КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 113 Прикладна математика
на тему:

ДОСЛІДЖЕННЯ МЕТОДІВ ТА АЛГОРИТМІВ ГЕНЕРУВАННЯ СЦЕН

Студента 4-го курсу
кафедри обчислювальної математики
Кляцького Дениса Олеговича
_____

Науковий керівник:
асистент
Денисов Сергій Вікторович
_____
«____» _____ 2021 р.

Роботу розглянуто й допущено до захисту на засіданні кафедри обчислювальної математики «____»
_____ 2021 р., протокол № ___
Завідувач кафедри Ляшко С. І.
_____

Київ – 2021

Зміст

Вступ

Без сумнівів можна казати, що в загальному випадку потреби інформаційної індустрії зростають з розвитком науки та техніки, проте інколи дійсне й протилежне – певні технології з'являються для того, щоб вирішити конкретні задачі.

Однією з таких задач є створення нових об'єктів за довільним чи заданим набором вхідних параметрів. Простіше розібрати це на прикладах з реального життя, бо з певними задачами ми стикаємося доволі часто, навіть не усвідомлюючи цього.

Перш за все, слід зазначити, що під створенням нових об'єктів, мається на увазі якісний результат, який може бути використаний для сучасних потреб. Тож, згенерований шум чи хаотичний набір пікселів складно використати з користю, хоча інколи необхідні саме невпорядковані дані (наприклад, генерація текстур для графічних об'єктів).

Найбільш широко програмну генерацію об'єктів використовують в процесах, де присутня людино-машинна взаємодія. Яскравим прикладом можуть бути певні алгоритми створення капчі (особливих зображень для розпізнавання людської присутності в веб-застосунках) або мобільні додатки, що використовують одне фото для створення купи нових з різними ефектами.

Можна зазначити, що доволі широкий підхід до створення сцен та персонажів застосовує галузь виробництва мобільних та комп'ютерних ігор, а також певні підрозділи кіноіндустрії.

Таким чином сьогодення пропонує доволі багато варіантів застосування алгоритмів штучного створення нових об'єктів, що цікаві своїми особливостями застосування та характерними обмеженнями.

Метою даної роботи є дослідження та порівняння підходів до створення таких штучних зображень та об'єктів, що могли б бути корисно використані для потреб людської діяльності, а також способи їхньої оптимізації для більш широкого застосування, наприклад, за умов обмежених апаратних та програмних ресурсів.

Розділ 1

1.1 Актуальність теми в умовах сучасних потреб ІТ

Зображення та похідні медіа-формати мають вагоме місце в сучасній культурі, тому що інформаційний простір постійно розширюється та потребує нових даних. Саме проблема генерації нових медіа-об’єктів постає більш суттєво з-за того, що за рахунок автоматизації процесів генерації нових даних можна зменшити частку людської праці, а може навіть прискорити виробництво нових цифрових продуктів.

В цьому може стати в нагоді машинне навчання на основі генеративних моделей. Їхнє головне призначення – саме генерація нових об’єктів на відміну від дискримінаційних моделей, які здатні лише розрізняти різні класи об’єктів. Цей напрямок цікавий ще й тому, що він набув поширення відносно нещодавно та є досить перспективним як для вивчення та модифікації, так і для кінцевого застосування в контексті вирішення прикладних задач. Проте щоб краще зрозуміти особливості генеративних моделей, слід оглянути класичні підходи до вирішення задач за допомогою методів машинного навчання.

Перспектива розвитку методів глибокого машинного навчання полягає у вивченні та подальшому ефективному застосуванні ієрархічних моделей, які представляють розподіл ймовірностей щодо видів даних, які зустрічаються в програмах штучного інтелекту. Мова йде про зображення, звукові форми, що містять мову, та символи в контексті зображень.

Подібні ієрархічні моделі можна застосовувати як для вирішення задач класифікації, що і визначає математичний зміст розподілів ймовірностей щодо видів даних, так і навпаки - для генерації нових об’єктів на основі факту належності їх до певного класу. Іншими словами, розподіл ймовірностей визначає кількість та типи класів об’єктів, з якими може працювати нейронна мережа, - в загальному розумінні це саме ті особливості,

які вирізняють одні об’єкти від інших (коти і собаки, велосипеди та автобуси, тощо).

Наразі найяскравіші успіхи у глибокому навчанні належать дискримінаційним моделям, як правило, тим, які накладають багатовимірний сенсорний вхід на позначення класу. Саме цей багатовимірний сенсорний вхід визначає кількість та тип вхідних даних, що визначають певний об’єкт. Іншими словами, це кількість унікальних якостей, що в сукупності визначають об’єкт та його належність до певного класу.

Ці вражаючі успіхи, в першу чергу, базувались на алгоритмах зворотного розповсюдження, що полягає в поширенні сигналів помилки від виходів мережі до її входів, у напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи, та відсіву, який передбачає випадкове виключення або «випадання» певних результатів шару нейронної мережі на етапі навчання, щоб уникнути явища надмірного навчання.

Ці алгоритми, в свою чергу, використовують математичні принципи, за яких для вирішення поставлених задач застосовуються гладкі функції. Застосування функцій з гладкими градієнтами значно прискорює пошук мінімума чи максимума функції, що дозволяє значно оптимізувати вирішення задач машинного навчання.

Моделі, що застосовують глибші алгоритми, мають менший вплив на методи машинного навчання через труднощі наближення багатьох нерозв'язних імовірнісних обчислень, які виникають за оцінки максимальної вірогідності та пов'язаних із ними стратегій, а також через труднощі використання переваг кусково-лінійних функцій саме для генерації нових даних.

Відповідно, подібні моделі складно налаштовувати, тому як результат плідної роботи багатьох науковців та ентузіастів, з’явився такий підхід до вирішення задач генерації нових даних, як генеративні моделі або GAN (англ. Generative adversarial network). Їхня перевага в відносно простих алгоритмах та можливостях модифікації для спеціалізованих досліджень чи

задач.Переваги нейронні мереж, на відміну від статистичних методів багатовимірного класифікаційного аналізу, базуються на паралельній обробці інформації і мають здатність до самонавчання, тобто отримувати обгрунтований результат на підставі даних, які не зустрічались у процесі навчання.

Ці властивості дозволяють нейронним мережам створювати складні програмні комплекси, в складі яких можуть бути не тільки одиничні мережі, але навіть кластери нейронних мереж. Це явище цікаво розглянути саме в контексті генеративно-змагальної моделі нейронної мережі.

Тож, в чому перевага генеративно-змагальної моделі нейронної мережі? У запропонованій GAN мережі, генеративна модель протиставляється супернику: недискримінаційна модель, яка навчається визначати, чи є вибірка з розподілу моделі чи розподілу даних. Генеративна модель може розглядатися як аналог групи підробників, яка намагається виробляти фальшиву валюту та використовувати її без виявлення, тоді як дискримінаційна модель аналогічна поліції, намагається виявити фальшиву валюту. Змагання в цій грі спонукає обидві команди вдосконалювати свої методи, доки підробки не будуть відрізнятися від справжніх взірців.

Хоча існують і інші генеративні моделі, які в певній мірі мали вплив на створення генеративно-змагальної моделі нейронної мережі, в цій роботі буде розглянуто саме GAN-мережі, тому що це є відносно «легка» архітектура нейронної мережі для генерації зображень, і відповідно, вона може бути розгорнута навіть на невеликих потужностях, що безумовно має свої переваги. Ці мережі можна застосовувати для багатьох прикладних задач без суттєвої зміни їхньої архітектури, тому і було обрано саме цей тип нейронних мереж для більш детального опрацювання.

1.2 Огляд технологій генерації сцен за допомогою методів машинного навчання

Загалом можна сказати, що задача генерації нових медіа-об’єктів на основі певної вибірки пов’язана напряму з задачею класифікації об’єктів. Основна концепція такого підходу дозволяє генерувати незчисленну кількість нових об’єктів та відбирати з них лише ті, що належать до певного класу.

Для того, щоб краще зрозуміти кожен з наведених нижче методів машинного навчання, а також комплексно уявляти процеси, які супроводжують кожну технологію, слід почати з загальної характеристики цілого класу генеративних та моделей.

Генеративне моделювання - це завдання машинного навчання без вчителя (неконтрольоване навчання), яка полягає в автоматичному виявленні закономірностей і залежностей у вхідних даних, які можна було б використовувати для генерації на виході нових прикладів, які могли б бути несуперечливими відносно присутнім в оригінальному (вихідному) наборі даних.

Генеративні моделі (англ. Generative model) - це клас моделей, які застосовують спільний розподіл даних $p(x, y)$; звідси легко отримати умовний розподіл $p(x) = \frac{p(x,y)}{p(x)}$, проте спільне дає більше інформації і його можна використовувати, наприклад, для генерації нових фотографій тварин, які виглядають як справжні тварини. В контексті даних формул x є, власне, об’єкт (вхідні дані), а y – відповідно, маркер належності до певного класу. З іншого боку, дискримінаційна модель (англ. Discriminative model) застосовує тільки умовний розподіл і може, наприклад, відрізнити собаку від кішки.

Можна використовувати деякі емпіричні правила для генерації нових об'єктів, не використовуючи машинного навчання. Задля того, щоб навчитися створювати правдоподібний об'єкт з деякої прихованої структури вихідних об'єктів, слід вивчити їх розподіл, а потім генерувати новий об'єкт з цього

розподілу. Відповідно, задача відноситься до класу задач навчання без учителя.

