УДК 004
DOI https://doi.org/10.32782/IT/2022-2-7

Борис МОРОЗ
доктор технічних наук, професор, професор кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Яворницького, 19, м. Дніпро, Дніпропетровська обл., Україна, 49005, moroz.boris.1948@gmail.com
***ORCID:** 0000-0002-5625-0864*
***Scopus-Author ID:** 57218242332*

Леонід КАБАК
кандидат технічних наук, доцент, доцент кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Яворницького, 19, м. Дніпро, Дніпропетровська обл., Україна, 49005, kabak.leo@gmail.com
***ORCID:** 0000-0001-6267-1772*
***Scopus-Author ID:** 5720222205*

Катерина РОДНА
асистент кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Яворницького, 19, м. Дніпро, Дніпропетровська обл., Україна, 49005, kitti_ukr@ukr.net, Rodna.K.S@nmu.one
***ORCID:** 0000-0002-3610-7091*
***Scopus-Author ID:** 57219006777*

Євгеній РУКСОВ
студент кафедри програмного забезпечення комп'ютерних систем Національного технічного університету «Дніпровська політехніка». просп. Яворницького, 19, м. Дніпро, Дніпропетровська обл., Україна, 49005, euruksov@gmail.com

Бібліографічний опис статті: Мороз, Б., Кабак, Л., Родна, К., Руксов, Є. (2022). Порівняння різних конфігурацій моделі GAN на базі сервісу хмарних обчислень AWS. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 2, 61–78, doi: https://doi.org/10.32782/IT/2022-2-7

ПОРІВНЯННЯ РІЗНИХ КОНФІГУРАЦІЙ МОДЕЛІ GAN НА БАЗІ СЕРВІСУ ХМАРНИХ ОБЧИСЛЕНЬ AWS

У роботі було розглянуто модель машинного навчання, яка реалізує концепцію машинного мистецтва. За основу було взято такий алгоритм штучного інтелекту як Генеративна змагальна мережа (Generative adversarial network, GAN). Порівняно різні конфігурації цієї моделі. Для реалізації моделі було обрано мову програмування Julia. Також було проаналізовано продуктивність моделі на різних платформах, зокрема на базі сервісу хмарних обчислень AWS. В результаті дослідження було розроблено оптимальну конфігурацію моделі; запропоновано комбінацію підходів для навчання моделі; визначено найбільш продуктивну платформу для навчання моделі.

Метою роботи є розробка оптимальної конфігурації системи для ефективного навчання моделі машинного навчання, яка реалізує концепцію машинного мистецтва. Конфігурація системи передбачає такі аспекти як архітектура моделі, алгоритм навчання та платформа, на якій цей алгоритм буде виконуватись. Кожен з цих аспектів опрацьований та запропоновано рішення, яке покаже найкращі результати замірів продуктивності.

Методологія вирішення поставленого завдання полягає в проведенні порівняльного аналізу показників продуктивності різних конфігурацій системи, які запроваджені з урахуванням попередніх досліджень моделей машинного мистецтва, які реалізують концепцію машинного мистецтва.

Наукова новизна. В ході виконання роботи набув подальший розвиток напрямок застосування мови програмування Julia на базі сервісу хмарних обчислень AWS. Вперше запропоновано ефективний варіант програмної та апаратної конфігурації системи для навчання моделі машинного навчання, яка реалізує концепцію машинного мистецтва.

Висновки. Результати даної роботи можуть бути використані для реалізації ефективної системи для навчання моделі машинного навчання, яка реалізує концепцію машинного мистецтва, та в подальших дослідженнях перспектив використання мови програмування Julia в поєднанні з сервісом хмарних

обчислень AWS. Всі отримані результати замірів продуктивності та якості навчання різних конфігурації подано в таблицях та графіках.

Ключові слова: *мова програмування Julia, машинне навчання, машинне мистецтво, Generative adversarial network (GAN), сервіс хмарних обчислень AWS.*

Borys MOROZ
Doctor of Technical Sciences, Professor at Department of Software Engineering, Dnipro University of Technology, 19 Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49005, moroz.boris.1948@gmail.com
Scopus-Author ID: *57202222055*
ORCID: *0000-0002-5625-0864*

Leonid KABAK
Candidate of Technical Sciences, Professor at Department of Software Engineering, Dnipro University of Technology, 19 Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49005, kabak.leo@gmail.com
ORCID: *0000-0001-6267-1772*

Kateryna RODNA
Assistant at Department of Software Engineering, Dnipro University of Technology, 19 Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49005, kitti_ukr@ukr.net, Rodna.K.S@nmu.one
ORCID: *0000-0002-3610-7091*
Scopus-Author ID: *57219006777*

Yevhenii RUKSOV
Student, Dnipro University of Technology, 19 Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49005, euruksov@gmail.com

To cite this article: Moroz, B., Kabak, L., Rodna, K., Ruksov, Ye. (2022). Porivniannia riznykh konfihuratsii modeli GAN na bazi servisu khmarnykh obchyslen AWS [Comparison of different configurations of the GAN model based on the AWS cloud computing service]. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 2, 61–78, doi: https://doi.org/10.32782/IT/2022-2-7

COMPARISON OF DIFFERENT CONFIGURATIONS OF THE GAN MODEL BASED ON THE AWS CLOUD COMPUTING SERVICE

The aim. *The aim of the work is to develop an optimal configuration of the system for effective learning of the machine learning model, which implements the concept of machine art. The configuration of the system includes such aspects as the architecture of the model, the learning algorithm and the platform on which this algorithm will run. Each of these aspects is developed and a solution is proposed that will show the best results of performance measurements. The methodology of the solution of the task set is a comparable analysis of productivity indices of different system configurations that were used taking into account previous studies of machine art models which realized the machine art concept.*

Scientific novelty*. In the course of the work the further development in the direction of application of the Julia programming language based on the AWS cloud computing service was made. An effective version of the software and hardware configuration of the system for learning of a machine learning model that implements the concept of machine art is proposed for the first time.*

Conclusions*.*

The results of this work could be used for implementation of an effective system for learning of a machine learning model that use the concept of machine art, as well as in further studies of the prospects of using the Julia programming language in combination with the AWS cloud computing service. All the results of the measurements of productivity and quality of training of different configurations are presented in tables and graphs.

