинні легко знати тип об'єкта. Отже, ми використовуємо Inception мережу для класифікації згенерованих зображень і прогнозування $P(y|x)$ – де y – мітка, а x – згенеровані дані. Це відображає якість зображень. Далі потрібно виміряти різноманітність зображень. $P(y)$ – це гранична ймовірність, яка обчислюється як:

$$\int_z p(y|x = G(z))dz ,$$

де $G(z)$ це згенероване зображення, z —випадковий вектор, розподілений за нормальним законом.

Якщо створені зображення різноманітні, розподіл даних для y має бути рівномірним (висока ентропія). Щоб поєднати ці два критерії, обчислюється їх KL-розбіжність і використовується у рівнянні нижче для обчислення IS.

$$\text{IS}(G) = \exp \left(\mathbb{E}_{\mathbf{x} \sim p_a} D_{KL}(p(y|\mathbf{x}) \parallel p(y)) \right).$$

Одним із недоліків IS є те, що він може спотворити продуктивність, якщо генерує лише одне зображення на клас. $p(y)$ буде однорідним, навіть якщо різноманітність низька.

2.5.2 Fréchet Inception Distance (FID)

Відстань Фреше (Fréchet Inception Distance або FID) є метрикою якості для оцінки генеративних моделей, зокрема, для вимірювання якості зображень, створених мережами (GAN). Вона оцінює схожість між реальними та згенерованими зображеннями в просторі ознак.

У FID використовується мережа Inception для видобутку ознак із проміжного рівня. Після цього моделюється розподіл цих ознак, використовуючи багатовимірний розподіл Гауса з параметрами середнього значення μ і коваріації Σ . Розрахунок FID між реальними зображеннями x і згенерованими зображеннями g визначається як квадрат норми різниці середнього значення для x та g в сумі з Trace (Tr), що є сумою діагональних елементів:

$$\text{FID}(x, g) = \|\mu_x - \mu_g\|_2^2 + \text{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{\frac{1}{2}}),$$

де Tr підсумовує всі діагональні елементи.

FID чутливий до згортання режиму. Як показано нижче, відстань збільшується із змодельованими відсутніми режимами.

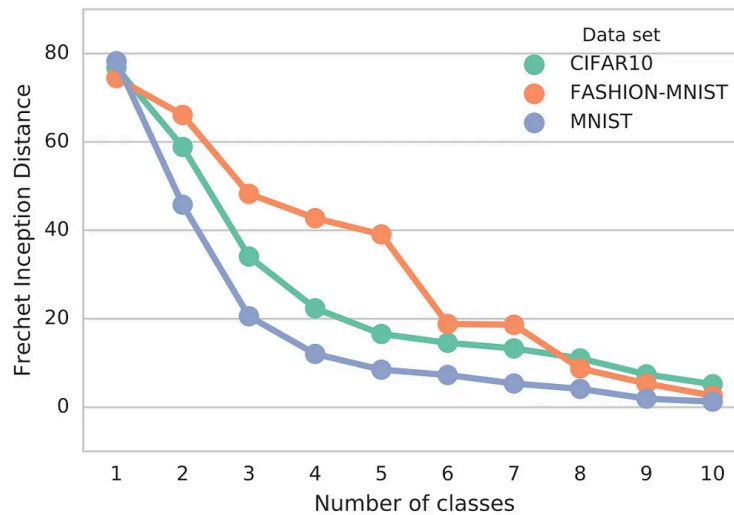


Рисунок 2.19—Графік FID від кількості класів [14]

FID виявляється більш стійким до шуму порівняно з IS. У випадку, коли модель генерує лише одне зображення для кожного класу, відстань буде великою. Отже, FID є більш точним індикатором різноманітності зображень. Важливо відзначити, що FID має високий зсув, але низьку дисперсію. При обчисленні FID між тренувальним та тестовим наборами даних можна очікувати, що FID буде дорівнювати нулю, оскільки обидва є реальними зображеннями. Однак проведення тестів з різними тренувальними наборами не видає жодного значення FID, відмінного від нуля.

Крім того, як FID, так і IS засновані на виділенні ознак (наявність або відсутність ознак).

2.5.3 Точність, повнота та оцінка F1

Якщо згенеровані зображення в середньому подібні до реальних, точність висока. Значення високої повноти вказує на те, що генератор може створювати будь-який зразок, який міститься в тренувальному наборі даних. Оцінка F1

представляє собою гармонічний середній показник між точністю та повнотою. Метрика F1-score використовується для оцінки якості бінарної класифікації, і вона є гармонічним середнім між точністю (precision) і повнотою (recall). У випадку GAN, де генератор створює зображення, його можна розглядати як бінарний класифікатор, який вирішує, чи зображення є "реалістичним" чи "синтетичним"

У дослідницькій роботі Google Brain «Чи GAN створені рівними» [7] створюється експеримент із набором даних трикутників для вимірювання точності та повноти різних моделей GAN. Цей набір даних може вимірювати продуктивність різних моделей GAN. Його можна використовувати для вимірювання якості різних функцій витрат.

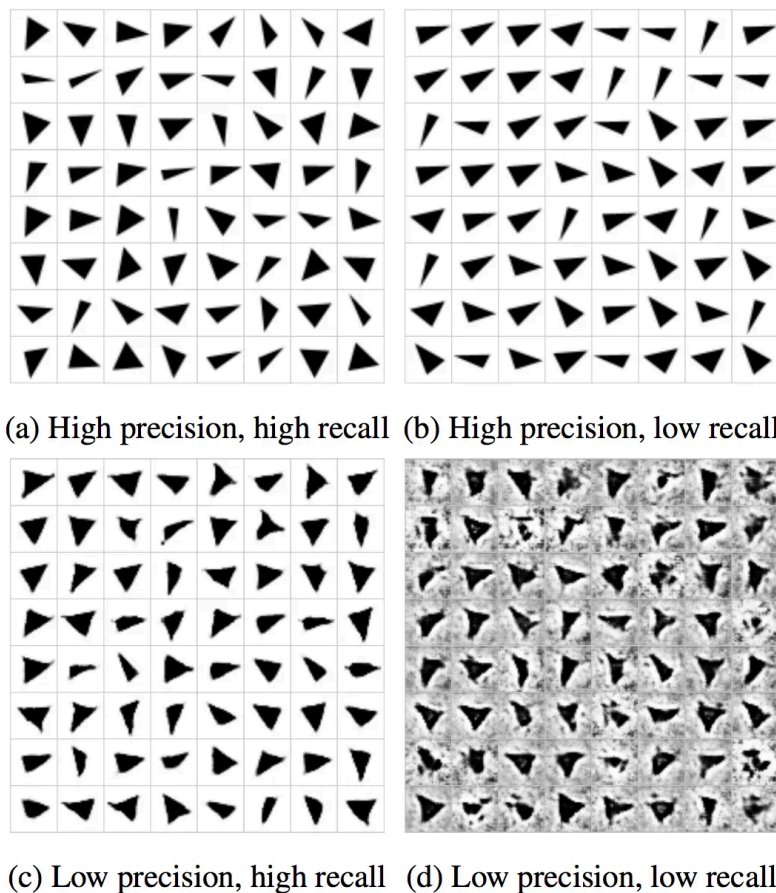


Рисунок 2.20—Демонстрація результату навчання моделі на тестовому датасеті трикутників що показує рівень повноти(recall) та точності (precision) [14]

Висновки до розділу 2

У цьому розділі детально розглянуто загальну структуру та алгоритм навчання генеративно-зустрічних мереж (GAN). Також було представлено опис існуючих архітектур моделей GAN, включаючи звичайну модель GAN, DCGAN, WGAN, WGAN-GP, SGAN, CGAN, та складений (прогресивний) GAN. Було розглянуто переваги та недоліки кожної із архітектур, на основі чого в подальшому буде обрано архітектуру мережі для поточної роботи.

Розділ присвячено також проблемам, що можуть виникнути під час навчання GAN, таким як згортання режиму, рівновага Неша та зникаючий градієнт. Закінчено розділ оглядом метрик, які застосовуються до GAN, таких як Inception Score (IS), Fréchet Inception Distance (FID), точність, повнота та оцінка F1.

Оглянута та проаналізована інформація стане корисною для подальшого застосування у роботі при створенні моделі GAN із застосуванням генетичного алгоритму.

РОЗДІЛ 3 ГІБРИДНИЙ АЛГОРИТМ МАШИННОГО НАВЧАННЯ ГЕНЕРАТИВНО-ЗМАГАЛЬНОЇ МЕРЕЖІ

У цьому розділі пропонується розглянути задачу створення генеративно-змагальної мережі із застосуванням генетичного алгоритму.

3.1 Огляд робіт з машинного навчання генеративно-змагальних мереж

Існує багато архітектур генеративно-змагальних мереж. Головна ціль, створити покращення класичної моделі з метою збільшення швидкості навчання, стабільності та ефективності загалом.

Розглянемо роботу [1] в якій представлена альтернативна формула функції втрат для генератора та дискримінатора у порівнянні із класичною формулою для моделей GAN:

$$\min_D - \mathbb{E}_{x \sim P_{\text{data}}} [\log D(x)] - \mathbb{E}_{z \sim P_z} \left[\sum_{n=1}^k \left(1 - \log D(G_n(z)) \right) \right],$$

де P_{data} множина реальних даних, P_z латентний простір, k кількість генераторів в ансамблі.

Функція втрат генератора:

$$\min_{G_i} \mathbb{E}_{z \sim P_z} \left[1 - \log \left(D(G_i(z) + \sum_{j \in \{1, \dots, k\}, j \neq i} \max(0, D(G_i(z)) - D(G_j(z)))) \right) \right],$$

де P_{data} множина реальних даних, P_z латентний простір, k кількість генераторів в ансамблі,

Ідея застосування цієї функції має декілька переваг: по-перше, генератори змагаються один з одним, що змушує гірші екземпляри ансамблю оптимізуватись більше. Цей ефект досягається за рахунок пошуку різниці між оцінками дискримінатора для G_i та G_j . Функція втрат дискримінатор натомість залишається незмінною, такою як в класичній моделі GAN із використанням ансамблю генераторів.

Мінусами цього підходу є те, що через змагальну природу цього підходу, генератори імітують найкращого представника, що призводить до наближення розподілів генераторів, і як результат, зменшення варіативності в згенерованих даних кожного з них.

Розглянемо роботу [5] в якій запропоновано новий алгоритм навчання, де кожен генератор змагається із дискримінатором в окремих парах. Далі ці натреновані моделі тренують головну пару D_0, G_0 . Перевагою цього алгоритму є можливість використання паралелізму: в той час як пари дискримінаторів та генераторів навчаються один з одним, головна модель генератора навчається не з самою множиною дискримінаторів, а з її копією. Цю множину скопійованих дискримінаторів в роботі названо дискримінаторами-месенджерами.

Недоліками цього алгоритму є висока навантаженість на комп'ютер при проведенні розрахунків, адже використовується велика кількість моделей, а кількість моделей дискримінатора ще й збільшена за рахунок копіювання.

Розглянемо топологію WGAN яка у вигляді функції втрат використовує відстань Вассерштейна, яка має більш плавний градієнт. За рахунок цього, досягається стійкість перед проблемою колапсу.

З іншої сторони, через плавність градієнту модель потребує більше часу щоб досягти конвергенції.