Генеративна модель іноді дозволяє використовувати навчання з частковим залученням вчителя. Наприклад, завдання полягає в тому, щоб відрізнити кішок від собак на фотографіях, як було зазначено раніше. Зазвичай недостатньо просто розмічених даних, на яких кішки і собаки класифіковані вручну. Основна частина завдання полягає в тому, щоб зрозуміти, чим корисні фотографії відрізняються від випадкового шуму. Інакше кажучи, якщо спочатку визначити розподіл $p(x)$, то простіше навчити розподіл $p(x) = \frac{p(x,y)}{p(x)}$, де y - це один біт, що відповідає за окрему ознаку, а x - це вся фотографія.

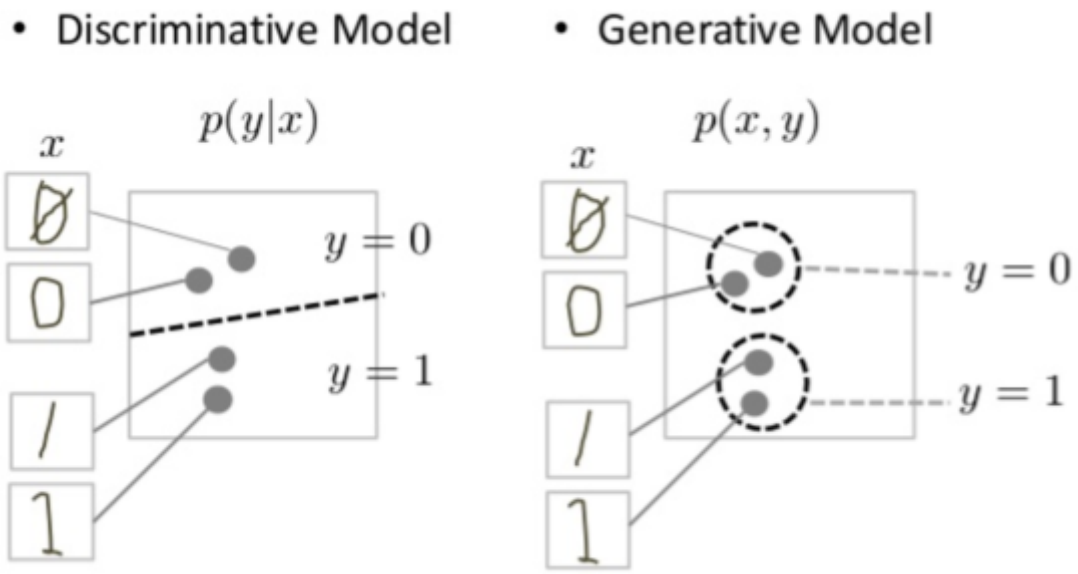

Рис 1. Схема алгоритму роботи генеративно-змагальної нейронної мережі. Генеративна модель намагається генерувати рукописні 0 і 1, для цього моделює розподіл по всьому простору даних. Навпаки, дискримінаційна модель намагається розділити дані, без необхідності точно моделювати, як об'єкти розміщуються по обидві сторони від лінії.

Більш детально генеративна та дискримінаційна моделі будуть розглянуті в відповідних пунктах роботи.

Тож, існує цілий клас моделей, які слугують для генерації нових даних з певної вибірки, до яких класично відносять наступні підходи:

- наївний байєсівський класифікатор;
- автокодери та варіаційні автокод ери (VAE);
- генеративні мережі на основі суперництва (GAN).

Додатково слід зазначити, що корисно розглянути одну з найпопулярніших дискримінаційних моделей – згорткову нейронну мережу, бо це загалом альтернативний підхід до роботи з зображеннями, що досягається за допомогою технології «bottleneck» та є досить зручним для застосування в контексті вирішення поточної задачі.

Слід розуміти, що в кожного з вищенаведених підходів свої межі застосування, тому в рамках даної роботи було прийнято розглядати генеративні та дискримінаційні моделі саме в контексті генерації зображень, тому всі наступні твердження справедливі для конкретного класу моделей, що орієнтовані на роботу з графічними об’єктами.

Всі вищезазначені підходи, окрім байєсівського класифікатора, будуть детальніше розглянуті в наступних пунктах роботи.

1.2.1 Структура та призначення згорткових нейронних мереж

Архітектура даного типу нейронної мережі має дозволяти виокремлення конкретних локальних трендів для кожного зображення для проведення класифікації за загальними для класу особливостями. Нейронні мережі такого типу створені для опрацювання зображень та мають класичну архітектуру: шари згортання – їх задача формувати матрицю ваг для зображень, щоб виокремити характерні риси для цього класу; шари максимізації для виведення середніх ознак для даного набору, тобто зведення ознак класу до найпростішого вигляду; повнозв’язні шари, власне, виконують класифікацію та формують прогноз. Таким чином для нейронної мережі

прямого розповсюдження досягається найбільша точність та найменша ресурсоємність.

Загальний алгоритм, на основі якого працює даний вид нейронних мереж поданий нижче.

Згортка (англ. Convolution) - операція над парою матриць A (розміру $n_x \times n_y$) і B (розміру $m_x \times m_y$), результатом якої є матриця $C = A * B$ розміру $(n_x - m_x + 1) \times (n_y - m_y + 1)$. Кожен елемент результату обчислюється як скалярний добуток матриці B і деякої підматриці A такого ж розміру (підматриця визначається положенням елемента в результаті). Тобто,

$$C_{i,j} = \sum_{u=0}^{m_x-1} \sum_{v=0}^{m_y-1} A_{i+u, j+v} B_{u,v}$$

Загальний принцип цього методу базується на тому, що матриця B «рухається» по матриці A, і в кожному положенні знаходиться скалярний добуток матриці B і тієї частини матриці A, на яку вона зараз накладена.

Число, що отримане цими діями, записується у відповідний елемент результату. Логічний сенс згортки полягає в наступному: чим більша величина елемента згортки, тим більше ця частина матриці A схожа на матрицю B (схожа в контексті скалярного добутку). Тому матрицю A називають зображенням, а матрицю B - фільтром або зразком.

Застосування згорткових нейронних мереж дозволило вирішити певний клас задач класифікації, що в значній мірі зменшило витрати людських ресурсів та дозволило автоматизувати ці процеси.

Даний підхід розглядається в контексті генерування нових зображень для певного класу з тої причини, що зазвичай дискримінаційна модель, зазначена в загальній концепції методів цієї галузі машинного навчання, представлена саме нейронною мережею згортки, бо вона дозволяє відносно оптимізовано проводити аналіз відповідності згенерованого зображення.

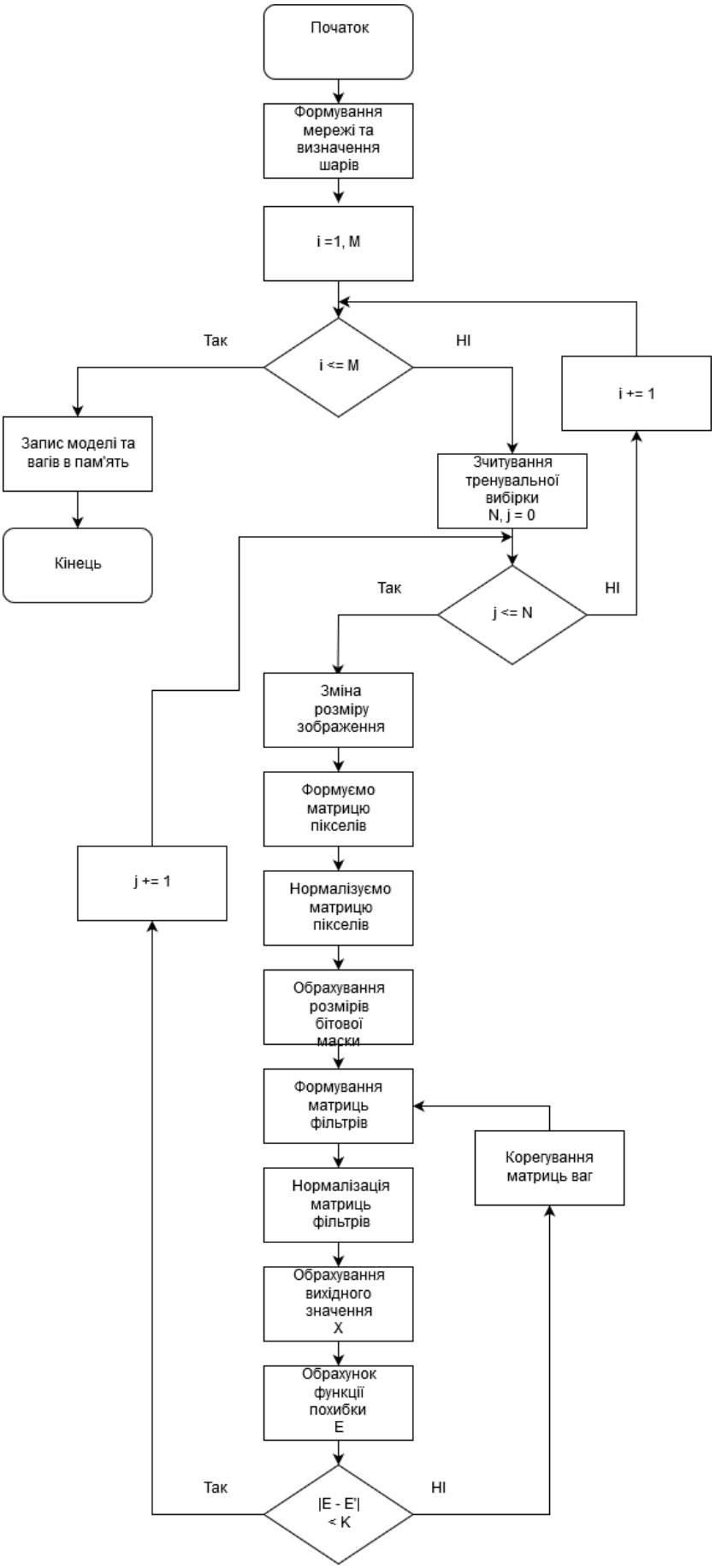

Рис 2. Блок-схема алгоритму навчання нейронної мережі згортки

1.2.2 Автокодери та варіаційні автокодери (VAE)

Автокодер (англ. Autoencoder) - спеціальна архітектура штучних нейронних мереж, що дозволяє застосовувати навчання без вчителя з використанням методу зворотного поширення помилки. Найпростіша архітектура автокодера - мережа прямого поширення, без зворотного зв'язку, найбільш схожа з перцептроном і містить вхідний шар, проміжний шар і вихідний шар. На відміну від перцептрону, вихідний шар автокодеру повинен містити стільки ж нейронів, скільки і вхідний шар.