Key words: *Julia programming language, machine learning, machine art, Generative adversarial network (GAN), AWS cloud computing service.*

Актуальність проблеми. Відома теза «мозок людини – найскладніша обчислювальна система в світі» має багато підтверджень та проявів, особливо, у сучасному світі. Існує певний набір показників, значення яких свідчить про існування та рівень розвитку цивілізації. Мозок, як велика нейронна мережа, якість і швидкість навчання якої та складність оброблених та згенерованих нею структур надзвичайно високі, зміг дійти до такого рівня ефективності, що зі складних структур навколиш-

нього світу зміг створити ще складніші. Саме це і є головним показником, який характеризує цивілізацію.

Проте такий складний процес, як творчість та мистецтво, дуже складно оцінити або проаналізувати. Складною задачею є визначення механізму, який реалізує процес синтезу чогось нового. Над цією задачею працювало вчених, філософів та митців. Бажання пізнавати в людині не менше ніж бажання творити, тому спроби пізнати саму творчість з наукової точки зору робляться постійно. Певного успіху в цьому напрямку досягли саме математики та спеціалісти з інформаційних технологій. Вони пішли шляхом моделювання найбільш близького до людського мозку механізму синтезу складної систематизованої інформації. Ця робота присвячена саме синтезу графічної інформації.

Ця робота відноситься до тієї групи досліджень, які вивчають феномен графічного мистецтва. Це одна з багатьох робіт, які складають загальну картину представлення людини про механізм творчості. Результати цієї роботи дозволять розширити та систематизувати вже накопичені знання в цій сфері. Також дослідження та порівняння різних конфігурацій моделі дозволять робити правильний вибір для ефективної та швидкої роботи в подальших дослідженнях.

Крім наукових досліджень для побудови теоретичної системи опису феномену мистецтва, результати цієї роботи будуть корисними і для спеціалістів з аналізу даних, адже розглянута модель вміє не тільки генерувати зображення, а й розпізнавати справжні та фальшиві зображення. Це може знадобитися, наприклад, в аналізі великої бази картин, і знаходження справжніх, які належать пензлю того чи іншого митця. Також є потенціал застосування такої моделі, наприклад, при розробці ігор, для генерування унікальних ігрових об'єктів.

Отже, актуальність цієї роботи має широке значення, як практичне, так і теоретичне та фундаментальне.

Аналіз останніх досліджень і публікацій. Докладний опис та аналіз досліджень, які описують найбільш перспективні та ефективні методики та підходи до всіх аспектів системи подано в основній частині разом з описом технічних та математичних особливостей системи. Загалом такого порівняльного аналізу різних конфігурацій моделі на рівні дослідження ще не робилося. Головна відмінність від схожих досліджень полягає в тому, що тут застосовується молода мова програмування Julia на базі сервісу хмарних обчислень AWS, а саме додатку AWS SageMaker, який призначений для побудови моделей машинного навчання у зручному інтерфейсі Jupyter Notebook. AWS дозволяє обирати машини різного типу та потужності, що дає змогу робити такий порівняльний аналіз, який представлено в цій роботі. Ще однією особливістю в порівнянні з іншими дослідженнями в цій сфері є те, що з'явилась необхідність розробляти ручний спосіб запуску ядра Julia на базі машини AWS SageMaker, бо зараз цей сервіс не підтримує автоматичні налаштування для Julia.

Мета статті. Необхідно знайти оптимальну конфігурацію системи для ефективного навчання моделі машинного навчання, яка реалізує концепцію машинного мистецтва. Конфігурація системи передбачає такі аспекти як архітектура моделі, алгоритм навчання та платформа, на якій цей алгоритм буде виконуватись. Кожен з цих аспектів опрацьований та запропоновано рішення, яке покаже найкращі результати замірів продуктивності та ефективності.

Виклад основного матеріалу. Для досягнення поставленої цілі роботи необхідно розглянути кожен аспект конфігурації системи окремо. Потім надається порівняльний аналіз.

Перший аспект – це архітектура розроблюваної моделі машинного навчання, яка реалізує концепцію машинного мистецтва. Розглянемо призначення машинного мистецтва як такого.

Комп'ютерна творчість (також відома як штучна творчість, машинна творчість, креативні обчислення або творчі обчислення, машинне мистецтво) – це багатопрофільна діяльність, яка знаходиться на перетині сфер штучного інтелекту, когнітивної психології, філософії та мистецтва.

Метою обчислювальної творчості є моделювання або імітація творчості за допомогою комп'ютера для досягнення однієї з кількох цілей:

створити програму, яка здатна до творчості на рівні людини;

щоб краще зрозуміти творчість людини та сформулювати алгоритмічний погляд на творчу поведінку людей;

розробляти програми, які можуть підвищити творчість людини, не обов'язково бути креативними.

Сфера обчислювальної творчості стосується теоретичних і практичних питань у вивченні творчості. Теоретична робота щодо природи та правильного визначення креативності виконується паралельно з практичною роботою над впровадженням систем, що демонструють

креативність, при цьому один напрям роботи доповнює інший.

Проте ця робота направлена на інше, а саме аналіз конкретних методів та принципів, які дозволяють реалізувати практичну сторону машинної творчості. Тож, розглянемо одну з найбільш розповсюджених моделей штучного інтелекту, які використовуються в цій сфері застосування.