3.2 Постановка задачі

Задано множину справжніх зображень P_{data} , латентний простір P_z . Потрібно розробити покращення обраного алгоритму навчання з метою пришвидшення конвергенції моделі та стабілізації навчання.

Оскільки SGAN дуже навантажує систему а також має складну архітектуру, WGAN та WGAN-GP має повільну конвергенцію, то серед розглянутих топологій для даної роботи було обрано топологію DCGAN, через можливість генерувати високоякісні зображення та простоту архітектури, що майже не відрізняється від класичного GAN.

Для покращення алгоритму навчання з метою його пришвидшення було запропоновано гібридний алгоритм навчання, що складається із звичайного навчання з використанням алгоритму backpropagation та генетичного алгоритму.

Структура моделі дискримінатора складається із вхідного шару, трьох прихованих шарів та вихідного шару. Вхідний шар складається із згорткового шару, що отримує зображення розміром 64×64 , та активаційної функції LeakyReLU. Приховані шари складаються із згорткового шару, шару пакетної нормалізації та активаційної функції LeakyReLU. Вихідний шар складається із згорткового шару та активаційної функції сигмоїди, що фактично повертає оцінку чи зображення справжнє.

Структура моделі генератора складається із вхідного шару, трьох прихованих шарів та вихідного шару. Вхідний шар складається із розгорткового шару, що отримує на вхід випадковий вектор шуму, та активаційної функції ReLU. Приховані шари складаються із розгорткового шару, шару пакетної нормалізації та активаційної функції ReLU. Вихідний шар складається із розгорткового шару та активаційної функції тангенс.



Рисунок 3.1—Структура дискримінатора

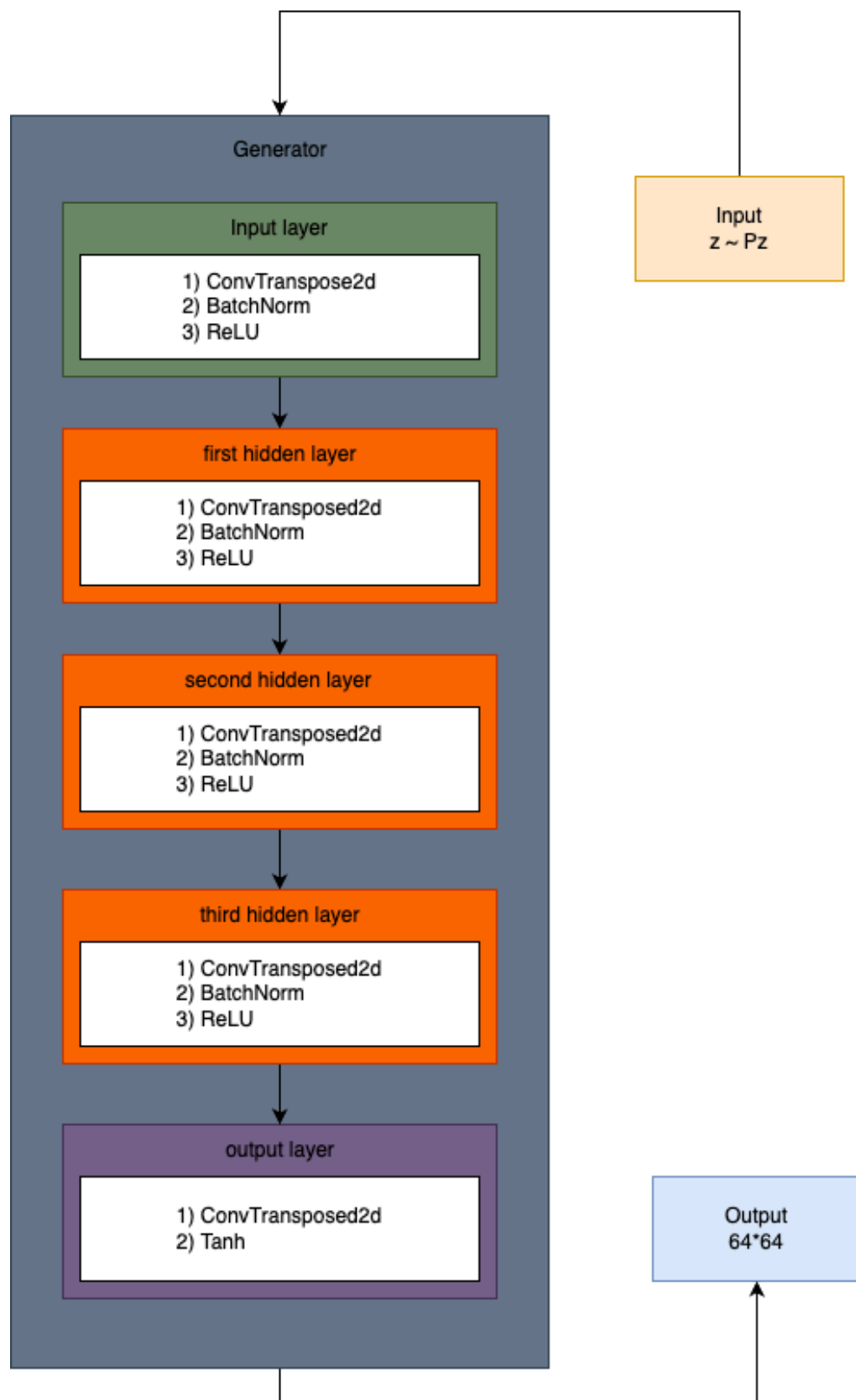


Рисунок 3.2—Структура генератора

3.3 Структура хромосоми генетичного алгоритму

Для застосування генетичного алгоритму потрібно визначити оператори селекції, кросоверу, мутації та фітнес функцію, а також спосіб кодування геному, визначити що є хромосомами, що є генами.

У даній задачі було визначено хромосоми, гени та популяцію наступним чином: у ролі хромосом (фактично особин) виступає модель генератора G_i , $i \in \{1 \dots m\}$, що має сталу кількість шарів із вагами. Множина що включає усі ваги кожного з шарів i є хромосомою. Тоді, у ролі генів будуть виступати ваги шарів моделі. Популяція складається із списку моделей генераторів, де кожен генератор має структуру, зазначену в розділі 3.2. При дослідженні літератури [2] було знайдено некласичний метод кросоверу. У даній роботі було використано та модифіковано процес екстракції знань батьків. Алгоритм екстракції знань виглядає наступним чином:

Дано: батьки G_a, G_b , отримані за допомогою оператора селекції, випадковий нормально розподілений вектор шуму z , Дискримінатор D , гіперпараметри оптимізатора Adam α, β_1, β_2 ,

- 1) Знайти оцінку дискримінатора для кожного з батьків $D(G_a(z)), D(G_b(z))$.
- 2) Створити нащадка G_{child} .
- 3) порахувати градієнт $g_{child} \leftarrow grad_{child} L_c(z)$.
- 4) $G_{child} \leftarrow Adam(g_{child}, \alpha, \beta_1, \beta_2)$.

Де

$$L_c = \sum_i^{n_{distillation}} \left\| G_{child}(z_i) - G_a(z_i) \right\|^2 \text{ If } [D(G_a(z_i)) > D(G_b(z_i))] + \\ + \sum_j^{n_{distillation}} \left\| G_{child}(z_j) - G_a(z_j) \right\|^2 \text{ If } [D(G_b(z_j)) > D(G_a(z_j))].$$

Фактично за допомогою функції L_c рахується помилка, що потім використовується для мінімізації середньоквадратичної відстані між розподілами згенерованих зображень нащадка і батьків для кожного $z_i, z_j \in P_z, i = 1 \dots n_{distillation}, j = 1 \dots n_{distillation}$, P_z латентний простір. Простими словами, нащадок прагне генерувати зображення схоже на зображення кращого з батьків, отриманого із випадкового вектора z , для i -го кроку дистиляції. Перевагами такої імплементації кросоверу є те, що в результаті буде створено нащадка, що має переваги кожного з батьків, на відміну від класичного підходу перетасування геномів, що не гарантує наближення до оптимума.

Оператор мутації був визначений як зсув випадкового гена (ваги шару) індивіда на випадкову величину, розподілену нормально та помножену на фактор мутації, що дозволяє контролювати силу мутації.

Оператор селекції був визначений як вибір двох випадкових індивідів із популяції, що не дорівнюють один одному. Індивід обирається із певною вірогідністю, що прямо пропорційна його фітнес функції.

Фітнес функція складається з двох компонент та виглядає наступним чином:

$$F = F_q + F_d,$$

де F_q оцінка якості згенерованого зображення, а F_d оцінка варіативності.

У ролі F_q фітнес функції використовується оцінка дискримінатора для зображення згенерованого індивідом популяції:

$$F_{q_i} = D(G_i(z)),$$

де $z \sim P_z$, випадковий вектор розподілений за нормальним законом.

Оцінка варіативності F_d вимірює відмінність двох згенерованих зображень одним генератором. Формально F_d визначена наступним чином:

$$F_{d_i} = \frac{1}{n_e} \sum_{k=1}^{n_e} E_{z_1, z_i \sim P_z} [||G_i(z_1) - G_i(z_k)||_1],$$

де n_e це кількість порівнянь згенерованих зображень.

Фактично вимірюється середня абсолютна похибка між розподілами згенерованих зображень. Чим більше похибка, тим різноманітніші зображення генерує генератор.

Окремою задачею при використанні генетичного алгоритму є налаштування гіперпараметрів, а саме вірогідності кросоверу $p_{crossover}$, для даного алгоритму із використанням екстракції знань, кількість ітерацій екстракції $n_{distillation}$, вірогідність мутації $p_{mutation}$, а також фактор мутації $c_{mutation}$. Гіперпараметри кросоверу впливають на швидкість пошуку оптимального рішення (або вірогідність перевикористання вже відомого рішення), а гіперпараметри мутації впливають на кількість створення нових варіантів (або на пошук рішення серед множини нових варіантів). За своєю природою ці оператори протидіють одне одному. Логічно що оператор кросоверу повинен домінувати, адже якщо це не так, система не буде прагнути до пошуку розв'язку. З іншої сторони, при надмірному домінуванні оператора кросовера система ризикує опинитися в області локального мінімуму (локальне оптимальне рішення). При ідеальному балансі, оператор кросоверу домінує, але мутація достатньо сильна щоб не дати індивіду залишитись в області локального мінімуму.

Окремої уваги потребує гіперпараметр розміру популяції m . При занадто малій популяції система немає достатньої кількості розв'язків що розглядаються в межах епохи, але з іншої сторони, алгоритм виграє у швидкості

через складність розрахунків. Якщо ж популяція велика, час на виконання алгоритму значно збільшиться, хоча цієї ситуації можна уникнути при використанні достатньої кількості обчислювальних ресурсів.

3.4 Вибір функції втрат

Класичне навчання GAN передбачає визначення функції втрат для моделей дискримінатора та генератора. За основу функції втрат було обрано класичну функцію втрат GAN, запропоновану Іаном Гудфелоу в роботі [9].

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))].$$

З цієї формули мінімаксної гри у моделі GAN маємо формулу функції втрат дискримінатора та генератора відповідно:

$$\max_D V(D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

$$\min_G V(G) = \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

Тим не менш, для використання цієї формули у поєднанні із генетичним алгоритмом потрібно перевизначити втрату дискримінатора, що залежить від розпізнавання фейкових зображень, створених популяцією генераторів.

$$\max V(D) = E_{x \sim P_x} [\log(D(x))] + E_{z \sim P_z} \left[\sum_{i=1}^m \log(1 - D(G_i(z))) \right],$$

де m – це розмір популяції.