Автокодер складається з двох частин: енкодера g і декодера f. Енкодер переводить вхідний сигнал в його уявлення (код): $h = g(x)$, а декодер відновлює сигнал за його кодом: $x = f(h)$.

Автокодер, змінюючи f і g, прагне дослідити тотожну функцію $x = f(g(x))$, мінімізуючи певний функціонал помилки.

$$L(x, f(g(x)))$$

При цьому сімейства функцій енкодера g і декодера f обмежені в певній мірі для того, щоб автокодер був змушений відбирати найбільш важливі властивості сигналу.

Автокодер можна використовувати для попереднього навчання, наприклад, коли постає завдання класифікації, а розмічених пар занадто мало. Інший випадок – це зниження розмірності даних для подальшої візуалізації. Або коли просто треба навчитися розрізняти корисні властивості вхідного сигналу.

Іншими словами автокодер зменшує розмірність задля зручного представлення вхідних даних та використовує кодовану інформацію для генерації та перевірки нових даних. Однак при такій конфігурації існує ряд недоліків.

При спробі використання звичайного автокодера для генерації нових об'єктів (бажано з того ж апріорного розподілу, що і вихідні дані) виникає наступна проблема. Випадковою величиною з яким розподілом слід

проініціалізувати приховані вектори, для того, щоб зображення, після застосування декодера, стало схоже на зображення з набору для навчання, але при цьому не збігалося ні з одним з них?

Відповідь на це питання не очевидна в зв'язку з тим, що звичайний автокодер не може нічого стверджувати про розподіл прихованого вектора і навіть про його область визначення. Зокрема, область визначення може бути навіть дискретною. Варіаційний автокодер, в свою чергу, пропонує користувачеві самому визначити розподіл прихованого вектора. В цьому і полягає головна відмінність варіаційного підходу від звичайного – більш оптимізований та складний алгоритм генерації та перевірки згенерованих даних.

Генеративне моделювання зазвичай має справу з розподілом $P(W)$, визначеному на вихідних даних W з простору (можливо багатовимірного) W. Так, наприклад, популярні завдання генерації зображень мають справу з величезною кількістю розмірностей (пікселів).

Так само, як і в випадку використання звичайних автокодерів, існує прихований імовірнісний простір Z, що відповідає випадковій величині $(z,\ P(z))$, що розподілена фіксовано $(\sim N\ (0, 1))$. За таких умов декодер:

$$f\ (z,\ \theta)\colon\ Z \times \Theta \to W.$$

При цьому ми хочемо знайти такі θ, щоб після представлення z по $P(z)$ ми отримали відносну схожість з елементами W.

Іншими словами, знаходження θ є пошуком ваг, за яких випадково підібрані параметри відповідали оригінальним ознакам вихідного класу об’єктів.

Взагалі, для будь-якого $x \in W$ вважаємо:

$$P(x) = \int P(z;\ \theta)P(z)dz,$$

тут ми замінили $f(z,\ \theta)$ на $P(x|z;\ \theta)$, щоб явно показати залежність між x і z та після цього застосувати формулу повної ймовірності. Зазвичай $P(x|z;\ \theta)$ є близькою до нуля майже для всіх пар $(x,\ z)$. Основна ідея полягає

в тому, щоб генерувати z, який би був близьким до x та їх вже підсумовувати в $P(x)$. Для цього нам потрібно ввести ще один розподіл $Q(z|W)$, який буде отримувати x і визначати розподіл z, яке найбільш ймовірно буде генерувати нам такий x. Наразі постає необхідність зробити схожими розподіли $E_{z\sim Q}P(W|z)$ і $P(W)$.

Для спрощення вищезазначених розподілів зручно використовувати дивергенцію Кульбака-Лейблера. Задля кращого розуміння необхідності використання цього підходу слід знати, що він застосовується для мінімізації складних розподілів до більш простих та, відповідно, більш зручних у прикладних задачах. Практично це зазвичай значить укрупнення моделей, що досягається за рахунок пошуку найбільш суттєвих ознак об’єктів одного класу та виключення з розподілу менш вагомих ознак. Тобто об’єкти стають більш простими не за своїм змістом, а за наявністю вагомих ознак, що використовуються при визначенні розподілу.

Після застосування вищезазначеного підходу стає можливим використання принципу зворотнього розповсюдження помилки.

Зворотне поширення помилки - це систематичний метод для навчання багатошарових штучних нейронних мереж. У процесі навчання багатошарового персептрона із застосуванням алгоритму зворотного поширення помилки йому багаторазово пред'являється безліч навчальних прикладів. Один повний цикл пред'явлення повного набору прикладів навчання називають епохою. Процес навчання проводиться від епохи до епохи, поки синаптичні ваги і рівні порога не стабілізуються, а середньоквадратична помилка на всій навчальній множині не зійдеться до деякого мінімального значення. Доцільно випадковим чином змінювати порядок подання прикладів навчання для різних епох. Такий принцип пред'явлення образів робить пошук в просторі ваг стохастичним, запобігаючи потенційної можливості появи замкнутих циклів в процесі еволюції синаптичних ваг.

Загалом, можна сказати, що розуміння процесів, які відбуваються всередині даного класу генеративних моделей, дає змогу порівнювати не тільки результати роботи різних типів моделей, але й звертати увагу на використання апаратних та програмних ресурсів, що може допомогти більш доцільно використовувати ці інструменти для якісного виконання поставлених задач.

Наразі можна чітко зазначити головну відмінність автокодерів від мереж згортки, що розглядалися в попередньому пункті: вона полягає саме в можливості генерувати нові об’єкти, а не лише виконувати задачу класифікації.

Розглянуту технологію зручно використовувати для опрацювання великих однотипних об’ємів даних та подальшої генерації відносно простих даних. Область застосування варіаційних автокодерів збігається з областю застосування звичайних автокодерів, а саме:

- каскадне навчання глибоких мереж (хоча зараз застосовується все рідше, в зв'язку з появою нових методів ініціалізації ваг);
- зменшення шуму в даних;
- зменшення розмірності даних (іноді працює краще, ніж метод головних компонент).

Завдяки тому, що користувач сам встановлює потрібний розподіл прихованого вектора, варіаційний автокодер добре підходить для генерації нових об'єктів (наприклад, зображень). Для цього досить розрахувати прихований вектор згідно його розподілу і подати на вхід декодера. Результатом буде об'єкт з того ж розподілу, що і вихідні дані.

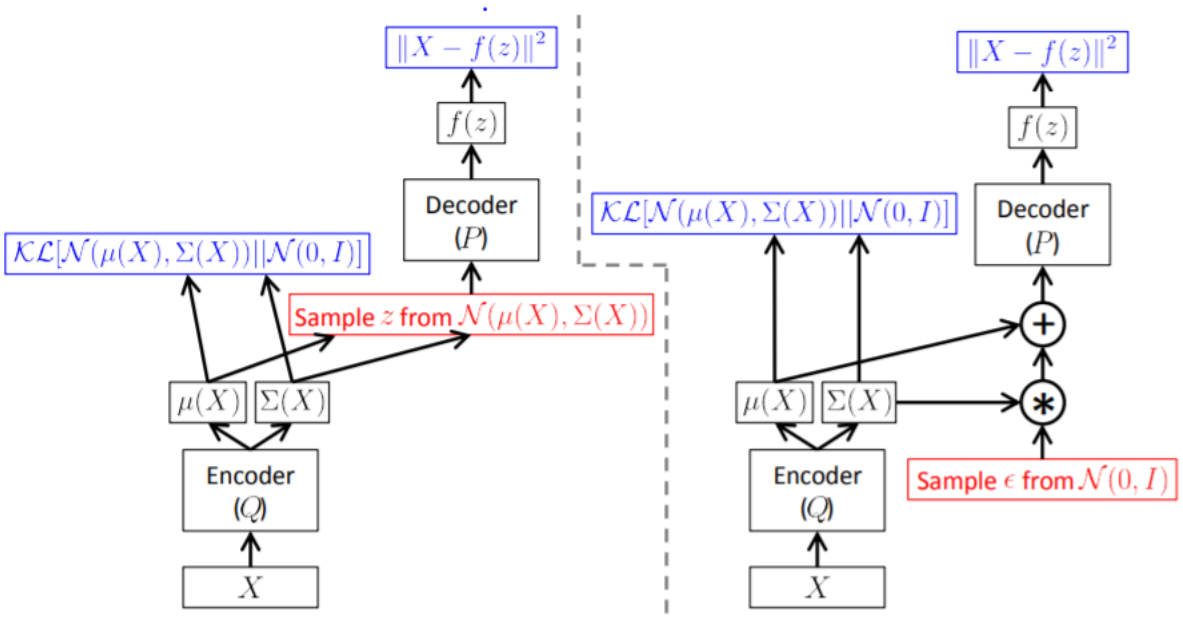

Рис 3. Принципова схема алгоритму роботи варіаційного автокодера

1.2.3 Генеративні нейронні мережі на основі суперництва (GAN)

На початку даного пункту варто зазначити, що нижче розглянутий саме класичний алгоритм навчання генеративно-змагальної мережі, котрий після публікації був значно допрацьований та оптимізований, тому слід розглядати даний пункт роботи як введення в принципи роботи нейронних мереж даного типу.