Генеративна змагальна мережа (GAN) – це клас моделей машинного навчання, розроблений Яном Гудфеллоу та його колегами в червні 2014 року. Дві нейронні мережі змагаються один з одним у так званій «грі з нульовою сумою» (гра, у якій гравці мають повністю протилежні цілі, а виграш одного буде означати програш іншого). Загалом це така архітектура моделі, яка дозволяє оцінити генеративні можливості системи через процес змагання, у якому ми одночасно навчаємо дві підмоделі: генеративну модель G, яка фіксує розподіл даних; і дискримінаційну модель D, яка оцінює ймовірність того, що примірник вхідних даних надійшов з навчального набору, а не відноситься до тих, які були згенеровані моделлю G. Процедура навчання для G полягає в тому, щоб максимізувати ймовірність помилки при розпізнанні фальшивки моделлю D. Ця структура відповідає грі «minimax» для двох гравців. У просторі довільних функцій G і D існує єдине рішення, у якому G відтворює розподіл навчальних даних, а D дорівнює всюди 1/2. У випадку, коли G і D є багатошаровими персептронами (повнозв'язна нейронна мережа), всю систему можна навчити за допомогою методу зворотного поширення помилки. Немає потреби в будь-яких ланцюгах Маркова або розгорнутих мережах наближених висновків під час навчання або генерації вибірок. Експерименти демонструють потенціал такої системи шляхом якісної та кількісної оцінки згенерованих зразків. (моделі GAN)

Ян Гудфеллоу з колегами розробили доволі дієву так ефективну модель для вирішення зазначених задач. Розглянемо детально кожен елемент моделі.

Першим розглянемо дискримінатор (в деяких варіаціях GAN, таких як WGAN, його ще називають критиком). Це багатошаровий персептрон, якому на вхід подіється масив даних певної розмірності. Досліджена модель працює із зображеннями, тож далі будуть розглянуто різні методи нормалізації зображень для обробки. Вихід дискримінатора – це одне значення в діапазоні від 0 до 1, що відображає вірогідність належності вхідного примірника до тренувальної вибірки (0 – відповідає згенерованим даним, 1 – даним із тренувального набору). У варіації моделі WGAN на виході критика подається числова оцінка від 0 до певного значення, яке може бути необмеженим.

Генератор – це також багатошаровий персептрон, але має зворотню структуру – вхідних нейронів менше ніж вихідних. Вхідний вектор значень – це так званий шум, який складається із випадкового набору числових значень в діапазоні від –1 до 1. На виході має розмірність таку ж, як на вході у дискримінатора. В залежності від форми вихідних даних є декілька способів збільшення розмірності через шари нейронної мережі. Вони будуть розглянуті далі разом із методами нормалізації зображень для обробки моделлю.

Як відомо із теорії штучних нейронних мереж, кожен нейрон на кожному шарі повинен застосовувати функцію активації над виваженою сумою вхідних значень з урахуванням також додаткових відхилень. Ян Гудфеллоу у своїй роботі використовував так званий випрямлений лінійний нейрон (функція активації має назву ReLU) у комбінації із стандартною сигмоїдою для генератора та функцією активації maxout (на виході нейрона береться максимальне значення серед розрахованих виважених входів) для дискримінатора (Функція активації maxout, яка була застосована в першій публікації про модель GAN). Проте у даній роботі буде використано функцію активації Leaky ReLU (від'ємні значення не відкидаються повністю, а пригнічуються із певним коефіцієнтом) як для генератора, так і для дискримінатора. Значення коефіцієнта для Leaky ReLU обрано 0.2, виходячи із емпіричних даних, які отримали Алек Редфорд та Люк Метс у свій роботі, присвяченій більш складній та потужній моделі DCGAN (моделі DCGAN). Аргументація використання функції активації Leaky ReLU полягає в тому, що ця функція зменшує прояв проблеми зникання градієнту (Свідчення спеціалістів, які підтвердили ефективність застосування функції активації LeakyReLU в моделі GAN). На вихідному шарі генератора було використана функція активації гіперболічний тангенс (tanh), тому що значення кожного згенерованого пікселя повинно бути від –1 до 1. На вихідному шарі дискримінатора використовується стандартна сигмоїда, проте для моделі WGAN вихідне значення може не нормуватись взагалі, тоді змінюється функція помилки для правильної роботи алгоритму навчання.

Розглядалось два основних методи нормалізації вхідної матриці даних:

1. Перетворення матриці у вектор та нормалізація значень яскравості пікселів у діапазон від –1 до 1. Цей підхід придатний для невеликих зображень (наприклад із набору MNIST, де кожне зображення має розмір 28 на 28 пікселів). Але для великих матриць вхідних даних, а особливо коли є декілька каналів (наприклад кольорові зображення RGB), такий спосіб нормалізації вхідних даних неприйнятний, адже це буде потребувати надзвичайно великі обчислювальні потужності.

2. Використання технології згорткової нейронної мережі (CNN), які є спеціалізованим типом нейронної мережі для обробки даних, яка має відому топологію сітки. Приклади включають дані часових рядів, які можна розглядати як одновимірну сітку, що бере вибірки через регулярні інтервали часу, і дані зображення, які можна розглядати як двовимірну сітку пікселів. Згорткові мережі були надзвичайно успішними у практичному застосуванні. Назва «згортка нейронна мережа» вказує на те, що мережа використовує математичну операцію під назвою згортка. Згортка – це спеціалізований вид лінійної операції, в теорії обробки зображень його ще називають локальним оператором або фільтром. Згорткові мережі – це просто нейронні мережі, які використовують згортку замість загального множення матриці принаймні в одному або декількох зі своїх шарів (Гудфелоу). Отже, вхідна матриця даних в такій моделі залишається незмінною, може бути із декількох каналів. Проте для нашої моделі необхідно значення кожного пікселю нормалізувати у діапазон від –1 до 1. Загалом у такої моделі є своя назва – Глибока згорткова генеративна змагальна мережа (DCGAN). Для генератора така модель передбачає шари оберненої згортки.