Для зручності використання в кодї функція втрат дискримінатора змінена на мінімізацію – $V(D)$, тому формула набуває вигляду:

$$\min V(D) = - E_{x \sim P_x} [\log(D(x))] - E_{z \sim P_z} \left[\sum_{i=1}^m \log(1 - D(G_i(z))) \right].$$

Завдяки такій формулі, буде враховано результат кожного генератора, а дискримінатор навчатиметься більше. До того ж варто відмітити що на навчання дискримінатора сильніше будуть впливати кращі особини популяції, і навпаки. Це додатково посилить конкурентність і вимирання гірших генів.

Натомість, функція втрат генератора лишається незмінною, але видозміненою для зручності використання:

$$\min V(G) = - E_{z \sim P_z} [\log(D(G(z)))].$$

Зміна полягає у мінімізації функції втрат замість максимізації. Цей прийом дозволяє змусити функцію втрат генератора прямувати до 0 замість нескінченності, що в програмному кодї є принциповим, адже правильно визначити термін $+\infty$ на жаль не є можливим.

3.6 Визначення гіперпараметрів

Перед описом алгоритму є необхідність у визначенні деяких гіперпараметрів, а саме: кількість епох навчання N_{epoch} , розмір пакету навчання B_{size} , коефіцієнт епохи генетичного алгоритму $N_{GA_{epoch}}$, розмір популяції m ,

вірогідність кросоверу $p_{crossover}$, кількість циклів екстракції знань $n_{distillation}$, вірогідність мутації гена $p_{mutation}$, фактор мутації $c_{mutation}$, кількість порівнянь для виміру різноманітності в фітнес функції n_e . Для оптимізатора Adam були визначені наступні гіперпараметри: α що задає швидкість навчання, β_1 та β_2 параметри експоненційного згладжування для попереднього моменту.

3.5 Структура гібридного алгоритму машинного навчання генеративно-змагальної мережі

Розроблена структура гібридного алгоритму машинного навчання генеративно-змагальної мережі складається з двох основних компонентів: класичного навчання моделей із застосуванням backpropagation, та модифікованого генетичного алгоритму. Опишемо покроково запропонований алгоритм.

Дано: множина реальних зображень P_{data} , латентний простір P_z .

1. Створити дискримінатор D .
2. Створити популяцію генераторів G_i , $i \in \{1. \dots m\}$.
3. Впродовж $k = 1 \dots N_{epoch}$ повторювати:
 - 3.1. Поки можливо виокремити мінібатч $x \sim P_{data}$ розміру B_{size} повторювати:
 - 3.1.1. Виокремити випадковий вектор шуму $z \sim P_z$.
 - 3.1.2. Порахувати помилку дискримінатора $V(D)$ та градієнт $g_D \leftarrow grad_D V(D)$.
 - 3.1.3. $D \leftarrow Adam(g_D, \alpha, \beta_1, \beta_2)$.

3.1.4. Для кожного з генераторів G_i із популяції виконати:

3.1.4.1. Порахувати помилку генератора $V(G_i)$ та градієнт

$$g_{G_i} \leftarrow \text{grad}_{G_i} V(D).$$

3.1.4.2. $G_i \leftarrow \text{Adam}(g_{G_i}, \alpha, \beta_1, \beta_2)$.

3.1.5. Якщо $k \bmod N_{GA_{epoch}} = 0$ виконати:

3.1.5.1. Порахувати значення фітнес функції F_{G_i} для кожного з генераторів G_i .

3.1.5.2. Впродовж $l = 1 \dots m$ виконувати:

3.1.5.2.1. Вибрати двох батьків за допомогою функції селекції з урахуванням значень фітнес функції $G_{parent_1}, G_{parent_2} \leftarrow \text{selection}(F, G)$.

3.1.5.2.2. Виконати кросовер (алгоритм описаний в розділі 3.3) $G_{child} \leftarrow \text{crossover}(G_{parent_1}, G_{parent_2})$.

3.1.5.2.3. Виконати мутацію створеного нащадка $G_{child}^* \leftarrow \text{mutation}(G_{child})$.

3.1.5.2.4. Додати нащадка до нової популяції G_i^* .

3.1.5.3. Популяції G_i присвоїти G_i^* .

Модель навчається із використанням мінібатчів розмір яких заданий гіперпараметром. На кожній ітерації виокремлюється набір реальних зображень $x \sim P_x$ та випадковий вектор $z \sim P_z$. Далі дискримінатор D навчається за допомогою класичного алгоритму `backpropagation` із використанням функції втрат наведеною вище в цьому розділі. Наступним кроком є оптимізація кожного з генераторів із популяції G_i за допомогою класичного алгоритму `backpropagation`.

Кожні $N_{GA_{epoch}}$ мінібатчів виконується одна ітерація генетичного алгоритму створюється нащадок, який зрештою додається до нової популяції G_i^* що складається з нащадків отриманих під час виконання операцій кросоверу, та мутації. Далі алгоритм продовжує навчання таким чином впродовж N_{epoch} . Як результат, отримаємо популяцію генераторів G_i що вирішує поставлену задачу.

3.6 Налаштування гіперпараметрів

Для створеної моделі експериментально було встановлено наступні гіперпараметри: коефіцієнт кількості карт ознак у шарах генератора $N_{D_{neuron}} = 64$, коефіцієнт кількості карт ознак у шарах дискримінатора $N_{G_{neuron}} = 64$, розмір випадкового вектора $Z_{dim} = 100$, кількість епох навчання $N_{epoch} = 5$, розмір пакету навчання $B_{size} = 128$, коефіцієнт епохи генетичного алгоритму $N_{GA_{epoch}} = 70$, розмір популяції $m = 5$, вірогідність кросоверу $p_{crossover} = 0.7$, кількість циклів екстракції знань $n_{distillation} = 5$, вірогідність мутації гена $p_{mutation} = 0.25$, фактор мутації $c_{mutation} = 0.003$, кількість порівнянь для виміру різноманітності в фітнес функції $n_e = 5$. Для оптимізатора Adam були обрані наступні гіперпараметри: $\alpha = 2e - 4$, $\beta_1 = 0.5$, $\beta_2 = 0.999$.

3.7 Аналіз отриманих результатів

Дослідження були проведені для задачі MNIST рукописних цифр. Нижче наведені приклади виконання програми.

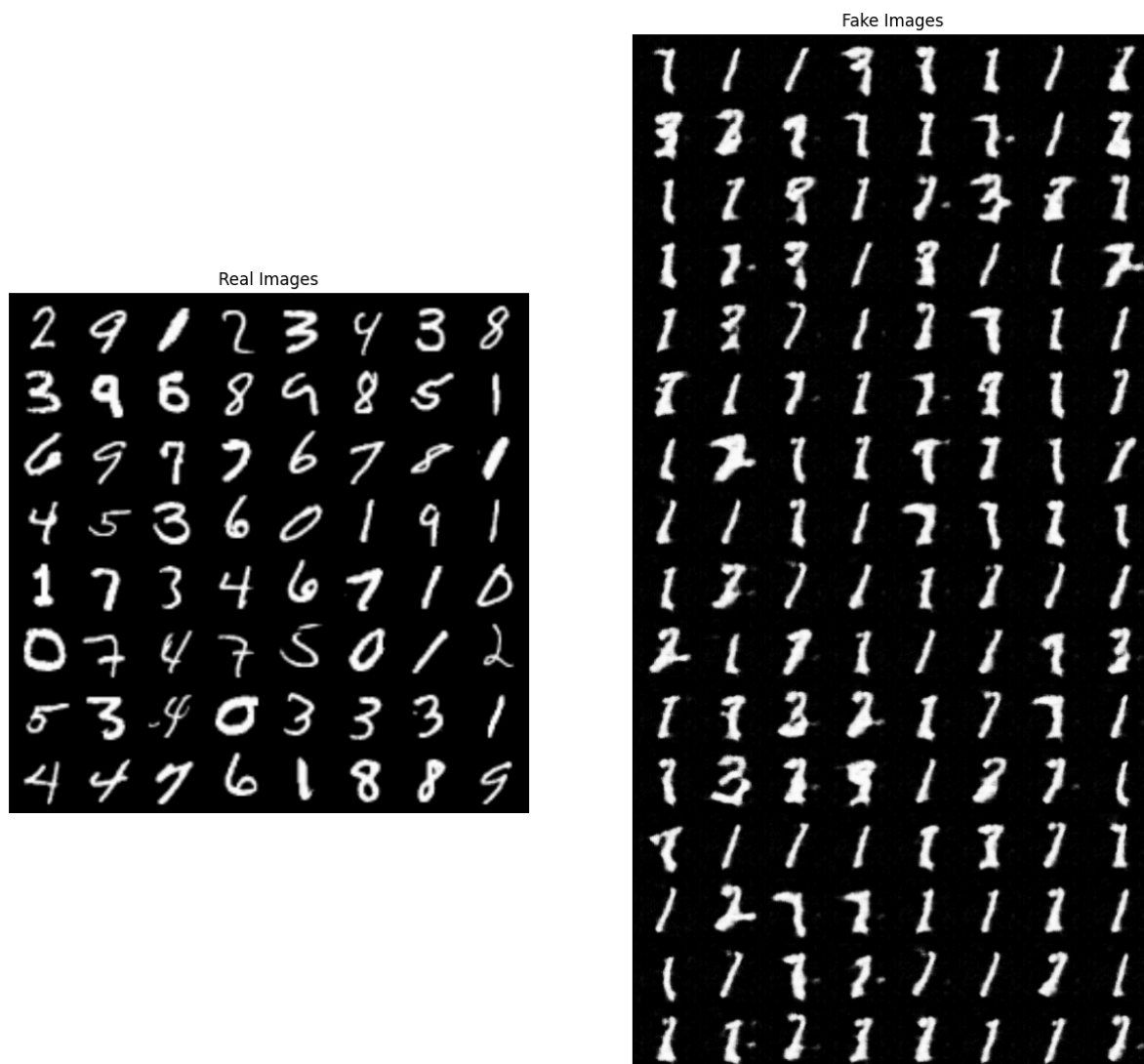


Рисунок 3.3 – Приклад роботи одного з генераторів програми 1 епоха.