Генеративні нейронні мережі на основі суперництва (англ. Generative Adversarial Nets, GAN) - алгоритм машинного навчання, що входить в сімейство генеративних моделей і побудований на комбінації з двох нейронних мереж: генеративна модель G, яка будує наближення розподілу даних, і дискримінаційна модель D, що оцінює ймовірність того, що зразок прийшов з тренувальних даних, а не згенерованих моделлю G. Навчання для моделі G полягає в максимізації ймовірності помилки дискримінатора D. Вперше такі мережі були представлені Єном Гудфеллоу в 2014 році.

Як було зазначено раніше в описі методу, задача полягає в навчанні двох моделей: генеративної і дискримінаційної. Оскільки, найзручніше використовувати багатошарові перцептрони для навчання змагальної моделі, використовують саме їх для детального опису роботи моделі.

Щоб вивести імовірнісний розподіл генератора p_g з набору даних W, визначимо апріорну ймовірність шуму $p_z(z)$ і представимо генератор, як відображення $G(z, \gamma_g)$, де G - диференційована функція, представлена багатошаровим перцептроном з параметром γ_g.

Аналогічним чином представлено другий багатошаровий перцептрон $D(z, \gamma_d)$, який на вихід подає одне скалярне значення - ймовірність того, що x прийшло з тренувальних даних, а не з p_g. Під час тренування D ставиться на меті максимізувати ймовірність правильної ідентифікації об'єктів з тренувальної і згенерованої вибірок. І в той же час G тренується так, щоб мінімізувати $log(1 - D(G(z)))$: іншими словами, D і G умовно грають в "мінімакс гру":

$$V(D, G) = E_{x \sim pdata}[logD(x)] + E_{z \sim pz}[log(1 - D(G(z)))]$$

Можна уявити, що генеративні змагальні мережі навчаються шляхом одночасного поновлення дискримінаційного розподілу, так щоб дискримінатор міг розрізняти об'єкти з розподілу тренувального набору і з розподілу генератора. Після кількох кроків навчання G і D прийдуть в стан, в якому не зможуть покращитися, так як буде виконуватися умова $p_g = p_{data}$ і дискримінатор не зможе розрізняти два розподіли, тому його вихід завжди буде $D(x) = \frac{1}{2}$.

У процесі навчання потрібно робити два кроки оптимізації по черзі: спочатку оновлювати ваги генератора γ_g при фіксованому γ_d, а потім ваги дискримінатора γ_d при фіксованому γ_g. На практиці дискримінатор оновлюється k раз замість одного, оскільки, повністю оптимізувати

дискримінатор обчислювально не вигідно і на кінцевих вибірках він може перенавчитися. В даному контексті k чисельно відповідає кількості об’єктів, що подаються на етапі навчання в рамках однієї епохи. Тобто фактично за розміру підвибірки для навчання у 8 зображень, дискримінатор оновиться саме стільки разів, що прискорює загальну швидкість навчання, бо зникає потреба два рази оновлювати фіксовані параметри, а як наслідок, дискримінатор оновлюється після проходу кожного зображення. Таким чином k є гіперпараметром.

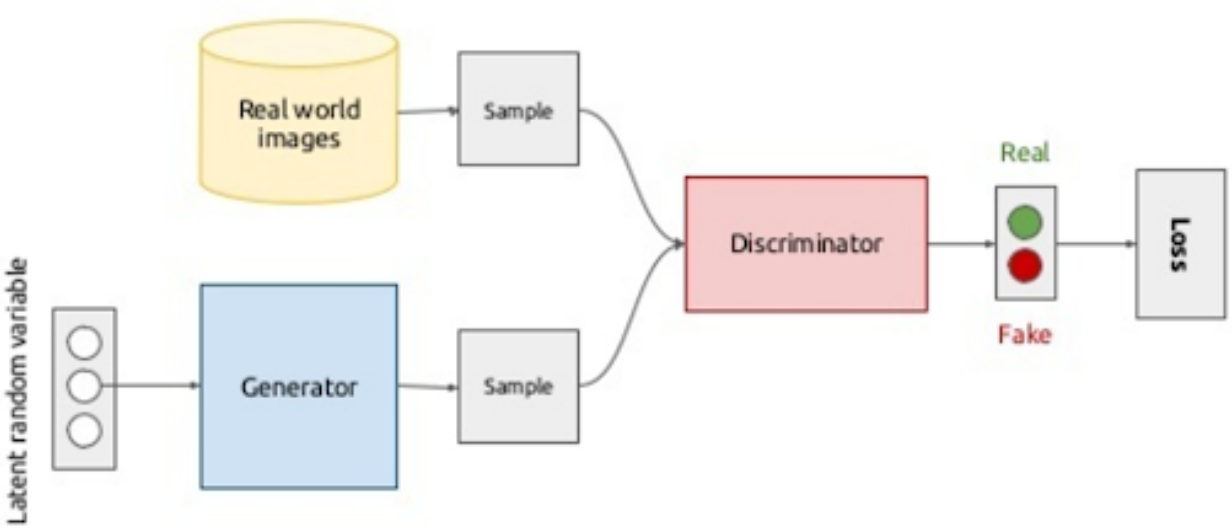

Рис 4. Принципова схема алгоритму роботи GAN-мережі

На практиці не завжди зручно використовувати рівняння описані вище. На початку навчання, коли G погано налаштований, дискримінатор D може не враховувати об'єкти, з високою впевненістю в класифікації, так як вони сильно відрізняються від тренувального сету, в такому випадку $log(1 - D(G(z)))$ стагнує. Щоб уникнути цього, можна замість мінімізації $log(1 - D(G(z)))$ максимізувати $logD(G(z))$.

Більшість моделей даного класу стикається з наступними проблемами:

- Схлопування мод розподілу (англ. Mode collapse): генератор колапсує, тобто видає обмежену кількість різних зразків.

- Проблема стабільності навчання (англ. Non-convergence): параметри моделі дестабілізуються і не сходяться.
- Зникаючий градієнт (англ. Diminished gradient): дискримінатор стає занадто "сильним", а градієнт генератора зникає і навчання не відбувається.
- Проблема заплутування (англ. Disentanglement problem): виявлення кореляції в ознаках, не пов'язаних (слабо зв'язаних) в реальному світі. Висока чутливість до гіперпараметрів.

Слід зауважити, що універсального підходу до вирішення більшості з цих проблем немає. Проте існують практичні поради, які можуть допомогти при навчанні мереж, основними з них є:

- Нормалізація даних. Всі ознаки в діапазоні [-1; 1];
- Заміна функції помилки для G з $minlog(1 - D)$ на $maxlog(D)$, Тому що вихідний варіант має маленький градієнт на ранньому етапі навчання і великий градієнт при збіжності, а запропонований навпаки;
- Семплірування з багатовимірного нормального розподілу замість рівномірного;
- Використовувати нормалізаційні шари (наприклад, batch normalization або layer normalization) в G і D;
- Використовувати мітки для даних, якщо вони є, то є навчати діскрімінатор ще й класифікувати зразки.

Таким чином, можна виявити головні переваги генеративно-змагальних нейронних мереж над автокодерами та варіаційними автокодерами. А саме вирішується головна проблема їхніх алгоритмів – неможливість помічати менш очевидні тренди та ознаки. Тож, генеративно-змагальні моделі нейронних мереж мають більшу гнучкість у навчанні та, власне, генерації нових об’єктів.

1.3 Структура та зміст генеративно-змагальних нейронних мереж

Нескладно побачити, що певні ідеї даного класу генеративних моделей запозичені в більш простої та старшої моделі автокодерів, проте інші засади даного підходу мають суттєві переваги, що в основному визначають саме область використання.

По-перше, як було зазначено в попередньому пункті, модель структурно складається з двох підмоделей – генератора та дискримінатора, тобто хоча з одного боку загальний відсоток похибки зростає, така взаємодія на практиці дозволяє швидше знаходити необхідні параметри для оптимальної швидкості навчання та використовувати різні архітектури комп'ютерів приблизно однаково ефективно.

По-друге, з-за того, що кожна структурна ланка такої моделі є окремою нейронною мережею, стає можливим більш гнучке налаштування моделі, що дозволяє значно оптимізувати процеси, що протікають всередині моделі. Це також позитивно впливає як на швидкість навчання, так і на якість генерації, що є головною метою роботи даної моделі.