Тепер представимо код на Julia, який відображає два розглянутих типи моделі GAN:

1. Стандартна модель з першим типом нормалізації вхідних даних та функцією активації сигмоїда на виході дискримінатора:

```
discriminator = Chain(Dense(n_features ^ 2, 1024,
x-> leakyrelu(x, 0.2f0)),
x-> customDropout(x, 0.3f0, true),
Dense(1024, 512, x-> leakyrelu(x, 0.2f0)),
x-> customDropout(x, 0.3f0, true),
Dense(512, 256, x-> leakyrelu(x, 0.2f0)),
x-> customDropout(x, 0.3f0, true),
Dense(256, 1, sigmoid));
generator = Chain(Dense(latent_dim, 256),
BatchNorm(256, x-> leakyrelu(x, 0.2f0)),
Dense(256, 512),
BatchNorm(512, x-> leakyrelu(x, 0.2f0)),
Dense(512, 1024),
BatchNorm(1024, x-> leakyrelu(x, 0.2f0)),
Dense(1024, n_features ^ 2, tanh));
```

2. Моделі DCGAN із лінійною активацією на виході дискримінатора:

```
function create_discr(image_size::Integer)
discr_design = [LayerDesign(4, 2, 1), LayerDesign
(4, 2, 1)];
return Chain(Conv((discr_design[1].kernel, discr_
design[1].kernel), 1 => 64;
stride = discr_design[1].stride, pad = discr_design[1].
pad),
x->leakyrelu.(x, 0.2f0),
x-> customDropout(x, 0.3f0, true),
Conv((discr_design[2].kernel, discr_design[2].
kernel), 64 => 128;stride =
discr_design[2].stride, pad = discr_design[2].pad),
x->leakyrelu.(x, 0.2f0),
x-> customDropout(x, 0.3f0, true),
Flux.flatten,
Dense(culc_size(discr_design, image_size) ^ 2 * 128,
1))
end
function create_gen(image_size::Integer)
gen_design=[LayerDesign(4,2,1),
LayerDesign(4,2,1),LayerDesign(5,1,2)];
size = culc_size(gen_design, image_size);
return Chain(Dense(latent_dim, size ^ 2 * 256, relu),
BatchNorm(size ^ 2 * 256, relu),
x-> reshape(x, size, size, 256, :),
ConvTranspose((gen_design[3].kernel,gen_
design[3].kernel),256=>128;
stride = gen_design[3].stride, pad = gen_design[3].
pad),
BatchNorm(128, relu),
ConvTranspose((gen_design[2].kernel,gen_
design[2].kernel), 128 => 64;
stride = gen_design[2].stride, pad = gen_design[2].
pad),
BatchNorm(64, relu),
ConvTranspose((gen_design[1].kernel, gen_
design[1].kernel), 64 => 1,
tanh; stride = gen_design[1].stride, pad = gen_
design[1].pad))
end
```

В зазначеному коді є декілька додаткових елементів, які необхідні для покращення процесу навчання. Серед них BatchNorm та Dropout. Розглянемо їх більш детально.

Існує так звана «пакетну нормалізація» або «нормалізація по вибірці» (Batch Normalization), яку запропонували Сергій Іові та Крістіан Сегеді з компанії Google в 2015 році. Цей метод полягає в тому, що виконує нормалізацію частини

архітектури моделі та робить це для кожного навчального міні-набору. Такий підхід дозволяє ефективно боротися з феноменом «внутрішнього коваріантного зсуву». Пакетна нормалізація дозволяє використовувати набагато вищі темпи навчання і бути менш обережними в ініціалізації гіперпараметрів моделі. Він також діє як регулятор, у деяких випадках зменшуючи необхідність використання методу Dropout. Суть цього методу полягає в статистичному аналізі та відповідному корегуванні проміжних результатів активацій нейронів у відповідності з нормалізаціє статистичного розподілу цих значень всередині вибірки вхідних примірників. Далі приведене аналітичне представлення цієї ідеї (ідеї про додатковий шар BatchNorm в нейронній мережі)

На вході: значення x в міні-наборі $B = \{x_{1...m}\}$;

На виході: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

Математичне очікування в наборі розраховуємо так:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

Середньоквадратичне відхилення розраховуємо так:

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

Нормалізоване значення активації нейрона розраховуємо так:

$$\widehat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

Масштабування та зміщення нормалізованого значення розраховуємо так:

$$y_i \leftarrow \gamma \widehat{x_i} + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

Цей підхід показав доволі високу ефективність в процесі навчання розглянутих типів нейронних мереж, тому саме його було обрано для розробки ефективної конфігурації системи.

Є ще один методи, який також показав певну ефективність для стабілізації процесу навчання мережі. Цей метод більш простий у реалізації, але має один недолік – необхідність генерації випадкового числа, що додає певної невизначеності внутрішньої роботи мережі. Метод називається «відкидання» (Dropout), винайдений Нітішем Срівастава та колегами із університету в Торонто у 2014 році. Цей підхід допомагає вирішити проблему «перенавчання мережі». Ключова ідея полягає в тому, щоб під час навчання випадковим чином відкидати нейрони (разом з їх вагами та зв'язками з іншими нейронами) із нейронної мережі. Це запобігає надмірній коадаптації нейронів. Під час навчання вилучається вибірка з певною кількістю різних «потоншених» мереж. Під час тестування легко наблизити ефект усереднення прогнозів усіх цих розріджених мереж, просто використовуючи одну нерозріджену мережу, яка має менші ваги. Це значно зменшує ефект перенавчення та дає значні покращення порівняно з іншими методами регуляції. «Відкидання» покращує продуктивність нейронних мереж в завданнях «навчання з вчителем». (ідеї про додатковий шар Dropout в нейронній мережі) Для написання такого алгоритму, щоб його можна було запускати на базі графічного процесора, було необхідно створити власну реалізацію для Dropout, яка наведена нижче:

```
function customDropout(x, p::Float32, active::Bool)
if (!active)
return x
end
y = rand!(similar(x, size(x)))
y = broadcast(el-> el>p ? 1/(1-p) : 0f0, y)
return x .* y
end
```

Обидва методи працюють тільки в циклі навчання, але повинні бути вимкнені при тестуванні та експлуатації мережі. В результаті експериментів ефективною виявилась конфігурація коли BatchNorm застосовується для генератора, а Dropout – для дискримінатора. Це обумовлено тим, що дискримінатор не повинен навчатися занадто швидко, а генератор повинен бути більш стабільним в процесі навчання, щоб не випадати в крайнощі.