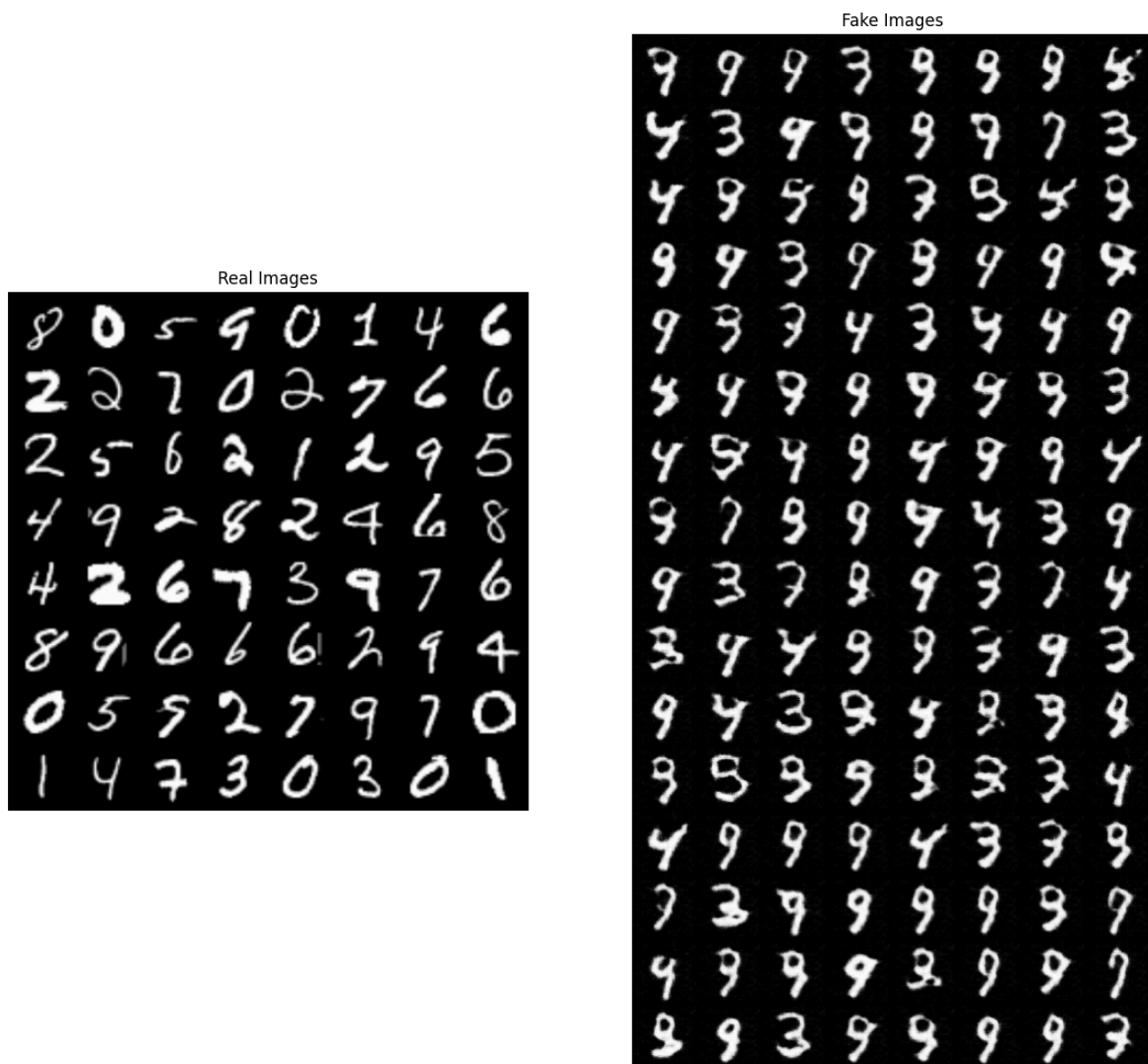


Рисунок 3.4 – Приклад роботи одного з генераторів програми 1 епоха.

Як видно з рис. 3.3 та 3.4 у генераторів спостерігається частковий колапс режимів. Це може бути спричинено перенавчанням моделі дискримінатора, тому структуру дискримінатора було розширено із додаванням шарів дропауту.

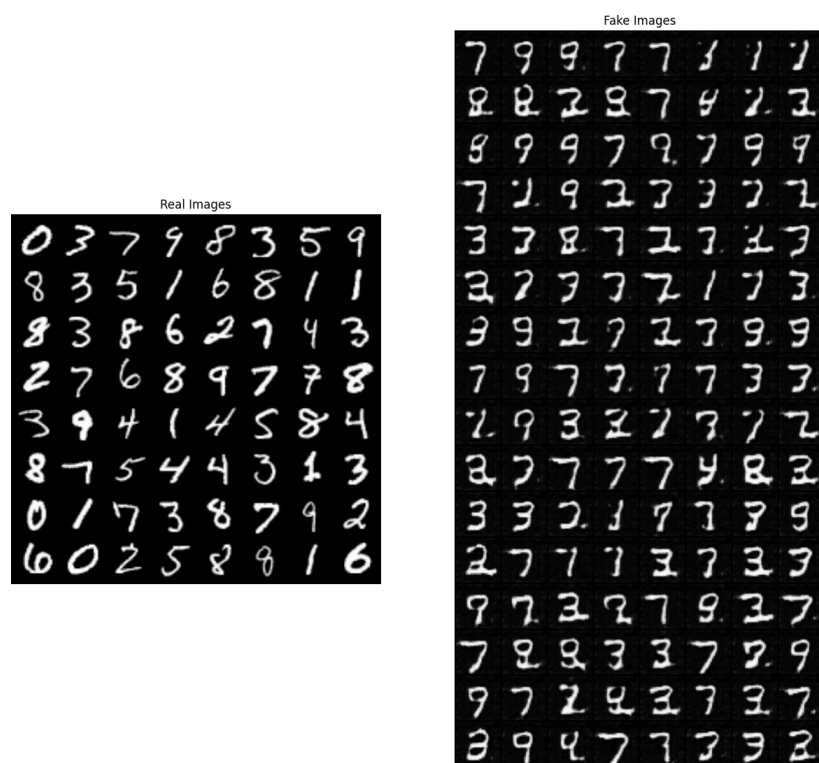


Рисунок 3.5 – Приклад роботи моделі з дропаутом 5 епох

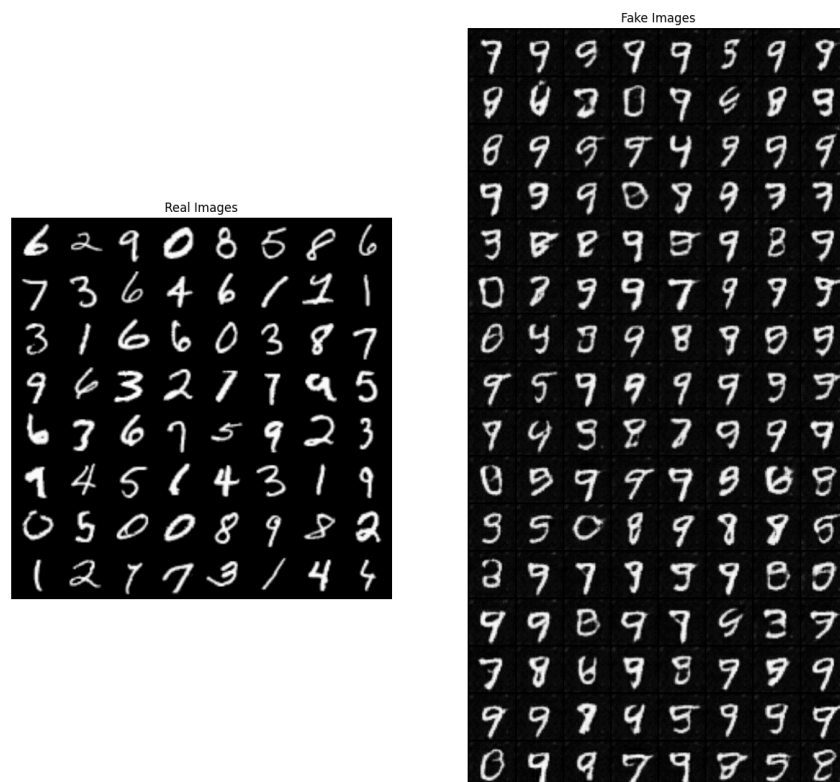


Рисунок 3.6 – Приклад роботи моделі з дропаутом 5 епох

Як видно з рис., збільшення кількості епох та додавання дропаут шарів до дискримінатора частково вирішує проблему колапсу режиму.

Для дослідження ефективності використання генетичного алгоритму було проведено дослід, в якому для навчання популяції не використовували кроки генетичного алгоритму.

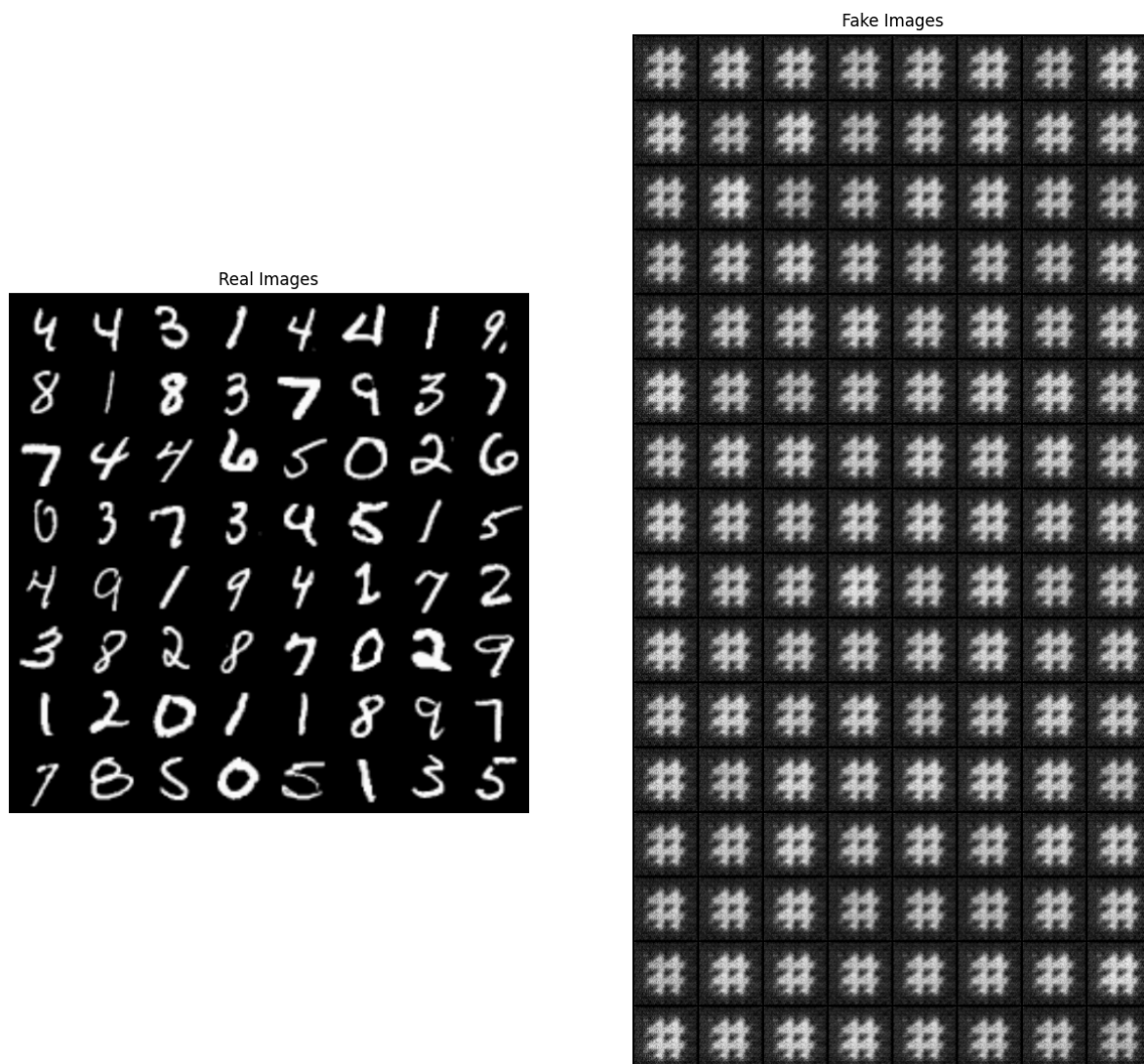


Рисунок 3.7 – Результат навчання популяції без використання генетичного алгоритму 1 епоха.