З цього підпункту можна виокремити ще й те, що точне налаштування підмоделей, застосовується для спеціалізованого використання моделей даного типу. Наприклад, існують варіації моделей на основі GAN, що працюють для забезпечення, так званого, ефекту «переносу рухів», тобто модель має медіа-файл (зазвичай відео чи анімацію) та цільове фото, яке вдається умовно «оживити» за рахунок накладання цільового набору на вихідні рухи з першоджерела (модель Everybody Dance Now дослідників з університету Берклі). Іншим прикладом може бути підклас GAN моделей, що використовуються для стилізації зображень (чи відео) за певним патерном. Це може виявлятися в переробці цільового фото таким чином, щоб в ньому вгадувався стиль відомого митця, або в накладанні певних текстур чи навіть цілих зображень на ділянки відео чи фото із максимальним збереженням реалізму (як технологія DeepFake).

По-третє, так як архітектура даної моделі забезпечується більш складними алгоритмами, ніж архітектура автокодерів та варіаційних автокодерів, а саме завдяки використанню нейронних мереж, стає можливою генерація достатньо складних та нетипових об’єктів за умови достатньої кількості даних для навчання.

Саме останній підпункт цікавий для подальшого дослідження, бо відкриває широкі можливості застосування моделей даного класу як в контексті спеціалізованої генерації об’єктів, так і для загального використання в якості класичного підходу для вирішення задач генерації.

Генеративно-змагальні мережі, або скорочено GAN, імплементують підхід до генеративного моделювання з використанням методів глибокого навчання, таких як згорткові нейронні мережі.

GAN є захоплююча і швидко мінлива область, яка передбачає створення генеративних моделей зі здатністю створювати реалістичні приклади для цілого ряду проблемних областей, особливо в таких завданнях обробки зображень, як трансформація зображень з літніх в зимові або день у ніч, а також у створенні реалістичних фотографій об'єктів, сцен і людей, які люди не могли б відрізнити від підроблених.

В даному пункті роботи буде також розглянуто альтернативне бачення задачі, що вирішують генеративні мережі на основі суперництва, тому що класичне представлення алгоритму має певні недоліки, що вирішуються застосуванням нетипових підходів до вирішення даної задачі.

Як і було зазначено раніше, в рамках даної роботи була поставлена ціль дослідження процесів оптимізації генеративних моделей для ширшого їх застосування в контексті обмежених апаратних та програмних ресурсів, тому в наступних пунктах буде наведена додаткова інформація, що висвітлює та пояснює один із найбільш широко застосовуваних підходів – екстраполяції ваг для забезпечення стабільності роботи моделі.

1.3.1 Призначення генератора та дискримінатора

В вищенаведених пунктах даної роботи було визначено, що структура даного класу генеративних моделей складається саме з генератора та дискримінатора. Так як вище вже було згадано певні алгоритми роботи GAN, далі будуть наведені уточнення та доповнення до вже зазначеного, що дозволить більш розгорнуто поглянути на структуру даної моделі.

У оригіналі статті, що презентує GAN, мета генеративних змагальних мереж формулюється як нуль-сума, де функція витрат дискримінатора від'ємна логарифмічна ймовірність завдання бінарної класифікації між реальними або фальшивими даними, що генеруються q_θ генератором,

$$L(\theta, \varphi) \text{ де } L(\theta, \varphi) \stackrel{\text{def}}{=} -E_{x\sim p}[logD\varphi(x)] - E_{x'\sim q}\theta[log(1 - D\varphi(x'))].$$

Однак Гудфелло (автор оригінального алгоритму генеративно-змагальної мережі) рекомендує використовувати на практиці інше формулювання, яке називається «ненаситним». Це формулювання представлено ненульовою сумою, де на меті стає спільна мінімізація.

За такого твердження виникає наступна теорема про оптимальність: за кожен прохід генератора, дискримінатор оновлюється таким чином, щоб досягати такого оптимального рівня, за якого стає можливим оновити загальний критерії їхньої «гри». Іншими словами, це й є той самий умовний підхід, за якого генератор та дискримінатор намагаються віднайти максимально подібний до оригінального класу об'єкт, та при цьому постійно навчаються, щоб відповідати один одному.

Головною перевагою такого альтернативного підходу є більш вигідні початкові умови, а також більша швидкість стабілізації ваг для генерації конкретного класу зображень. Це також цікаво з точки зору використання ресурсів: якщо згадати основні проблеми автокодерів та варіаційних автокодерів, то до них належить в тому числі і нестабільність початкових

умов та сильна залежність всього процесу генерації від них. Тож, генеративно-змагальна модель з «ненаситним» формулювання дозволяє уникнути цієї проблеми та зосередитися саме на процесі навчання та генерації.

Формулювання мінімаксу є теоретично зручним, оскільки багато праць з теорії ігор вивчає цю проблему та забезпечує гарантії існування рівноваги. Тим не менше, практичні роздуми спонукають розглянути іншу мету для кожного гравця, як сформульовано в альтернативному формулюванні.

У цьому випадку проблема двох гравців полягає у пошуку наступної рівноваги за Нешем:

$$\theta^* \in arg\, L_G(\theta, \varphi^*) \text{ та } \varphi^* \in arg\, LD\,(\theta^*, \varphi)\,.$$

Тільки тоді, коли $L_G = - L_D$, гра, яка називається нуль-сумою може бути сформульована як мінімакс-проблема. Тобто, як можна бачити, суть полягає не в знаходженні одного загального значення, за якого сума буде нульовою, як раз навпаки: вирішення мінімакс-проблеми полягає в знаходженні таких значень L_G та L_D, що їх різниця буде мінімальною, тобто прямувати до нуля. За таких умов процес оптимізації якісно змінюється та пришвидшується. Одним із важливих моментів, на який слід звернути увагу, є те, що дві задачі оптимізації в останньому формулюванні є парами і їх слід розглядати спільно з точки зору оптимізації.

Стандартні функції GAN не опуклі (тобто кожна функція витрат не опукла), і, отже, чистих рівноваг може не існувати.

Зі всього вищенаведеного можна виокремити кілька основних тез щодо призначення структурних елементів генеративно-змагальних моделей.

По-перше, це створення умов, за яких процес генерації зображень поліпшується на рівні з ускладненням перевірки результатів дискримінатором, що дозволяє моделі постійно самовдосконалюватися за умов достатньої кількості вхідних даних для навчання.

По-друге, на результат впливають параметри як генератора, так і дискримінатора, та, що є суттєвим, значно зменшується вплив початкових умов (ініціалізація ваг) на кінцевий результат.

1.3.2 Основні алгоритми оптимізації взаємодії генератора та дискримінатора

Як можна побачити, головною метою алгоритмів генератора є створення більш якісних зображень, а дискримінатора їх більш чіткий відбір з метою постійного поліпшення якості результату. Проте складно уявити за рахунок чого це досягається на практиці. В ході дослідження даної тези, було виявлено, що більш оптимізовані моделі вимагають менше ресурсів для створення якісних результатів, та, як результат, вони є більш пристосованими до різноманітних умов вхідних даних. З цього можна зробити висновок, що в більшості випадків оптимізація алгоритмів взаємодії генератора та дискримінатора дозволяє пришвидшити та якісно покращити процес навчання та генерації.

Далі наведені типові процеси, які виникають в GAN мережі:

- Генератор приймає випадкові числа і повертає зображення.
- Це сформоване зображення подається в дискримінатор поряд із потоком зображень, взятих із фактичного набору даних.
- Дискримінатор приймає як реальні, так і підроблені зображення і повертає ймовірності, число від 0 до 1, при цьому 1 представляє передбачення достовірності, а 0 - підроблене.

Отже, у існує подвійний цикл зворотного зв'язку:

- Дискримінатор знаходиться у циклі зворотного зв'язку із вихідним набором зображень, який був заданий.
- Генератор знаходиться в циклі зворотного зв'язку з дискримінатором.

Як вже згадувалося раніше, одна з труднощів, що виникає при оптимізації таких ігор генератора та дискримінатора, полягає в тому, що дві різні функції витрат повинні бути мінімізовані спільно. На щастя, дослідження з оптимізації протягом тривалого часу вивчали так звані проблеми варіаційної нерівності, які узагальнюють стаціонарні умови для ігрових задач для двох гравців.

Спочатку розглянемо місцеві необхідні умови, що характеризують рішення гладкої гри двох гравців, визначаючи стаціонарні моменти, які будуть впливати на визначення варіаційної нерівності. В загальному випадку стаціонарна точка - це пара (θ^*, φ^*) з нульовим градієнтом:

$$\|\nabla_\theta L_G(\theta^*, \varphi^*)\| = \|\nabla_\varphi L_D(\theta^*, \varphi^*)\| = 0.$$

Ці стаціонарні умови можна узагальнити на будь-яке безперервне векторне поле: нехай $\Omega \subset R^d$ та F: $\Omega \to R^d$ є неперервне відображення. Задача про варіаційну нерівність (залежно від F та Ω) полягає в знаходженні такого $\omega^* \in \Omega$, що $F(\omega^*)^T(\omega - \omega^*) \geq 0, \forall\omega \in \Omega$.

Називаємо оптимальним набір Ω^* від $\omega \in \Omega$, що задовольняє попередній вираз.

Зміст цього виразу полягає в тому, що будь-яке $\omega^* \in \Omega^*$ є фіксованим покажчиком обмеженої динаміки F (обмежене Ω), що виражається в пошуку найбільш швидкого спуску по заданій функції. Іншими словами, задача віднайти такі параметри $\omega \in \Omega$, за яких функції спадають максимально швидко в полі F (враховуючи обмеження, що накладає простір Ω), так знаходяться рішення варіаційних нерівностей, що зустрічаються в моделях машинного навчання.

Таким чином, доведено що як оптимізація сідлової точки, так і оптимізація ігор з ненульовою сумою, що охоплює переважну більшість варіантів GAN, запропонованих у літературі, можуть бути імплементовані як в попередньому виразі.