Другий аспект – алгоритм навчання моделі. Суть процесу навчання полягає у корегування вагових коефіцієнтів, коефіцієнтів зміщень нейронів, коефіцієнтів масштабування γ та зміщення β для шару BatchNorm та інших параметрів у відповідності із типом оцінки якості роботи моделі та в напрямку підвищення цієї якості. Як відомо із теорії нейронних мереж, існує три основних типи навчання нейронних мереж: з вчителем, без вчителя, з підкріпленням. Навчання з вчителем передбачає наявність чіткого розподілення тренувальної вибірки на класи, тобто розподіл даних відомий завчасно. Навчання без вчителя призначене для пошуку кластерів в множині даних, у яких не має чіткого розподілу по класам. Навчання з підкріпленням передбачає оцінку результату роботи нейронної мережі від середовища, яке здатне реагувати на діяльність моделі. У випадку з моделями GAN та DCGAN на загальному рівні застосовується підхід навчання з вчителем. Для такої моделі існує всього два класи: справжні та згенеровані дані. Проте якщо розглядати

окремо дискримінатор (критик) та генератор, то для першого це навчання з вчителем, а от для другого – навчання з підкріпленням. Генератор корегує свої налаштування у відповідності із реакцією та оцінкою середовища, яким для ного виступає дискримінатор (який в свою чергу теж навчається, але має при цьому чіткий розподіл тренувальних даних). Отже, модель GAN унікальна по своїй суті, адже вона поєднує два типи алгоритму навчання.

Однією із найважливіших частин алгоритму навчання є функція помилки. Їх існує багато та кожна з них ефективна у певній ситуації. Дійсно, від правильності оцінки роботи моделі залежить швидкість та ефективність процесу навчання. У моделі GAN там DCGAN прийнято застосовувати функцію помилки binary cross-entropy або logistic binary cross-entropy. Аналітична аргументація доцільності використання саме цього типу функцій помилки дуже добре виклав Ян Гудфеллоу у своїй роботі (моделі GAN). Тут буде лише зазначено коротке роз'яснення суті бінарної перехресної ентропії.

Для розуміння поняття «ентропія» в теорії інформації треба зазначити, що з точки зору теорії статистики «інформація» еквівалентна «невизначеності» випадкової величини. Тобто інформація відображає невизначеність результату якоїсь події. Це можна розуміти так, що чим більше інформації в системі, тим більш невизначеним є результат роботи цієї системи. Для визначення кількості цієї інформації, а отже рівня невизначеності системи, було введено поняття «ентропія». Як відомо, ентропія (в контексті теорії інформації) – це міра невизначеності системи, міра кількості інформації. Формула ентропії:

$$\mathrm{H}(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i),$$

де X = {x1, x2, …, xn} – випадкова величина, яка може мати n можливих станів; p – вірогідність і-того стану випадкової величини.

Перехресна ентропія, в свою чергу, відображає відстань від розподілу p до розподілу q. В нашому випадку в якості розподілу p виступає розподіл очікуваних вихідних значення моделі, а розподіл q – це розподіл дійсних вихідних значень моделі. Ціль навчання моделі – мінімізація цієї відстані. Проте існує два поняття, які відображають відстань між розподілами: відносна ентропія (або ще називають відстань Кульбака-Лейблера) та перехресна ентропія. Різниця полягає в тому, що для відносної ентропії рівність розподілів визначається нульовим значенням цієї функції, а для перехресної ентропії рівність розподілів – це ентропія розподілу p. Для навчання мережі використовується саме перехресна ентропія, бо для відносної ентропії розподіл p є фіксованим і потрібно підганяти саме розподіл q, а в перехресній ентропії немає такого обмеження. (Опис розрахунку функції помилки binary cross-entropy) Загальна формула перехресної ентропії з n можливих станів випадкової величини X зазначено далі:

$$\mathrm{H}(p,q) = -\sum_{i=1}^{n} p(x_i) \log q(x_i).$$

Бінарність цієї ентропії обумовлена тим, що в моделі GAN є всього два класи для розпізнання даних: справжні та згенеровані. Тому можливих станів випадкової величини всього два. Формула бінарної перехресної ентропії:

$$\mathrm{H}(p,q) = -\left[y \log \hat{y} + (1-y) \log\left(1-\hat{y}\right) \right],$$

де p = {y, 1–y} – очікувані вихідні значення; q = {ŷ, 1–ŷ} – дійсні вихідні значення. Для правильного розрахунку функції помилки на наборі даних треба знайти середнє значення цих ентропій. Формула:

$$L(w) = \frac{1}{N} \sum_{n=1}^{N} \mathrm{H}(p_n, q_n) =$$

$$= -\frac{1}{N} \sum_{n=1}^{N} \left[y_n \log \hat{y}_n + (1-y_n) \log\left(1-\hat{y}_n\right) \right],$$

де L(w) – функція помилки, аргументом якої є множина параметрів моделі; n = {1, 2, …, N} – порядкові номери вхідних примірників.

Логарифмічна бінарна перехресна ентропія – це те ж саме, що бінарна перехресна ентропія, але для покращення якості роботи методу градієнтного спуску над вихідним значенням застосовується сигмоїда.

Ціль навчання моделі, як вже зазначалось раніше, є знаходження мінімуму цієї функції. Для цього зазвичай використовується метод градієнтного спуску та його модифікації (але є альтернативи, наприклад еволюційні алгоритми). Разом з тим, дискримінатор та генератор мають однаковий принцип по мінімізації функції помилки, проте розподіл очікуваних вихідних значень p для них різний. Для дискримінатора згенерованим примірникам відповідає значення 0 (примірникам із тренувального набору – 1), а для генератора – навпаки, згенерованим примірникам відповідає значення 1. При цьому важливо, щоб в процесі навчання корегування параметрів кожної мережі відбувалось у свій час.