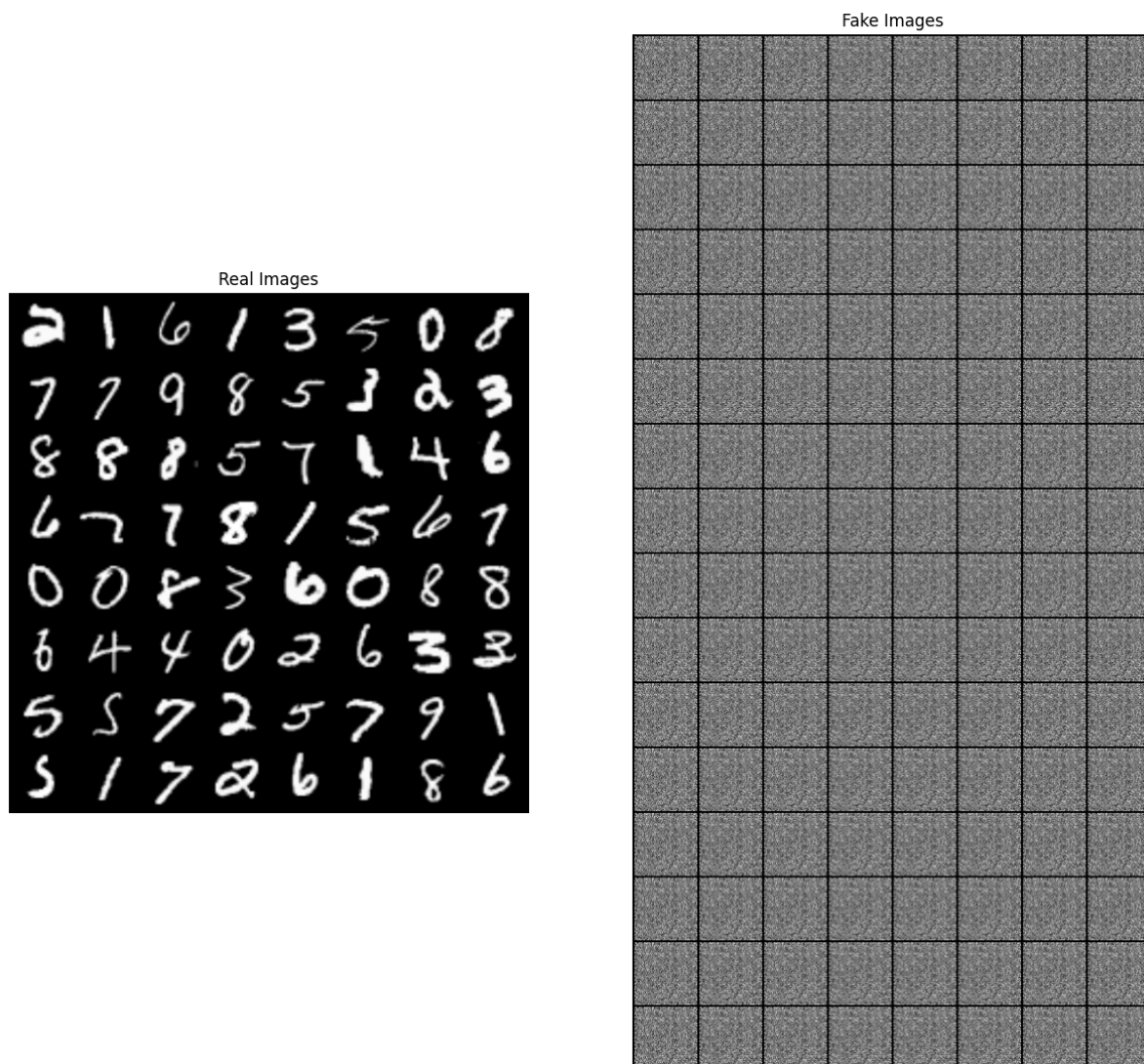


Рисунок 3.8 – Результат навчання популяції без використання генетичного алгоритму 1 епоха

Як видно з рис. 3.7 та 3.8, без використання генетичного алгоритму навчання нестабільне і часто призводить до колапсу. Завдяки генетичному алгоритму на кожному кроці k ($k \bmod N_{GA_{epoch}} = 0$) гірші представники популяції відсіюються, а кращі створюють нащадків із своїми генами, завдяки чому при тренуванні моделі дискримінатор та популяція генераторів знаходяться в балансі. В свою чергу, $N_{GA_{epoch}}$ ітерацій потрібно для визначення кращих представників популяції.

Для виміру ефективності використання запропонованого алгоритму було порівняно результат роботи запропонованого алгоритму та класичного алгоритму навчання.

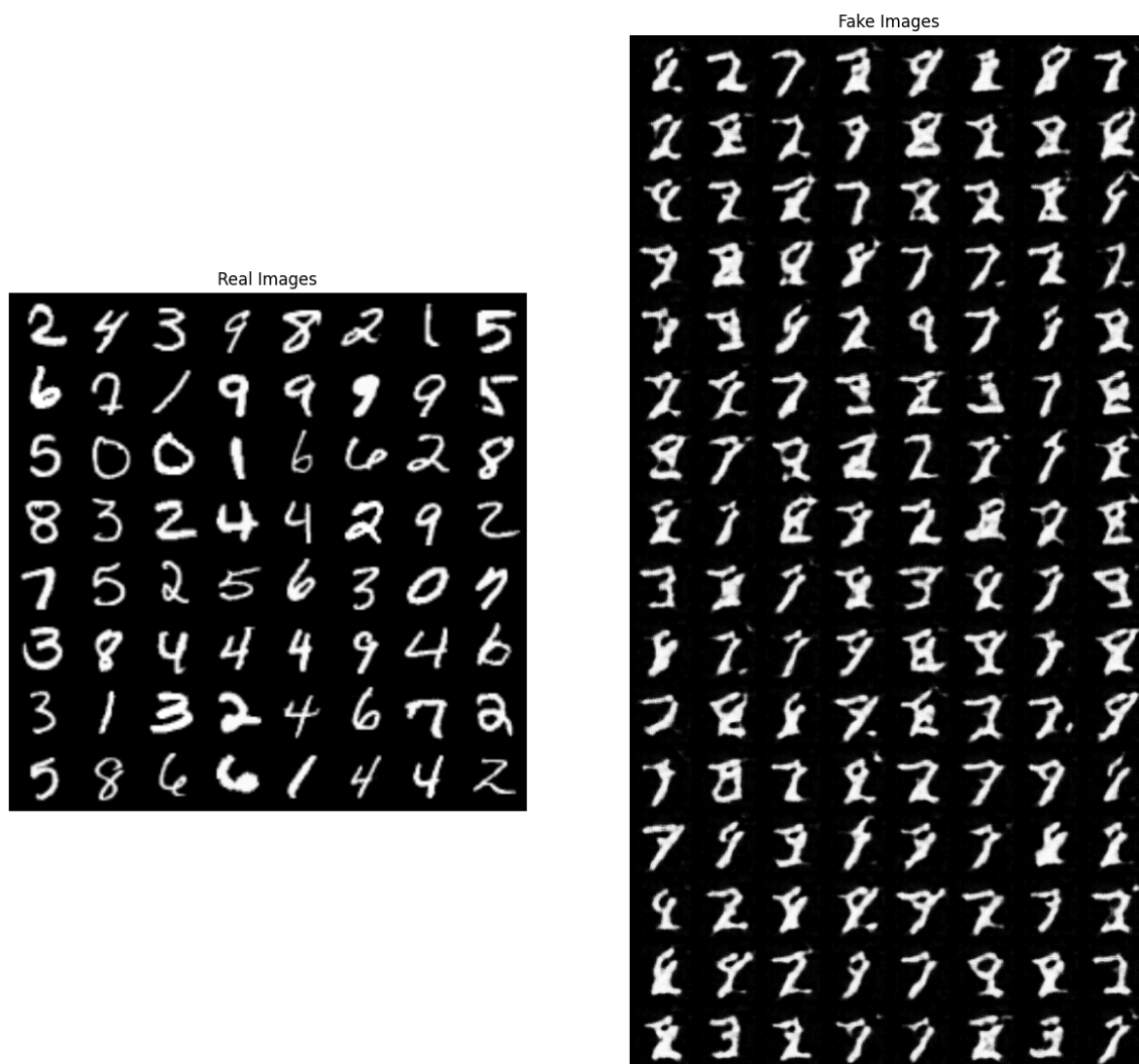


Рисунок 3.9 – Результат навчання класичним алгоритмом 1 епоха

Порівняємо рис. 3.6 та 3.7. На обох рисунках спостерігається частковий колапс режимів. Та на відміну від класичного алгоритму, результатом роботи запропонованого алгоритму є популяція із генераторів, що захоплюють різні режими, як видно з рис. 3.5, 3.6, а отже можна використати популяцію для створення одного генератора, що може генерувати різні режими. Порівняємо

рис. 3.4 та 3.9. Як видно з рисунків, запропонований алгоритм навчання вже на першій епосі досягає конвергенції швидше ніж звичайний алгоритм, адже на рис. 3.4 видно більш чіткі контури, та реалістичніші цифри, ніж на рис. 3.9.