Слід зазначити, що існують декілька альтернативних підходів до вирішення варіаційної нерівності, проте в рамках даної роботи буде розглянуто метод екстраполяції як найбільш широко застосовуваний.

Ідея цієї методики полягає в обчисленні градієнта за екстраградієнтним методом Корпелевича в точці, відмінній від поточної точки, з якої виконується оновлення, стабілізуючи динаміку:

$$\text{обчислення екстрапольованої точки: } \omega_{t+1/2} = P_{\Omega}[\omega_t - \eta F(\omega_t)],$$

$$\text{виконання кроку оновлення: } \omega_{t+1} = P_{\Omega}[\omega_t - \eta F(\omega_{t+1/2})].$$

Можна бачити, що кожен крок ω_t наближення до кінцевої точки виконується не поступово (перебираючи кожне значення функції), а умовними «стрибками», величини яких обчислюються саме за допомогою описаного вище методу екстраполяції. Таким чином, кожне нове значення береться ближче до кінцевого, ніж при класичному підборі.

Навіть у необмеженому випадку цей метод суттєво відрізняється від імпульсу Нестерова через цей крок перегляду градієнта:

$$\text{метод Нестерова: } \omega_{t+\frac{1}{2}} = \omega_t - \eta F\left(\omega_t\right),$$

$$\omega_t + 1 = \omega_{t+1/2} + \beta(\omega_{t+1/2} - \omega_t).$$

Метод Нестерова не сходиться при спробі оптимізації, тобто мінімакс сума прямує до нескінченності за спроби оптимізувати пошук кінцевих точок. Головна причина того, чому екстраполяція має кращу збіжність, ніж стандартний метод градієнту, походить від інтегральної системи Ейлера, за якої збіжність виконується тим швидше, чим менші аргументи цієї системи. Те саме ми маємо і для методу екстраполяції – з уточненням значень на кожному кроці ми наближаємось до знаходження кінцевого стану з заданою точністю.

Дійсно, до першого порядку маємо

$$\omega_{t+1/2} \approx \omega_{t+1} + o(\eta)$$

і, отже, крок оновлення може бути інтерпретований як наближення першого порядку до неявного кроку:

$$\text{неявний крок: } \omega_{t+1} = \omega_t - \eta F(\omega_{t+1}).$$

Неявні методи, як відомо, є більш стабільними та мають переваги від кращих властивостей збіжності, ніж явні методи. Хоча вони, як правило, не практичні, оскільки вимагають вирішення потенційно нелінійної системи на кожному кроці. Отримуємо такі правила оновлення:

$$\{\theta_{t+1} = \theta_t - \eta\varphi_{t+1},\ \varphi_{t+1} = \varphi_t + \eta\theta_{t+1}\}$$

Екстраполяція:

$$\{\theta_{t+1} = \theta_t - \eta\left(\varphi_t + \eta\theta_t\right),\ \varphi_{t+1} = \varphi_t + \eta\left(\theta_t - \eta\varphi_t\right)\}.$$

У наступній тезі, ми бачимо, що при $\eta < 1$ відповідні швидкості збіжності неявний метод та метод екстраполяції дуже подібні. Маючи на увазі, що останнє має головну перевагу в тому, що це є більш практичним, ця теза чітко підкреслює переваги екстраполяції.

Теза. Квадратна норма ітерацій $N_t^2 \stackrel{\text{def}}{=} \theta_t^2 + \varphi_t^2$, де правило оновлення θ_t та φ_t визначене в системі раніше, зменшується геометрично для будь-якого $\eta < 1$ як:

$$\text{Неявний метод: } N_{t+1}^2 = (1 - \eta^2 + \eta^4 + O(\eta^6))\, N_t^2,$$

$$\text{Екстраполяція: } N_{t+1}^2 = (1 - \eta^2 + \eta^4) N_4^2.$$

1.3.3 Сукупність програмних та апаратних засобів для розгортання GAN-мереж

Метою GAN є генерація нових медіа-об’єктів, що супроводжується значними витратами ресурсів, тому для розгортання навіть доволі простої нейронної мережі необхідні апаратні ресурси, проте розвиток сучасних підходів машинного навчання дозволяє уникнути ряду проблем з-за значної

оптимізації зв’язків між апаратними потужностями та програмним забезпеченням.

Навчання генеративно-змагальної моделі нейронної мережі на одному графічному процесорі є досить поширеним явищем, але таке навчання все одно може бути повільним, особливо якщо набір зображень великий, а графічний процесор не такий потужний, щоб опрацьовувати цей набір з достатньою ефективністю. Проте одним із варіантів вирішення апаратної проблеми є використання кластеру машин з GPU.

Для того, щоб отримати незмінно правдоподібні результати, традиційним GAN потрібно десь в межах від 50 000 до 100 000 навчальних зображень. Проте більшість з моделей, як правило, стикаються з проблемою, яка називається перенавчанням. У цих випадках у дискримінатора недостатньо інформації, щоб ефективно впливати на генератор.

Один із способів обійти цю проблему - використовувати підхід, який називається збільшенням даних (аугментація). Знову використовуючи в якості прикладу алгоритм генерації зображень: у випадках, коли немає багато матеріалу для роботи, модель намагається обійти цю проблему, створюючи «спотворені» копії наявного. Спотворення, у цьому випадку, може означати обрізання зображення, його обертання або симетричне відображення. Ідея тут полягає в тому, що мережа ніколи не обробляє одне і те ж саме зображення двічі.

Проблема такого підходу полягає в тому, що він призведе до ситуації, коли GAN навчиться імітувати ці спотворення, замість того, щоб створювати щось якісно нове. Новий підхід адаптивного дискримінатора запропонувала компанія NVIDIA (ADA). Цей підхід все ще використовує збільшення даних, але робить це адаптивно. Замість того, щоб спотворювати зображення протягом усього навчального процесу, це робиться вибірково. Цей підхід вже імплементований в деякі спеціалізовані відеокарти цього виробника, проте є досі досить унікальним, бо аналогів майже не існує.

Загалом, можна зазначити, що так як моделі даного класу працюють переважно з графічними об’єктами, бажано використовувати в процесі навчання та генерації саме графічні відеокарти, тому що вони більш адаптовані до подібної роботи.

Розділ 2

2.1 Огляд середовища розгортання GAN-мереж

Для розгортання методів машинного навчання зручно використовувати програмний пакет *Python3*, через те, що більшість програмних модулів, що імплементують методи машинного навчання підтримуються цим пакетом. До того ж особливості даної мови програмування усувають певні обмеження та дозволяють зосередитися саме на створенні моделі нейронної мережі.

Було вирішено використовувати фреймворк *Keras* (на основі *Tensorflow*) у вигляді пакету для *Python3,* через те, що це найбільш комплексне програмне рішення для відносно простого розгортання GAN-мережі на потужностях звичайного користувацького комп’ютера.

Keras - бібліотека для побудови нейронних мереж, що підтримує основні види шарів і структурні елементи. Підтримує як рекурентні, так і згорткові нейромережі, має в своєму складі реалізацію відомих архітектур нейромереж (наприклад, VGG16).

Дана бібліотека дозволяє на більш високому рівні працювати з нейронними мережами.

TensorFlow — відкрита програмна бібліотека для машинного навчання цілій низці задач, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для виявлення та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди. Її наразі застосовують як для досліджень, так і для розробки продуктів Google часто замінюючи на нього ролі її закритого попередника, *DistBelief*.

TensorFlow надає бібліотеку готових алгоритмів чисельних обчислень, реалізованих через графи потоків даних (data flow graphs). Вузли в таких графах реалізують математичні операції або точки входу/виводу, в той час як ребра графа представляють багатовимірні масиви даних (тензори), які перетікають між вузлами. Вузли можуть бути закріплені за обчислювальними пристроями і виконуватися асинхронно, паралельно обробляючи разом все підходящі до них тензори, що дозволяє організувати одночасну роботу вузлів в нейронної мережі за аналогією з одночасною активацією нейронів в мозку.

Розподілені системи машинного навчання можна створювати на типовому обладнанні, завдяки вбудованій підтримці в *TensorFlow* рознесення обчислень на кілька CPU або GPU.

Цей момент дуже важливий в контексті GAN-мереж, бо зображення зручніше опрацьовувати використовуючи саме графічні процесори, через їхню архітектуру, що спроектована працювати саме з графічними даними.

Додатково було використано пакет *NumPy*, за допомогою якого зручно оперувати масивами даних та проводити звичні математичні операції без зайвого програмування.

NumPy це open-source модуль для *Python*, який надає загальні математичні і числові операції у вигляді пре-скомпільовані, швидких функцій. Вони об'єднуються в високорівневі пакети. Вони забезпечують функціонал, який можна порівняти з функціоналом *MatLab*. *NumPy* (Numeric Python) надає базові методи для маніпуляції з великими масивами і матрицями. *SciPy* (Scientific Python) розширює функціонал *numpy* величезною колекцією корисних алгоритмів, таких як мінімізація, перетворення Фур'є, регресія, і інші прикладні математичні техніки.

Доцільно додатково представити такий зручний інструмент для структурного подання певних концептів методів машинного навчання в зручному вигляді, як *Jupyter Notebook.*

Jupyter Notebook в свою чергу є складовим елементом бібліотеки *IPython*, про яку докладніше далі. *IPython* може інтерактивно взаємодіяти з

бібліотеками *Tkinter*, *GTK*, *Qt* і *WX*. *IPython* може інтерактивно керувати паралельними кластерами використовуючи асинхронні статуси зворотних викликів та / або інтерфейс *MPI*. *IPython* може використовуватися, як заміна стандартної командної оболонки операційної системи. За замовчуванням, *IPython* нагадує роботу shell-оболонок UNIX-подібних систем, але той факт, що робота відбувається в оточенні *Python*, дозволяє досягнути більшої кастомізації і гнучкості у виконанні коду.