Під час навчання нейронної мережі для корегування кожного параметру (це ваги, зсуви, масштабуючі та коригуючі коефіцієнті для специфічних шарів, таких як BatchNorm) необхідно знайти відповідне значення градієнту і відняти його виважене значення від попереднього значення параметра. Ваговий коефіцієнт градієнту – це, так звана, швидкість навчання, і це один із гіперпараметрів моделі (тобто таких, що підбираються вручну). У цьому полягає сутність принципу градієнтного спуску. Принцип оберненого розповсюдження помилки, в свою чергу, описує безпосередньо методику розрахунку градієнтів кожного параметру. Цей алгоритм був винайдений у 1986 році Девідом Румелхартом, Джефрі Хінтоном та Рональдом Вільямсом, а стаття була оприлюднена найбільш авторитетним науковим журналом Nature (Стаття автора алгоритму зворотного розповсюдження помилки та градієнтного спуску).

Проте цей підхід до корегування параметрів моделі в чистому вигляді доволі повільний і потребує багато обчислень. Також для складних сучасних моделей глибокого навчання цей метод без введення певних коректівів виявився взагалі неефективним, а інколи навіть неспроможним досягти оптимального набору значень параметрів. Це було спричинено проблемою занадто швидкого або занадто повільного ковзання по функції помилки в пошуках глобального мінімуму. Також проблему складали локальні мінімуми, з яких цей алгоритм був неспроможний вивести. Основна причина цих проблем – це ручне підбирання гіперпараметру швидкості навчання для кожної конкретної архітектури моделі та кожного типу задачі. Саме тому було винайдено багато різних підходів для оптимізації процесу корегування параметрів, а саме динамічні швидкість навчання. Більше того, сучасні алгоритми оптимізації спроможні навіть розраховувати динамічні коефіцієнти швидкості для кожного конкретного параметру моделі індивідуально. Здебільшого, ці методи використовують статистику та різні поняття з теорії вірогідності.

В багатьох експериментах доволі добре себе показав алгоритм ADAM. Він базується на алгоритмі RMSprop та SGD з моментом. Для кращого розуміння необхідно спочатку визначити основну ідею RMSprop та SGD з моментом. Почнемо з другого, SGD з моментом (Stochastic Gradient Descent with momentum) – це звичайний алгоритм градієнтного спуску, але замість градієнта на один примірник вхідних даних розраховується момент першого порядку або ще називають рухомим середнім чи математичним очікуванням (застосовується емпірична формула для спрощення розрахунків) над значеннями градієнта для набору даних; при цьому коефіцієнт швидкості навчання залишається сталим.

Оптимізатор RMSprop (Root Mean Squared Propagation), в свою чергу, за основу бере градієнт, але коефіцієнт швидкості навчання адаптується на основі моменту другого порядку або ще називають нецентрованою дисперсією (теж застосовується емпірична формула для розрахунків) над градієнтами для набору примірників. Із назви цього методу зрозуміло, що береться квадратний корінь від моменту другого порядку (це є нецентрованим середньоквадратичним відхиленням), а коефіцієнт швидкості навчання ділиться на отримане значення.

Оптимізатор ADAM комбінує обидва підходи: замість градієнту застосовується момент першого порядку, а коефіцієнт швидкості навчання корегується за допомогою ділення на квадратний корінь від моменту другого порядку над градієнтом. Для кращого корегування швидкості навчання особливо на початку, щоб не дати алгоритму занадто швидко ковзати по функції помилки, значення моментів в оригінальній статті (стаття 2015 року за авторством Дієдеріка Кінгма та Джиммі Леі Ба) ще додатково корегуються за допомогою «корекції зміщення» (Стаття автора методу оптимізації алгоритму навчання ADAM).

Отже, комбінація всіх цих методик, доцільність яких була визначна в багатьох попередніх дослідженнях, дозволяють визначити два варіанти моделі, які будуть використовуватись при порівнянні: модель на базі згорткових шарів та модель з повною зв’язністю нейронів. Порівняння застосування тих чи інших додаткових архітектурних рішень та підходів до навчання не є доцільним в цій роботі, адже вони були вже проаналізовані іншими дослідниками. Проведений лише стислий мета-аналіз існуючих робіт.

На поточному етапі розвитку комп’ютерних технологій має місце тенденція до зростання попиту на хмарні сервіси. Це обумовлено тим, що об’єм та складність обчислень досягли такого рівня, що використання власних серверів та обчислювальних систем стало занадто дорого. Висока ціна обумовлена необхідністю постійного неперервного обслуговування великої обчислювальної мережі, тобто треба постійно мати великий штат системних адміністраторів, які будуть слідкувати та підтримувати мережу, і це на додачу до штату програмістів та DevOps. Багато IT-компаній, шукаючи альтернативи такому підходу, прийшли до хмарних сервісів.

Хмарні обчислення базуються на принципі доступності ресурсів комп'ютерної системи, особливо сховища даних (хмарного сховища) та обчислювальної потужності тільки тоді, коли це необхідно, та без прямого активного керування користувачем внутрішньою інфраструктурою. Великі хмари часто мають функції, розподілені в кількох місцях, кожне місце є центром обробки даних. Хмарні обчислення покладаються на спільне використання ресурсів для досягнення узгодженості та зазвичай використовують модель «оплати по мірі використання», яка може допомогти зменшити капітальні витрати, але також може призвести до неочікуваних операційних витрат для несвідомих пересічних користувачів. Проте загалом для побудови великої та складної обчислювальної системи хмари набагато дешевші та мають більш простий інтерфейс для налагодження та підтримки, що може звільнити IT-компанію від необхідності тримати великий штат системних адміністраторів.

Одним з найпопулярніших хмарних сервісів є AWS (Amazon Web Services). Для досягнення мети даної роботи буде використовуватись саме ця платформа. Причиною цьому є те, що серед особливостей хмарних сервісів є можливість запуску алгоритмів на базі машин різного типу та потужності – а отже можна порівняти ефективність різних конфігурацій системи для навчання описаної моделі.