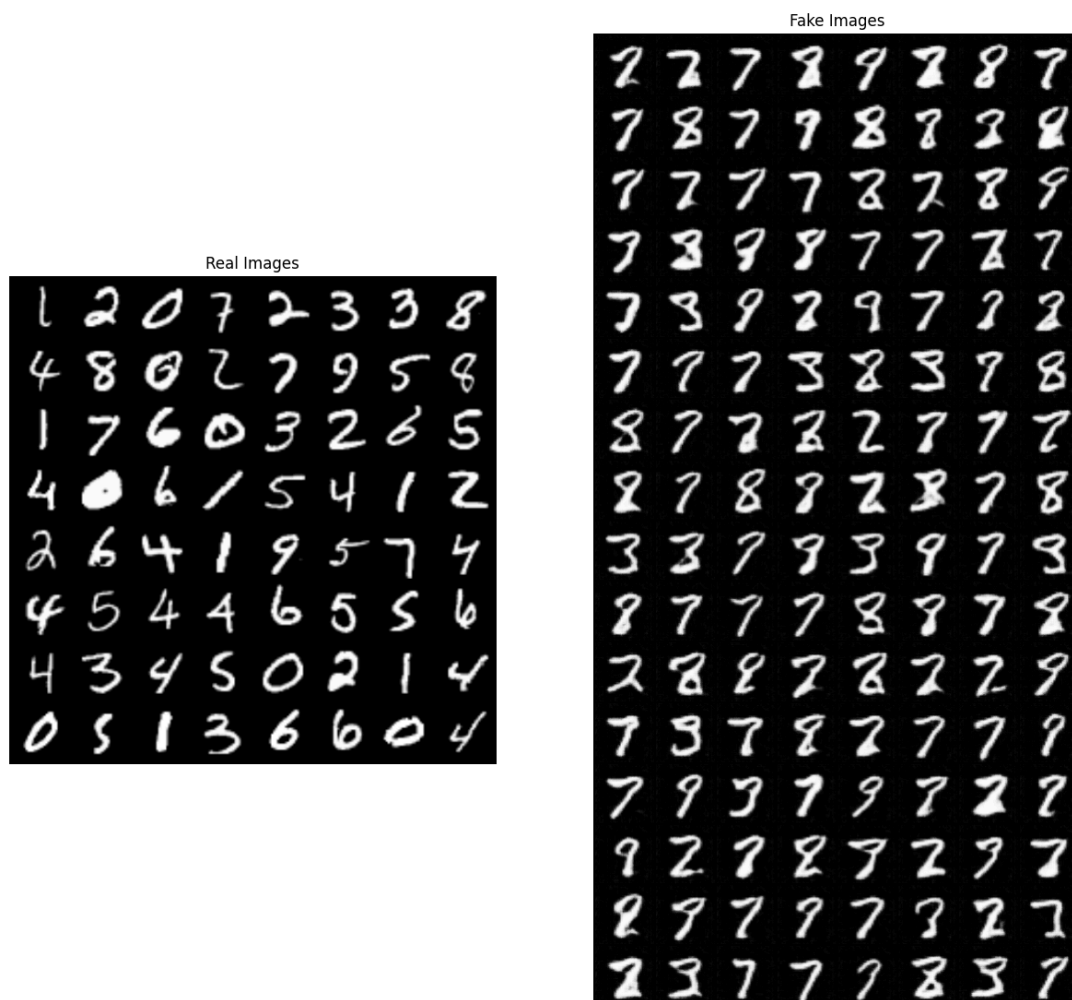


Рисунок 3.10 – Результат навчання класичним алгоритмом 5 епох

Висновки до розділу 3

Під час роботи було запропоновано гібридний алгоритм навчання моделі GAN що складається із навчання моделі класичним методом із використанням алгоритму backpropagation та генетичного алгоритму. Було розроблено власну

імплементацію операторів кросоверу та мутації. Розроблена модель частково розв'язує задачу MNIST рукописних цифр. Аналіз результатів роботи запропонованого алгоритму показує певне покращення у порівнянні із класичним алгоритмом навчання, а саме варіативність згенерованих зображень та швидкість конвергенції. Порівняння результатів роботи запропонованого алгоритму і класичного на 1 епосі та на 5 це підтверджує.

Експеримент, направлений на виявлення ефективності використання генетичного алгоритму показав його ефективність в стабілізації навчання та пришвидшенні конвергенції, дослід демонструє результат навчання з використанням генетичного алгоритму та без нього.

РОЗДІЛ 4 РОЗРОБКА ВЛАСНОГО СТАРТАП ПРОЄКТУ

GAN моделі (Generative Adversarial Networks) відіграють ключову роль у створенні та генерації нового контенту за допомогою штучного інтелекту. Основним призначенням GAN є навчання моделі генерувати дані, які не відрізняються від реальних даних. Це відкриває безліч можливостей у таких областях, як комп'ютерний зір, обробка природної мови, медицина, мистецтво та інші. Важливість GAN моделей визначається їхньою здатністю генерувати високоякісний та реалістичний контент. Ці моделі використовуються для створення фотореалістичних зображень, відео, аудіо та інших форм медіа. Крім того, GAN відкриває нові можливості у сферах віртуальної реальності, розширеної реальності та дизайну продуктів. Основною проблемою при використанні моделей GAN є швидкість навчання, отже є запит на створення методу навчання що швидший за класичний.

Мета стартапу – створити просту систему для генерації зображень що можуть бути використані для подальшого навчання інших моделей, або у комерційній діяльності.

4.1 План розробки стартапу та масштабування його на ринок

Наведемо план розробки стартапу та виведення його на ринок. Спочатку треба провести маркетинговий аналіз, який включає в себе:

- конкурентний аналіз, щоб зрозуміти, якими методами вирішення проблем вже користуються люди;
- формування ідеї самого проєкту та виділення цільової аудиторії;
- розробити стратегію виведення товару на ринок, базуючись на аналізі ринкового середовища.

Наступним кроком являється організація самого стартапу. На цьому етапі мають бути:

- складений весь план та побудований таймлайн розробки та запуску продукту;
- запланований обсяг виробництва та оцінений потенційний обсяг ресурсу, який буде потрібен для виконання плану;
- розраховані витрати, необхідні для реалізації проєкту, та витрати на запуск проєкту.

Далі необхідно виконати фінансово-економічний аналіз та оцінити ризики стартап-проєкту, в межах якого:

- визначити обсяг інвестиційних втрат;
- розрахувати основні фінансово-економічні показники проєкту (собівартість, ціну продукту/послуги, податковий збір та чистий прибуток) та визначити показники інвестиційної привабливості проєкту (рентабельність продажів, період окупності проєкту);
- визначити основні ризики проєкту та способи для їх запобігання.

Фінальним кроком являється розробка заходів з комерціалізації продукту. Цей крок являється важливим для масштабування та збільшення розмірів продукту.

Для того, щоб залучити інвесторів та знайти різні способи фінансування проєкту, необхідно:

- провести дослідження на предмет інтересів потенційних інвесторів та бізнесів;
- скласти інвестиційну пропозицію, яка включає в себе як опис самого продукту та його теперішні розміри, так і можливі шляхи розширення та розвитку;
- обрати канали комунікації із потенційно зацікавленими персонами.

Далі наведено результати виконання кожного з описаних кроків.

4.2 Опис ідеї стартап-проєкту

Стартап-проєкт полягає у вирішенні проблеми шви. Суть продукту стартапу полягає у тому, що розроблена система приймає на вхід зображення (датасет), навчає модель генерувати схожі зображення, та повертає на виході необхідну кількість схожих зображень.

Таблиця 4.1 – Інформаційна карта стартап-проєкту

Назва проєкту	Generator.io
Автор проєкту	Гречухов Микита Іванович
Коротка анотація	Продукт буде отримувати приклад зображень як датасет, а на виході повертати необхідну кількість
Термін реалізації проєкту	12 місяців
Опис проблеми, яку вирішує проєкт	Створення зображень що схожі на оригінальне
Необхідні ресурси	Приміщення з комп'ютерами, доступом до Інтернету, доступ до електромережі. Програмне забезпечення для розробки, хмарне сховище для даних, антивірусні програми. Фінансові кошти на оплату заробітної плати виконавцям на термін 12 місяців, а також на такі витрати як: оренда приміщення, комунальні послуги, оренда хмарного сховища тощо
Головні цілі та завдання проєкту	Метою проєкту є створення програми що буде створювати зображення схожі на вхідні
Очікувані результати	Реалізована система та залучення інших департаментів для її використання.

4.3 Технологічний аудит ідеї проєкту

Тепер можна розібрати ідею стартапу та провести конкурентний аналіз. У табл. 4.2 наведений опис ідеї стартапу.

Таблиця 4.2 – Опис ідеї стартапу

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Основна ідея полягає у створенні комплексної системи, яка буде створювати нові зображення для подальшого використання у роботі з нейронними мережами, або в комерційних цілях	Розширення датасету даних, генерація контенту	Користувачі можуть використовувати програму для розширення датасету для навчання нейромереж

Порівняльний аналіз конкурентів проєкту провести наразі неможливо, оскільки в Україні поки що не існує подібного інструменту.

Далі аналізуємо реальність технічно здійснити ідею проєкту (табл. 4.3).

Таблиця 4.3 – Технологічна здійсненність продукту

№ п/п	Ідея проєкту	Технології і реалізації	Наявність технологій	Доступність технологій
1	Створення програми що буде створювати зображення	Використання мови програмування Python	Наявні	Доступні
2	схожі на вхідні	Використання мови програмування C++	Не наявні, необхідні доопрацювання	Доступні

Закінчення табл. 4.3

№ п/п	Ідея проєкту	Технології і реалізації	Наявність технологій	Доступність технологій
3	Створення програми що буде створювати зображення схожі на вхідні	Використання мови програмування R	Не наявні, необхідні доопрацювання	Доступні
Обрана технологія реалізації ідеї проєкту: Python				

4.4 Аналіз ринкових можливостей запуску стартап-проєкту

Далі проведемо попередній аналіз ринку для запуску стартап-проєкту (табл. 4.4).

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап проєкту

№ п/п	Показники ринку (найменування)	Характеристика
1	Кількість головних гравців, од	5
2	Загальний обсяг продаж, грн/ум.од	4000
3	Динаміка ринку (якісна оцінка)	Позитивна, зростає

Закінчення табл. 4.4

№ п/п	Показники ринку (найменування)	Характеристика
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	13%

Тепер проведемо характеристику потенційних клієнтів, які можуть бути зацікавлені в проєкті (табл. 4.5).

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреби, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Генерація схожих зображень, які були представлені перед тренуванням як датасет	Компанії виробники штучного інтелекту, держава (міністерство цифрової трансформації)	Переслідують різні цілі генерації зображень	Простота та автоматизація використання

Обрахуємо фактори загроз (табл. 4.6) та можливостей (табл. 4.7). Проаналізуємо загрози, щоб зрозуміти можливі перешкоди при запуску продукту на ринок. Фактори можливостей ще треба обрахувати, щоб знати усі сприятливі умови та по можливості ними скористатися.

Таблиця 4.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	Хоча ринок є відкритим і неосвоєним, можуть з'явитися конкуренти, більші компанії, що запропонують кращий продукт	Знайти точки додаткової цінності для користувача, ввести нові фічі
2	Ціна збуту	Небізнесова сфера збуту може бути недостатньо лояльною та пропонувати занижку ціну	Сфокусуватися на якості, продумати маркетингову стратегію продукту

Таблиця 4.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Автоматизація	Продукт дає змогу замінити великий обсяг мануальної роботи	Зробити акцент при маркетингу, продовжувати розвиток як окремого продукту
2	Якість фінального продукту	Якісність згенерованих зображень та їх кількість можуть спонукати користувачів до використання саме цього продукту	Реалізувати зручний інтерфейс для завантаження
3	Швидкість та простота	Прискорений алгоритм навчання дає змогу швидше просуватись в основній роботі над моделлю	Пропонувати моделі з найкращими результатами, надавати необхідну технічну підтримку

Далі розглянемо питання конкуренції, а саме визначимо її тип та рівень (табл. 4.8).