Весь процес розробки відбувався в межах віртуального середовища розробки, що дуже важливо для розробки додатків з використанням методів машинного навчання, через розмежування ресурсів комп'ютера між системними процесами. Це гарантує відносно стабільну роботу нейронної мережі за умов довготривалої роботи.

Окремо слід зазначити, що модель проектованої нейронної мережі була створена для генерації зображень з лицями людей на основі вибірки зображень відомих людей.

2.2 Проектування та розгортання моделі генеративно-змагальної нейронної мережі

Перш за все, слід зазначити, що для практичного розгортання була обрана саме модель генеративно-змагальної мережі через її переваги над моделями автокодерів та варіаційних автокодерів, а також через відносну легкість розгортання.

Була поставлена мета дослідження процесу генерації зображень людей, на основі набору з лицями відомих людей з відкритих джерел.

В даному пункті проводиться лише ініціалізація середовища та підготовка зображень, а саме їхнє зменшення до розміру 64 x 64 пікселів (оптимальним вважається розмір 128 x 128 пікселів, проте подібна розмірність зображень вимагає достатніх апаратних потужностей).

В наступних пунктах поступово буде ініціалізована модель генератору та дискримінатору, і як завершальний етап практичного досліду є початок навчання та власне генерація зображень.

В додатку 1 наведено лістинг ініціалізації модулів та підготовки зображень (зменшення розмірності до 64 x 64 пікселів).

2.2.1 Проектування моделі генератора

При проектуванні генератора була використана наступна архітектура нейронної підмережі:

- **Input** визначає розмірність вибірки, що подається на вхід до генератора. Слід зауважити, що це не є звичайна матриця пікселів, яку мережа одразу починає обробляти. На вхід до генератора йде навчальна підвибірка, кожне зображення якої в даному випадку розкладене на канали задля кращого опрацювання та виявлення неочевидних трендів. Відповідно, дані повинні бути попередньо приведені до необхідного формату.
- Шар **Dense** зв'язує шари між собою, бо він приймає дані з попереднього шару, приводить їх до необхідної розмірності, яку вимагає наступний, та передає далі.
- **LeakyReLU** – оптимізована функція активації, що заснована на класичній функції активації **ReLU** - $f(x) = max(0,\ x)$. Відмінності між цими функціями постають в наступному. Функція активації **ReLU** повертає значення аргументу, коли він ненульовий, та нуль, коли аргумент дорівнює нулеві. Інколи, використовуючи класичну функції на початкових шарах, можна спостерігати явище «вимивання» градієнту, коли певні нейрони ніколи не будуть активовані та ваги будуть обраховані хибно. **LeakyReLU**, порівняно з класичним підходом, працює більш оптимізовано: в випадку, коли аргумент прямує до нуля, функція повертає мінімально можливий результат, що врівноважує градієнтний алгоритм загалом та виключає перенасичення нейронів мережі - $f(x) = \{x,\ \text{коли}\ x > 0\ \ 0,01x,\ \text{в інших випадках}$.

- **Conv2D** імплементує згортковий шар, що приводить дані до вигляду тензорів – саме по цих тензорах відбувається просування матриці маски, що було описано детальніше в розділі про згорткові нейронні мережі.

Загальна модель містить один шар входу (**Input**), один шар зв'язування (**Dense**), приведення розмірностей (**Reshape**), шість шарів згортки (**Conv2D**) з функціями активації **LeakyReLU**.

Як ми бачимо, на кінці модель генератора має багатовимірний вихід, що після обробки і стає згенерованим зображенням.

Варто зазначити, що модель генератора приймає на вхід досить велику розмірність даних, а на виході масив пікселів 64 x 64, як і в контрольному наборі даних.

В додатку 2 наведено лістинг моделі генератора.

2.2.2 Проектування моделі дискримінатора

Для дискримінатора підмережа має трохи іншу архітектуру, хоча певні елементи присутні в обох підмережах. Загалом, архітектура визначається функціональними особливостями, що докладніше розібрано далі.

Дискримінатор має кілька ключових елементів, які визначають його здатність до класифікації:

- **Dropout** застосовується для вирішення проблеми перенавчання, цей шар змінює ваги мережі таким чином, щоб уникнути стану, коли дискримінатор втратить змогу давати оцінку схожості початковому класу об'єктів.
- **Flatten** вирівнює розмірність виходу задля того, щоб отримати остаточну відповідь: чи належить згенероване зображення до початкового класу та, відповідно, яким чином змінити ваги мережі.

- Оптимізатор **RMSprop** дозволяє швидше визначати куди рухати ваги всієї системи, бо використовує по елементне піднесення до квадрату, що дозволяє більш якісно визначити напрямок руху градієнту.
- Сигмоїда застосовується в нейронних мережах в якості функції активації - $\sigma(x) = \frac{1}{1+e^{-x}}$. Іншими словами, врівноважує сигнали при переході до наступного шару. Що ще важливо, так це те, що похідна сигмоїди може бути легко виражена через саму функцію, що дозволяє істотно скоротити обчислювальну складність методу зворотного поширення помилки, що і застосовується на практиці досить широко.

Дискримінатор, навпаки від генератора, отримує на вхід саме розмірність зображень, бо його ціль перевіряти відповідність певному класу зображень.

Як ми бачимо, на виході дискримінатора буде саме ознака належності до класу об’єктів, що і дозволяє працювати цьому зв’язку генератор-дискримінатор в єдиному циклі.

В додатку 3 наведено приклад лістингу створення моделі дискримінатора.

2.2.3 Аналіз отриманих результатів

Після ініціалізації функцій створення моделей генератора та дискримінатора слід переходити до запуску навчання моделі та генерації нових зображень.

Додатково слід зазначити, що доцільно зберігати не тільки результати роботи генеративно-змагальної нейронної мережі, але й її ваги, бо їх можна

повторно використати для генерації зображень, якщо зупинити навчання на тому етапі, коли якість згенерованих зображень максимальна.

В додатку 4 наведено лістинг зібраного додатку для генерації нових зображень.

Після завершення навчання та генерації зображень можна переходити до, власне, аналізу результатів. Тривалість навчання та генерації при вхідній вибірці 10000 зображень, кількості епох навчання в 15000 та розмірі вибірки для навчання в 2 склала близько 26 годин на комп’ютері стандартної комплектації.

Доцільно буде навести типові дані для навчання моделі, щоб розуміти яких результат очікувати на виході проектованої моделі:

Рис 5. Вихідні дані для навчання GAN-мережі

Слід зазначити, що після підготовки наведені фото корегуються під формат 64 х 64 та зазнають певних змін, тому на вхід до мережі вони надходять у дещо гіршій якості:

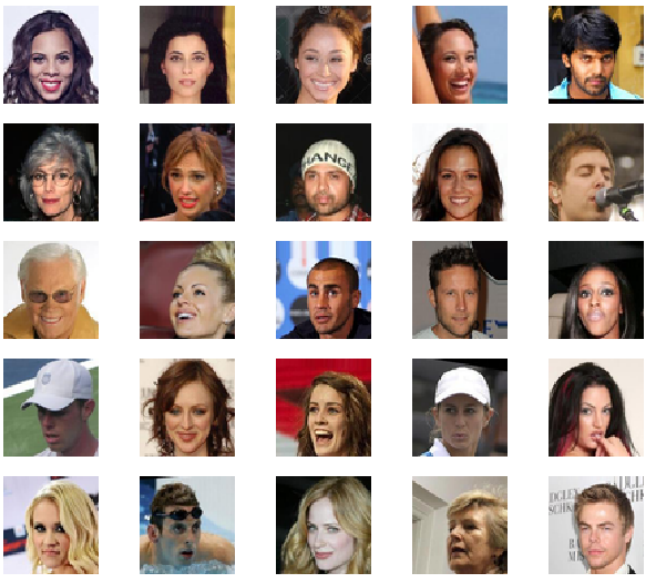

Рис 6. Підготовлені вихідні дані для навчання GAN-мережі

Найбільш виразні результати генерації наведені нижче (сети контрольних зображень 4 х 4):

Рис 7. Результати роботи GAN-мережі

Слід зазначити, що певна «зашумленість» фонових фрагментів чи наявність артефактів на фоні пов’язана із недостатністю вибірки навчання, проте навіть такі результати дозволяють робити більш широкі висновки щодо загального застосування моделей даного класу для вирішення задач генерації графічних об’єктів. Вже згадана недостатність вибірки полягає саме в малій підвибірці для навчання. Тобто, якщо задіяти більший об’єм графічної пам’яті, стане можливим передавати на навчання більші порції зображень, що позитивно має вплинути на кінцевий результат генерації.

Або існує інший підхід збільшення якості генерації, що полягає в передачі на навчання зображень більшої розмірності, проте це вже потребує потужніших графічних процесорів.

Також як можна бачити з ходом навчання якість контрольної вибірки вдосконалюється, проте останні результати, коли на одному сеті 4 х 4 майже збігаються всі фото, свідчать про схильність до перенавчання та тяжіння в наслідок цього до характерних патернів.

В цілому дослідження є якісним підтвердженням того, що дані моделі реально працюють та можуть функціонувати навіть в умовах обмежених ресурсів.