Для роботи із моделями машинного навчання в AWS є сервіс SageMaker. SageMaker дозволяє розробникам працювати на кількох рівнях абстракції під час навчання та розгортання моделей машинного навчання. На найвищому рівні абстракції SageMaker надає попередньо навчені моделі машинного навчання, які можна розгорнути як є. Крім того, SageMaker надає ряд вбудованих алгоритмів ML, які розробники можуть навчати на власних даних. Крім того, SageMaker надає керовані екземпляри TensorFlow і Apache MXNet, де розробники можуть створювати власні алгоритми ML з нуля. Незалежно від того, який рівень абстракції використовується, розробник може підключити свої моделі ML з підтримкою SageMaker до інших служб AWS, таких як база даних Amazon DynamoDB для зберігання структурованих даних, AWS Batch для автономної пакетної обробки або Amazon Kinesis для обробки в реальному часі. Для взаємодії з SageMaker розробникам доступний ряд інтерфейсів. По-перше, є веб-API, який віддалено керує екземпляром сервера SageMaker. Хоча веб-API не залежить від мови програмування, яку використовує розробник, Amazon надає прив'язки API SageMaker для кількох мов, включаючи Python, JavaScript, Ruby, Java та Go. Крім того, SageMaker надає керовані екземпляри Jupyter Notebook для інтерактивного програмування SageMaker та інших програм. Саме Jupyter Notebook разом з мовою програмування Julia буде використовуватись для побудови та навчання моделі. Проте для запуску серверу Julia на базі машини SageMaker необхідно описати скрипт, який буде завантажувати та запускати всі необхідні програми для роботи серверу Julia на базі Jupyter Notebook. Ось shell- скрипт, який завантажує Julia під час створення машини SageMaker:

```
#!/bin/bash
set-e
sudo-u ec2-user-i <<EOF
echo «. /home/ec2-user/anaconda3/etc/profile.d/conda.sh» >> ~/.bashrc
conda create--yes--prefix ~/SageMaker/envs/julia
wget -q https://julialang-s3.julialang.org/bin/linux/x64/1.7/julia-1.7.2-linux-x86
_64.tar.gz-O- | tar xzf-
cp-R julia-1.7.2/* ~/SageMaker/envs/julia/
mkdir -p ~/SageMaker/envs/julia/etc/conda/activate.d
echo 'export JULIA_DEPOT_PATH=~/SageMaker/envs/julia/depot' >> ~/Sage
Maker/envs/julia/etc/conda/activate.d/env.sh
echo -e 'empty!(DEPOT_PATH)\npush!(DEPOT_PATH,raw»/home/ec2-user/
SageMaker/envs/julia/depot»)' >> ~/SageMaker/envs/julia/etc/julia/
startup.jl
conda activate /home/ec2-user/SageMaker/envs/julia
julia--eval 'using Pkg; Pkg.add(«IJulia»); using IJulia'
EOF
```

Скрипт, який запускає сервер Julia під час ввімкнення машини SageMaker:

```
#!/bin/bash
set-e
sudo-u ec2-user-i <<EOF
echo «. /home/ec2-user/anaconda3/etc/profile.d/conda.sh» >> ~/.bashrc
conda run --prefix ~/SageMaker/envs/julia/ julia --eval 'using IJulia;
IJulia.installkernel(«Julia»)'
EOF
```

Для порівняння було обрано два типи машини: на базі CPU (ml.c5.2xlarge) та на базі GPU (ml.g4dn.xlarge). Обчислення на графічному процесорі потребують використання деяких бібліотек та відповідних функцій. Для автоматичного розподілу задач та даних на графічному процесорі Nvidia застосовується бібліотека CUDA. Детальніше про використані машини SageMaker в таблиці 1.

Таблиця 1

Використані машини SageMaker

Тип машини	CPU (ядра)	GPU	ОЗП (Гб)	SSD (Гб)	Ціна за час (USD)
ml.c5.2xlarge	8	**інтегрована**	16	5	0,408
ml.g4dn.xlarge	4	NVIDIA T4	16	5	0,7364

Загалом маємо 4 конфігурації:

1. GAN на ml.c5.2xlarge;
2. CGAN на ml.c5.2xlarge;
3. GAN на ml.g4dn.xlarge;
4. CGAN на ml.g4dn.xlarge.

В обох типах моделей на вхід генератора подається випадковий шум на 100 значень в діапазоні від –1 до 1. Навчання проходить на стандартному наборі чорно-білих зображень цифр MNIST (Modified National Institute of Standards and Technology database). Це базовий стандарт для навчання різних моделей машинного навчання і саме він використовується в більшості досліджень для оцінки якості роботи моделі. Загальний розмір навчальної вибірки для всіх конфігурацій становить 2000 зображень. Розмір підвибірки становить 128 зображень. Для кожної епохи навчальна вибірка перемішується, що запобігає ефекту перенавчання. Всі інші гіперпараметри (наприклад швидкість навчання для оптимізатора ADAM, що становить 0.0002 як для генератора, так і для дискримінатора) однакові для всіх чотирьох конфігурацій системи. Єдине, що відмінне між GAN та CGAN, – це загальна кількість параметрів, які потрібно корегувати для навчання моделі. Для GAN кількість параметрів дорівнює 2950161 (займає при серіалізації моделі близько 11.269 Мб на диску). А для CGAN – 2382466 (займає при серіалізації моделі близько 9.201 Мб на диску).

У результаті багатьох експериментів виявилось, що оптимальний баланс між генератором та дискримінатором досягається при середньому значення функції помилки на епоху у дискримінатора – близько 0.5–0.6, а у генератора – близько 1.0-1.1. Подальші ітерації навчання не дають значного покращення якості роботи моделі.

В результаті експериментів було зроблена оцінка таких показників, як:

1. Час обробки однієї епохи навчання;
2. Кількість епох, потрібних для досягнення оптимального балансу між генератором та дискримінатором;
3. Форма графіку зміни середнього значення функції помилки по епохам;
4. Якість згенерованих зображень (суб'єктивна оцінка).

Далі в таблиці 2 подано заміри часу обробки однієї епохи.