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	У чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії щоб бути конкурентоспроможною)
1. Вказати тип конкуренції: недосконала конкуренція	Не представлено продуктів та експертів	Зробити максимальним збут застосунку
2. За рівнем конкурентної боротьби: міжнародний	Наявні проєкти, розроблені та можуть бути доступні у всьому світі	Розширити цільову аудиторію, розробити варіант генерації різних типів даних, не лише зображень
3. За галузевою ознакою: міжгалузева	Може працювати з різними галузями	Персоналізація
4. Конкуренція за видами товарів: товарно-родова	Конкуренція з іншими системами та експертами	Підтримувати та покращувати якість існуючих функцій
5. За характером конкурентних переваг: нецінова	Компанії не пропонують різну якість	Розробляти якісніші алгоритми і моделі
6. За інтенсивністю: немарочна	Не представлені компанії із сильним брендом	Предметно створити комунікаційну стратегію для свого бренду

Далі необхідно виконаємо аналіз конкуренції за моделлю 5 сил конкуренції Майкла Портера (табл. 4.9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти у галузі	Потенційні конкуренти	Постачальники	Клієнти	Товарозамінники
	Конкуренти відсутні	Якість, ціни, кількість користувачів, капіталовкладення	Фактори сили постачальників	Контроль якості, порівняння цін	Сила бренду, якість, ціна, масштаби
Висновки	Конкуренція з мінімальною інтенсивністю, а також запит користувачів на ринку	Можливості входження на ринок, нові потенційні конкуренти	Постачальники відсутні	Клієнти не диктують умови роботи на ринку	Товарозамінники відсутні

Маючи результати аналізу конкуренції (табл. 4.9), характеристики ідеї стартап-проєкту (табл. 4.4), характеристики потенційних клієнтів і їх вимоги до

продукту (табл. 4.5) та фактори ринкового середовища (табл. 4.6 і 4.7) було сформульовано та обґрунтовано перелік факторів конкурентоспроможності (табл. 4.10).

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проєктів значущим)
1	Автоматизація	Продукт дає змогу замінити великий обсяг мануальної роботи
2	Якість фінального продукту	Якісність згенерованих зображень та їх кількість можуть спонукати користувачів до використання саме цього продукту
3	Швидкість та простота	Прискорений алгоритм навчання дає змогу швидше просуватись в основній роботі над моделлю

Тепер можна провести аналіз сильних та слабких сторін продукту (табл. 4.11).

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін системи

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів			
			-3	-2	-1	0
1	Автоматизація	17	+			
2	Якість фінального продукту	16	+			
3	Швидкість та простота	19	+			

Далі проведемо SWOT-аналіз продукту (табл. 4.12).

Таблиця 4.12 – SWOT-аналіз стартап-проєкту

Сильні сторони Автоматизація Якість фінального продукту Швидкість та простота	Слабкі сторони Відсутність сильного бренду Не сформована база клієнтів Не підключені альтернативні канали маркетингу
Можливості Покращення системи інтеграції з великими компаніями	Загрози Нові системи та експерти Збут

Завдяки SWOT-аналізу, ми змогли визначити сильні та слабкі сторони, можливості та загрози, пов'язані з конкуренцією та плануванням стартап-проєкту. Далі спроектуємо альтернативну ринкову поведінку для інтеграції стартап-проєкту на ринок та приблизний час реалізації системного комплексу, з урахуванням потенційних проєктів, що можуть бути виведені на ринок та наведемо результати у табл. 4.13.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап проєкту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Вихід на ринок з нижчою якістю	60 %	3 місяці
2	Пропонувати одразу платне використання	75 %	6 місяців
3	Представлення користувачам системи без інтерфейсу	40 %	7 місяців

У даному пункті був проведений детальний аналіз ринку та продукту. Також відповідно до результатів проведеного конкурентного аналізу, визначених факторів ринку та його сприятливості, описання ідеї та характеристик стартап-проєкту, робимо висновок, що існують дуже сприятливі умови для виходу продукту на ринок.

4.5 Розроблення ринкової стратегії стартап-проєкту

Для розробки ринкової стратегії продукту, у першу чергу, необхідно проаналізувати цільову аудиторію проєкту (табл. 4.14).

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит у межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Держава, дослідницькі центри	Середня	60 %	Низька	Середня
2	Великі бізнеси	Низька	30 %	Висока	Висока
3	Малі та середні бізнеси	Середня	55 %	Низька	Середня
4	Приватні користувачі	Середня	40 %	Низька	Висока
Які цільові групи обрано: 1					

Маючи аналіз цільових груп, далі визначимо базову стратегію розвитку продукту (табл. 4.15)

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Постійне оновлення і покращення продукту	Ринкове позиціонування на представників державної діяльності	Масштабування та максимізація	Оптимальні витрати

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (табл. 4.16, 4.17).

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Так	Шукати нових та забирати існуючих	Так, якщо ті популярні у користувачів	Заповнення конкурентної ніші

Таблиця 4.17 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспро- можні позиції власного стартап проєкту	Вибір асоціацій, які мають сформувану комплексну позицію власного проєкту (три ключових)
Універсальність Простота у використанні Якість результатів	Оптимальні витрати	Автоматизація Якість результатів Швидкість та простота	Система, яка генерує знімки рентгену легенів для подальшого дослідження туберкульозу

4.6 Розроблення маркетингової програми стартап-проєкту

Після проведеного комплексного аналізу, можемо повноцінно описати ключові переваги концепції потенційного товару (табл. 4.18) та побудувати концепцію маркетингових комунікацій (табл. 4.19).

Таблиця 4.18 – Ключові переваги концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Різномані- тність згенерова- них даних	Різноманітність покращує навчання моделей	Постійне покращення алгоритму що підтримує різноманітність
2	Простота у використа- нні	Максимальна простота у використанні	Такого виду систему може використовувати користувач, після мінімального інструктажу

Таблиця 4.19 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Пошук спеціалізованих систем	Державні замовлення розробних або дослідницьких проєктів	Автоматизація Простота Швидкість Якість	Поєднати повідомлення про те, що це якісна система, яка є швидкою	Пропозиції співпраці з науковими департаментами, середнім бізнесом

Висновки до розділу 4

Даний розділ був присвячений дослідженню стартап-проєкту. В якості такого була представлена система генерації зображень, що схожі на наданий датасет.

У межах цього розділу вивчалася розробка стратегій виходу на ринок та маркетингових стратегій для їхньої реалізації. Зокрема, розглядалися переваги цього ринку, який є сприятливим і поки що не має конкурентів. Запропонована система визначається як універсальна та легкодоступна, що створює перспективи для стартап-проєкту стати монополістом на ринку.

Також були проведені аналіз сильних і слабких сторін проєкту, SWOT-аналіз, дослідження конкурентів і цільової аудиторії. На основі всіх проведених досліджень був розроблений концепт маркетингової стратегії для обраної цільової аудиторії.

ВИСНОВКИ

Моделі генеративно-змагальних мереж мають на меті навчити генератор, який на основі датасету. Оскільки цей напрямок машинного навчання виник нещодавно та, на даний момент, ще активно розвивається, метою дослідження було дослідити існуючі топології моделей навчання без підготовки заснованих на вкладеннях та дослідити проблемні точки, які притаманні кращим моделям цього класу.

У ході дослідження було детально розібрано принцип роботи запропонованого алгоритму гібридного навчання, описано його компоненти, для генетичного алгоритму визначено хромосому, ген та популяцію. Було власноруч натреновано та проведено експерименти з моделлю, що навчалась запропонованим алгоритмом, для задачі MNIST. Проаналізовано результати експериментів та доведено ефективність запропонованого алгоритму.

ПЕРЕЛІК ПОСИЛАНЬ

1. Wang X. Shared Loss between Generators of GANs. 2022. arXiv:2211.07234v1. DOI: <https://doi.org/10.48550/arXiv.2211.07234>
2. Junjie Li, Jingyao Li, Wenbo Zhou, and Shuai Lu. IE-GAN: An Improved Evolutionary Generative Adversarial Network Using a New Fitness Function and a Generic Crossover Operator. 2022. arXiv:2109.11078v3. DOI: <https://doi.org/10.48550/arXiv.2109.11078>
3. Junjie Li, Junwei Zhang, Xiaoyu Gong, Shuai Lu. Evolutionary Generative Adversarial Networks with Crossover Based Knowledge Distillation. 2021. arXiv:2101.11186v2. DOI: <https://doi.org/10.48550/arXiv.2101.11186>
4. Quan Hoang, Tu Dinh Nguyen, Trung Le, Dinh Phung. MGAN: TRAINING GENERATIVE ADVERSARIAL NETS WITH MULTIPLE GENERATORS. 2018. URL: <https://openreview.net/pdf?id=rkmu5b0a-> (дата звернення 19.10.2023).
5. Chavdarova T., Fleuret F. SGAN: An Alternative Training of Generative Adversarial Networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 9407-9415. DOI:10.1109/CVPR.2018.00980
6. Ghosh A. et al. Multi-Agent Diverse Generative Adversarial Networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. pp. 8513-8521. URL: https://openaccess.thecvf.com/content_cvpr_2018/papers/Ghosh_Multi-Agent_Diverse_Generative_CVPR_2018_paper.pdf.
7. Mario Lucic, Karol Kurach, Marcin Michalski, Olivier Bousquet, Sylvain Gelly. Are GANs Created Equal? A Large-Scale Study. 2018. arXiv:1711.10337v4. DOI: <https://doi.org/10.48550/arXiv.1711.10337>
8. Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a

- Local Nash Equilibrium. 2018. arXiv:1706.08500v6. DOI: <https://doi.org/10.48550/arXiv.1706.08500>
9. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Nets. 2014. arXiv:1406.2661v1. DOI: <https://doi.org/10.48550/arXiv.1406.2661>
 10. Ishaan Gulrajani¹, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville. Improved Training of Wasserstein GANs. 2017. arXiv:1704.00028v3. DOI: <https://doi.org/10.48550/arXiv.1704.00028>
 11. Hoang Thanh-Tung, Truyen Tran. Catastrophic forgetting and mode collapse in GANs. 2020. arXiv:1807.04015v8. DOI: <https://doi.org/10.48550/arXiv.1807.04015>
 12. Martin Arjovsky, Leon Bottou. TOWARDS PRINCIPLED METHODS FOR TRAINING GENERATIVE ADVERSARIAL NETWORKS. 2017. arXiv:1701.04862v1. DOI: <https://doi.org/10.48550/arXiv.1701.04862> .
 13. Anas BRITAL. Genetic Algorithm Explained: Everything you need to know About Genetic Algorithm. 2021. URL: <https://medium.com/@AnasBrital98/genetic-algorithm-explained-76dfbc5de85d> (дата звернення 19.10.2023).
 14. Jonathan Hui. GAN – How to measure GAN performance? 2018. URL: <https://jonathan-hui.medium.com/gan-how-to-measure-gan-performance-64b988c47732> (дата звернення 19.10.2023).
 15. Jonathan Hui. GAN – What is Generative Adversarial Networks GAN?. 2018. URL: <https://jonathan-hui.medium.com/gan-whats-generative-adversarial-networks-and-its-application-f39ed278ef09> (дата звернення 19.10.2023).
 16. Jonathan Hui. GAN – Why it is so hard to train Generative Adversarial Networks! 2018. URL: <https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b> (дата звернення 19.10.2023).