Додатки

Додаток 1.

```
import numpy as np
import pandas as pd
import os
from PIL import Image

from keras import Input
from keras.layers import Dense, Reshape, LeakyReLU, Conv2D,
Conv2DTranspose, Flatten, Dropout
from keras.models import Model
from keras.optimizers import RMSprop

PIC_DIR = f'img_align_celeba/'
from tqdm import tqdm
IMAGES_COUNT = 100

ORIG_WIDTH = 178
ORIG_HEIGHT = 208
diff = (ORIG_HEIGHT - ORIG_WIDTH) // 2
WIDTH = 64
HEIGHT = 64

crop_rect = (0, diff, ORIG_WIDTH, ORIG_HEIGHT - diff)

images = []
for pic_file in tqdm(os.listdir(PIC_DIR)[:IMAGES_COUNT]):
pic = Image.open(PIC_DIR + pic_file).crop(crop_rect)
pic.thumbnail((WIDTH, HEIGHT), Image.ANTIALIAS)
```

```
                    images.append(np.uint8(pic))

                images = np.array(images) / 255
                    print(images.shape)

            from matplotlib import pyplot as plt

                plt.figure(1, figsize=(10, 10))
                    for i in range(25):
                     plt.subplot(5, 5, i+1)
                     plt.imshow(images[i])
                        plt.axis('off')
                        plt.show()



                        LATENT_DIM = 32
                         CHANNELS = 3
```

Додаток 2.

```
    def create_generator():
        gen_input = Input(shape=(LATENT_DIM, ))

        x = Dense(128 * 16 * 16)(gen_input)
        x = LeakyReLU()(x)
        x = Reshape((16, 16, 128))(x)

        x = Conv2D(256, 5, padding='same')(x)
        x = LeakyReLU()(x)

        x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
        x = LeakyReLU()(x)

        x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
```

```
        x = LeakyReLU()(x)

        x = Conv2D(512, 5, padding='same')(x)
        x = LeakyReLU()(x)
        x = Conv2D(512, 5, padding='same')(x)
        x = LeakyReLU()(x)
        x = Conv2D(CHANNELS, 7, activation='tanh', padding='same')(x)

        generator = Model(gen_input, x)

        return generator
```

Додаток 3.

```
    def create_discriminator():
        disc_input = Input(shape=(HEIGHT, WIDTH, CHANNELS))

        x = Conv2D(256, 3)(disc_input)
        x = LeakyReLU()(x)

        x = Conv2D(256, 4, strides=2)(x)
        x = LeakyReLU()(x)

        x = Conv2D(256, 4, strides=2)(x)
        x = LeakyReLU()(x)

        x = Conv2D(256, 4, strides=2)(x)
        x = LeakyReLU()(x)

        x = Conv2D(256, 4, strides=2)(x)
        x = LeakyReLU()(x)

        x = Flatten()(x)
        x = Dropout(0.4)(x)
```

```
    x = Dense(1, activation='sigmoid')(x)
    discriminator = Model(disc_input, x)

    optimizer = RMSprop(
        lr=.0001,
        clipvalue=1.0,
        decay=1e-8
    )

    discriminator.compile(
        optimizer=optimizer,
        loss='binary_crossentropy'
    )

    return discriminator
```

Додаток 4.

```
generator = create_generator()
generator.summary()
discriminator = create_discriminator()
discriminator.trainable = False

gan_input = Input(shape=(LATENT_DIM, ))
gan_output = discriminator(generator(gan_input))
gan = Model(gan_input, gan_output)

optimizer = RMSprop(lr=.0001, clipvalue=1.0, decay=1e-8)
gan.compile(optimizer=optimizer, loss='binary_crossentropy')
```

```
    import time
    iters = 15000
    batch_size = 2

    RES_DIR = 'res2'
    FILE_PATH = '%s/generated_%d.png'
    if not os.path.isdir(RES_DIR):
        os.mkdir(RES_DIR)

    CONTROL_SIZE_SQRT = 2
    control_vectors   =   np.random.normal(size=(CONTROL_SIZE_SQRT**2,
LATENT_DIM)) / 2

    start = 0
    d_losses = []
    a_losses = []
    images_saved = 0
    for step in range(iters):
        start_time = time.time()
               latent_vectors   =   np.random.normal(size=(batch_size,
LATENT_DIM))
        generated = generator.predict(latent_vectors)

        real = images[start:start + batch_size]

        combined_images = np.concatenate([generated, real])

              labels   =   np.concatenate([np.ones((batch_size,   1)),
np.zeros((batch_size, 1))])
        labels += .05 * np.random.random(labels.shape)

             d_loss  =  discriminator.train_on_batch(combined_images,
labels)
```

```
        d_losses.append(d_loss)

                latent_vectors   =   np.random.normal(size=(batch_size,
LATENT_DIM))
        misleading_targets = np.zeros((batch_size, 1))

                    a_loss    =    gan.train_on_batch(latent_vectors,
misleading_targets)
        a_losses.append(a_loss)

        start += batch_size
        if start > images.shape[0] - batch_size:
            start = 0

        if 1:
            gan.save_weights('gan.h5')

            print('%d/%d: d_loss: %.4f,  a_loss: %.4f.  (%.1f sec)' %
(step + 1, iters, d_loss, a_loss, time.time() - start_time))

                control_image  =  np.zeros((WIDTH  *  CONTROL_SIZE_SQRT,
HEIGHT * CONTROL_SIZE_SQRT, CHANNELS))
            control_generated = generator.predict(control_vectors)
            for i in range(CONTROL_SIZE_SQRT ** 2):
                x_off = i % CONTROL_SIZE_SQRT
                y_off = i // CONTROL_SIZE_SQRT
                    control_image[x_off * WIDTH:(x_off + 1) * WIDTH,
y_off  *  HEIGHT:(y_off + 1) * HEIGHT, :] = control_generated[i, :, :,
:]
            im = Image.fromarray(np.uint8(control_image * 255))
            im.save(FILE_PATH % (RES_DIR, images_saved))
            images_saved += 1
```

Висновки

Метою даної роботи було дослідження та порівняння робочих методів та підходів машинного навчання для генерації графічних об’єктів.

Як було визначено в відповідних пунктах роботи, найбільш гнучкою та прогресивною можна вважати саме генеративно-змагальну модель або GAN саме через її відносну оптимізованість навіть в стандартній версії, а також з-за можливості її доволі точного налаштування під різні потреби.

Додатково було оглянуто методи оптимізації даної моделі, що дало змогу зрозуміти наступний факт - генеративно-змагальні моделі можуть бути суттєво вдосконалені за рахунок оптимізації «гри в мінімакс», що описано більш детально в «ненаситному» формулюванні даної моделі. Крім того, це дозволяє уникнути сильної залежності від початкових умов навчання та зосередитися на процесі генерації об’єктів.

Той факт, що автокодери та варіаційні автокодери в певній мірі мали вплив на формування засад генеративно-змагальної моделі, дає змогу зрозуміти, що вони є більш пристосовані для виконання свого класу задач. Саме через це GAN модель була обрана для більш детального практичного дослідження.

Після проведення відповідного практичного дослідження можна зазначити, що даний клас генеративних моделей є цілком самодостатнім для використання в якості класичного підходу в контексті задач генерації зображень (або навіть тривимірних об’єктів, що не розглядається в рамках даної роботи). Проведення практичного дослідження також дало змогу зрозуміти доцільність використання відповідних технологій графічних відео карт саме для задач генерації графічних об’єктів, бо це суттєво зменшує час, що витрачається на навчання та генерацію, та, як наслідок, суттєва економія апаратних ресурсів за значних масштабів використання.

Можна сказати, що існує багато варіантів використання моделей даного класу і це вже доведено реальними прикладами, що були згадані в рамках даної роботи, але головною можливістю, що досі вражає, є створення абсолютно нових, унікальних витворів цифрового мистецтва, бо генеративні моделі не обмежуються лише зображеннями. Існують моделі, що здатні генерувати звуки, людську мову та це лише невелика частина всіх можливостей машинного навчання.

Список використаної літератури

1. Ian J. Goodfellow, Generative Adversarial Nets: [https://arxiv.org/pdf/1406.2661.pdf] / Ian J. Goodfellow, Jean Pouget-Abadie∗, Mehdi Mirza, Bing Xu, David Warde-Farley,Sherjil Ozair†, Aaron Courville, Yoshua Bengio - 2014. - 9 с.
2. Gauthier Gidel, A VARIATIONAL INEQUALITY PERSPECTIVE ON GENERATIVE ADVERSARIAL NETWORKS: [https://arxiv.org/pdf/1802.10551.pdf] - 2019. - 38 с.
3. YANG WANG, A MATHEMATICAL INTRODUCTION TO GENERATIVE ADVERSARIAL NETS (GAN): [https://arxiv.org/pdf/2009.00169.pdf] - 2009. - 30 с.
4. Christopher M. Bishop, PATTERN RECOGNITION AND MACHINE LEARNING [https://cds.cern.ch/record/998831/files/9780387310732_TOC.pdf] - 2006
5. Hugo Berard, A CLOSER LOOK AT THE OPTIMIZATION LANDSCAPES OF GENERATIVE ADVERSARIAL NETWORKS: [https://arxiv.org/pdf/1906.04848.pdf] - 2020. - 18 с.
6. Simon Haykin, NEURAL NETWORKS AND LEARNING MACHINES 3RD [http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf] - 2009
7. Constantinos Daskalakis, Andrew Ilyas, TRAINING GANS WITH OPTIMISM [https://arxiv.org/pdf/1711.00141.pdf] - 2018