17. Jonathan Hui. GAN – A comprehensive review into the gangsters of GANs (Part 2) 2018. URL: <https://jonathan-hui.medium.com/gan-a-comprehensive-review-into-the-gangsters-of-gans-part-2-73233a670d19> (дата звернення 19.10.2023)
18. Dewi C., Chen R.-C., Liu Y.-T., Yu H. Various Generative Adversarial Networks Model for Synthetic Prohibitory Sign Image Generation. Applied Sciences. Vol. 11. No. 7, p. 2913. 2021. DOI: <https://doi.org/10.3390/app11072913>
19. Liis Kolberg, Mari-Liis Allikivi. DCGAN for classroom images. 2018. URL: <https://neuro.cs.ut.ee/wp-content/uploads/2018/02/DCGAN.pdf> (дата звернення 19.10.2023)
20. Roy Ganz. A Review of Generative Adversarial Networks – part 1. 2020. URL: <https://medium.com/analytics-vidhya/a-review-of-generative-adversarial-networks-part-1-a3e5757a3dc2> (дата звернення 19.10.2023)
21. Ainur Gainetdinov. GAN Mode Collapse Explanation. 2023. URL: <https://pub.towardsai.net/gan-mode-collapse-explanation-fa5f9124ee73> (дата звернення 19.10.2023)

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО ПРОДУКТУ

main.py

```
import copy

import numpy as np
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.utils as vutils
import matplotlib.pyplot as plt
from typing import List, Tuple
from torch.nn.functional import adaptive_avg_pool2d
from scipy.linalg import sqrtm
import random

CUDA = False
DATA_PATH = './data'
BATCH_SIZE = 128
IMAGE_CHANNEL = 1
Z_DIM = 100
G_HIDDEN = 64
X_DIM = 64
D_HIDDEN = 64
EPOCH_NUM = 5
REAL_LABEL = 1
FAKE_LABEL = 0
```



```

class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            # input layer
            nn.ConvTranspose2d(Z_DIM, G_HIDDEN * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(G_HIDDEN * 8),
            nn.ReLU(True),

            # 1st hidden layer
            nn.ConvTranspose2d(G_HIDDEN * 8, G_HIDDEN * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(G_HIDDEN * 4),
            nn.ReLU(True),

            # 2nd hidden layer
            nn.ConvTranspose2d(G_HIDDEN * 4, G_HIDDEN * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(G_HIDDEN * 2),
            nn.ReLU(True),

            # 3rd hidden layer
            nn.ConvTranspose2d(G_HIDDEN * 2, G_HIDDEN, 4, 2, 1, bias=False),
            nn.BatchNorm2d(G_HIDDEN),
            nn.ReLU(True),

            # output layer
            nn.ConvTranspose2d(G_HIDDEN, IMAGE_CHANNEL, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, input):
        return self.main(input)

```

DROPOUT_RATE = 0.2

```

class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            # 1st layer
            nn.Conv2d(IMAGE_CHANNEL, D_HIDDEN, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # 2nd layer
            nn.Dropout2d(DROPOUT_RATE),
            nn.Conv2d(D_HIDDEN, D_HIDDEN * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(D_HIDDEN * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # 3rd layer
            nn.Dropout2d(DROPOUT_RATE),
            nn.Conv2d(D_HIDDEN * 2, D_HIDDEN * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(D_HIDDEN * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # 4th layer
            nn.Dropout2d(DROPOUT_RATE),
            nn.Conv2d(D_HIDDEN * 4, D_HIDDEN * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(D_HIDDEN * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # output layer
            nn.Dropout2d(DROPOUT_RATE),
            nn.Conv2d(D_HIDDEN * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input).view(-1, 1).squeeze(1)

def real_labels(b_size):
    label = torch.full((b_size,), REAL_LABEL, dtype=torch.float, device=device)
    return label

```

```
def fake_labels(b_size):
    label = torch.full((b_size,), FAKE_LABEL, dtype=torch.float, device=device)
    return label
```

```
Population = List[Generator]
```

```
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)
```

```
def generate_genome(g: Generator) -> Generator:
    g.apply(weights_init)
    return g
```

```
def generate_population(size: int) -> Population:
    return [generate_genome(Generator().to(device)) for _ in range(size)]
```

```
CROSS_RATE = 0.7
```

```
DISTILLATION_RANGE = 250
```

```
def single_point_crossover(a: Generator, b: Generator, d: Discriminator, b_size) -> Generator:
    noise = torch.randn(b_size, Z_DIM, 1, 1, device=device)

    fake_a = a(noise)
    fake_b = b(noise)
```

```
output_a = d(fake_a.detach()).view(-1)
```

```
output_b = d(fake_b.detach()).view(-1)
```

```
if np.random.rand() < CROSS_RATE:
```

```
    mse_loss = nn.MSELoss()
```

```
    child = Generator().to(device)
```

```
    child.apply(weights_init)
```

```
    if output_a.mean() < output_b.mean():
```

```
        child.load_state_dict(b.state_dict())
```

```
    else:
```

```
        child.load_state_dict(a.state_dict())
```

```
    iter_a = 0
```

```
    iter_b = 0
```

```
    child_optimizer = optim.Adam(child.parameters(), lr=2e-4, betas=(0.5, 0.999))
```

```
    for i in range(DISTILLATION_RANGE):
```

```
        child.zero_grad()
```

```
        noise_tmp = torch.randn(1, Z_DIM, 1, 1, device=device)
```

```
        fake_child = child(noise_tmp)
```

```
        fake_a = a(noise_tmp)
```

```
        fake_b = b(noise_tmp)
```

```
        output_a = d(fake_a.detach()).view(-1)
```

```
        output_b = d(fake_b.detach()).view(-1)
```

```
        if output_a.mean() < output_b.mean():
```

```
            child_err = mse_loss(fake_child, fake_b)
```

```
            child_err.backward()
```

```

        iter_b += 1

    else:
        child_err = mse_loss(fake_child, fake_a)
        child_err.backward()
        iter_a += 1

    child_optimizer.step()

    print(child_err, iter_a, iter_b)

else:
    if output_a.mean() < output_b.mean():
        return b
    else:
        return a

return child

MUTATION_RATE = 0.25
MUTATION_FACTOR = 0.003

def mutation(g: Generator, device) -> Generator:
    with torch.no_grad():
        for module in (g.modules()):
            for param in module.parameters():
                shape = param.data.shape
                mut_div = nn.Dropout(MUTATION_RATE)(torch.ones(shape, device=device) - 0.5) *
torch.randn(
                shape, device=device) * MUTATION_FACTOR
                param.data += mut_div
    return g

```

```

def sort_population(p: Population, loss_list) -> Population:
    seq = list(range(0, len(loss_list)))
    los_dict = dict(zip(seq, loss_list))
    los_dict_sorted = dict(sorted(los_dict.items(), key=lambda x:x[1]))
    result = []
    for key in los_dict_sorted:
        result.append(p[key])

    return result

```

```

def selection_function(p: Population, loss_list) -> Population:
    loss_list_sum = sum(loss_list)
    weights = []
    for l in loss_list:
        w_tmp = l / loss_list_sum
        weights.append(1.0 - w_tmp)
    ch = random.choices(p, weights, k=2)
    while ch[0] == ch[1]:
        ch = random.choices(p, weights, k=2)

    return ch

```

```

def main():
    netD = Discriminator().to(device)
    netD.apply(weights_init)
    print(netD)

    # Initialize BCELoss function
    criterion = nn.BCELoss()
    mae = nn.L1Loss()
    #sigmoid = nn.Sigmoid()

```

```

        optimizerD = optim.Adam(netD.parameters(), lr=D_lr, betas=(BETA1, BETA2),
weight_decay=WEIGHT_DECAY)

G_losses = []
D_losses = []

population = generate_population(POPULATION_SIZE)
optimizer_list = []

for g in population:
    optimizer = optim.Adam(g.parameters(), lr=G_lr, betas=(BETA1, BETA2),
weight_decay=WEIGHT_DECAY)
    optimizer_list.append(optimizer)

viz_noise = torch.rand(BATCH_SIZE, Z_DIM, 1, 1, device=device)
population_loss = [0 for _ in range(POPULATION_SIZE)]

print("Starting Training Loop...")
for epoch in range(EPOCH_NUM):
    for i, data in enumerate(dataloader, 0):
        real_cpu = data[0].to(device)
        b_size = real_cpu.size(0)

        netD.zero_grad()

        outputD_real = netD(real_cpu)
        errD_real = criterion(outputD_real, real_labels(b_size))
        errD_real.backward()

        errD_fake_sum = torch.tensor([0.0], dtype=torch.float32, device=device)
        fake_list = []
        for k, g in enumerate(population):
            noise = torch.randn(b_size, Z_DIM, 1, 1, device=device)
            fake = g(noise)

```

```

fake_list.append(fake)

outputD_fake = netD(fake.detach())
errD_fake = criterion(outputD_fake, fake_labels(b_size))
errD_fake.backward()

errD_fake_sum += errD_fake

errD = errD_real + errD_fake_sum
lossD_avg += errD.item() / GA_EPOCH

optimizerD.step()

if i % N_CRITIC == 0:
    batch_loss = []
    for k, g in enumerate(population):
        g.zero_grad()

        fake = fake_list[k]

        output = netD(fake).view(-1)
        errG = criterion(output, real_labels(b_size))
        errG.backward()
        population_loss[k] = errG.item()

    batch_loss.append(errG.item())

    for o in optimizer_list:
        o.step()

if i % 10 == 0:
    for idx, g in enumerate(population):
        plt.subplot(4, 5, idx + 1)

```

```

plt.axis("off")
plt.title("Fake Images")
plt.imshow(
    np.transpose(
        vutils.make_grid(g(viz_noise).detach().cpu(), padding=2, normalize=True)))
plt.show()

```

if $i \% \text{GA_EPOCH} == 0$ and $i > 0$:

for g in population:

noise1 = torch.randn(b_size, Z_DIM, 1, 1, device=device)

noise2 = torch.randn(b_size, Z_DIM, 1, 1, device=device)

fake1 = g(noise1)

fake2 = g(noise2)

$r = \text{mae}(\text{fake1}, \text{fake2})$

print("MAE: ", r.item())

population = sort_population(population, population_loss)

population_loss.sort()

next_gen_num = 0

next_generation = population[0:next_gen_num]

for j in range($\text{POPULATION_SIZE} - \text{next_gen_num}$):

parents = selection_function(population[0:int($\text{POPULATION_SIZE} * 1.0$)],

population_loss[0:int($\text{POPULATION_SIZE} * 1.0$)])

offspring_a = single_point_crossover(parents[0], parents[1], netD, b_size)

next_generation.append(offspring_a)

population = copy.deepcopy(next_generation)

```

del next_generation

optimizer_list = []
for g in population:
    optimizer_list.append(optim.Adam(g.parameters(), lr=G_lr, betas=(0.5, 0.999)))

population_loss = [0 for _ in range(POPULATION_SIZE)]

if i % GA_EPOCH == 0:
    print(lossD_avg)
    lossD_avg = 0

D_losses.append(errD.item())
if i % 1 == 0:
    print(
        f"[{epoch}/{EPOCH_NUM}] Batch {i}/{len(dataloader)} \
        Loss D: {errD.item():.4f}, Loss G_min: {min(batch_loss):.4f}      Loss G_max:
        {max(batch_loss):.4f} {batch_loss}"

    )
for g in population:
    real_batch = next(iter(dataloader))

    # Plot the real images
    plt.figure(figsize=(15, 15))
    plt.subplot(1, 2, 1)
    plt.axis("off")
    plt.title("Real Images")
    plt.imshow(
        np.transpose(vutils.make_grid(real_batch[0][:64], padding=5, normalize=True).cpu(), (1,
2, 0)))

    # Plot the fake images from the last epoch

```

```
fake = g(viz_noise).detach().cpu()
plt.subplot(1, 2, 2)
plt.axis("off")
plt.title("Fake Images")
plt.imshow(np.transpose(vutils.make_grid(fake, padding=2, normalize=True), (1, 2, 0)))
plt.show()
```

Press the green button in the gutter to run the script.

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