Таблиця 2

Заміри часу обробки однієї епохи

	GAN	CGAN
ml.c5.2xlarge	37	437
ml.g4dn.xlarge	0.1	0.45

Як бачимо, різниця в швидкості обробки між машиною на CPU та машиною на GPU значна. Машина ml.g4dn.xlarge в 370 рази швидша в роботі з GAN та 971 раз швидша в роботі з CGAN ніж машина ml.c5.2xlarge. Дивлячись з площини порівняння GAN та CGAN, то другий виявився складніший для обробки обома типами машин. Хоча CGAN має меншу кількість параметрів та займає менше місця в пам'яті, ця модель складніша з точки зору обчислення градієнтів для кожного параметра. Далі в таблиці 3 подано кількість епох до оптимуму.

Таблиця 3

Кількість таблиць до оптимуму

	GAN	CGAN
ml.c5.2xlarge	1200	300
ml.g4dn.xlarge	1200	300

Бачимо, що CGAN досягає оптимального стану в 4 рази швидше ніж GAN. Розраховуючи загальний час навчання кожної моделі, отримаємо такі середні значення (таблиця 4).

Таблиця 4

Загальний час навчання кожної моделі

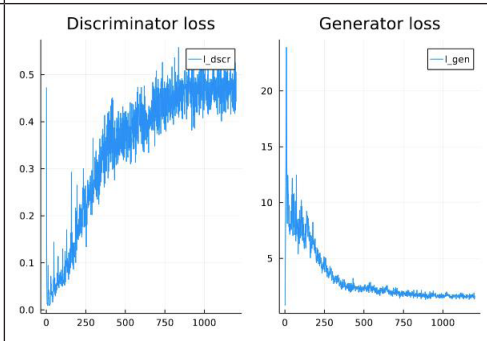
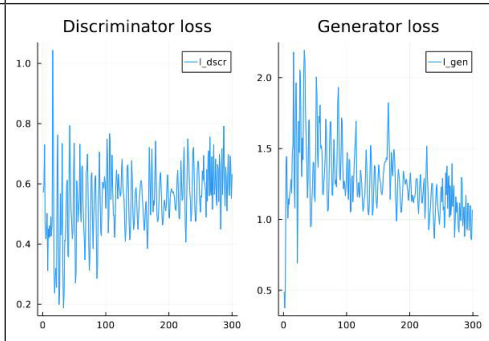
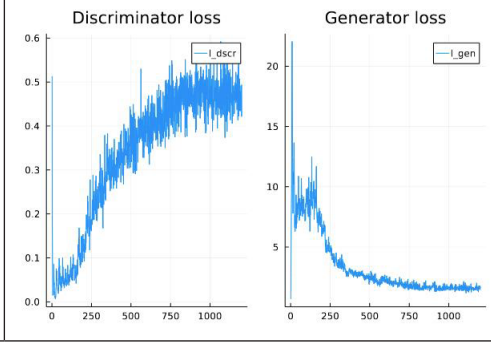
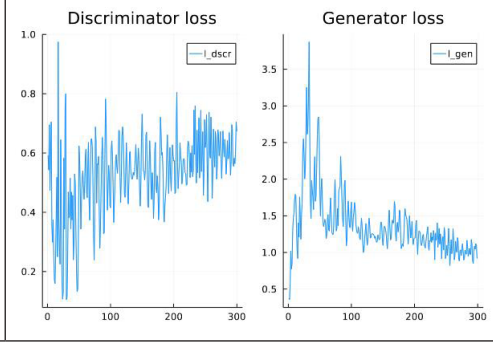
	GAN	CGAN
ml.c5.2xlarge	44400	131100
ml.g4dn.xlarge	120	135

Різниця між моделями для CPU-машини занадто велика, майже в 3 рази. Проте для GPU-машини ця різниця складає всього 15 секунд. І загалом, вже можна констатувати, що оптимальним вибором для навчання таких моделей є саме GPU-машина. Таку складну моделі як GAN ці машини можуть ефективно навчити на базі стандарту MNIST за 2–3 хвилини.

Представимо графіки зміни середнього значення функції помилки по епохам. Порівняння подано в таблиці 5.

Таблиця 5

Графіки зміни середнього часу по епохам

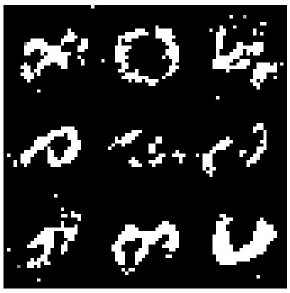
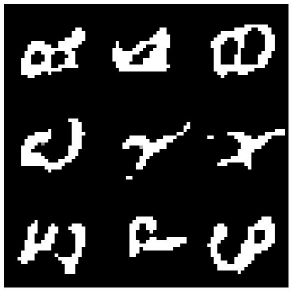
	GAN	CGAN
ml.c5.2xlarge		
ml.g4dn.xlarge		

Як бачимо, графіки зміни середнього значення функції помилки для моделі GAN мають експоненціальну форму і змінюються повільніше. Графіки для CGAN, в свою чергу, виходять в оптимальну полосу вже через 100–200 епох, далі функція лише стабілізується ти вирівнюється – немає експоненціальної форми. З іншого боку, графік для GAN більш стабільний та має менший діапазон відхилень. Тобто характер зміни функції і спосіб досягнення оптимуму різний. Для CGAN – це стабілізація «стрибаючої по великому діапазону» функції навколо оптимуму, а для GAN – експоненціальне наближення.

Далі представлено згенеровані зображення (по 9 на кожну конфігурацію). Порівняння подано в таблиці 6.

Таблиця 6

Згенеровані зображення

	GAN	CGAN
ml.c5.2xlarge		
ml.g4dn.xlarge		

Знову будемо порівнювати GAN та CGAN між собою. За рахунок згорткових шарів CGAN має ефект згладжування зображення і видалення шумів. У GAN такого немає, тому в згенерованих зображеннях спостерігаються зайві артефакти та шуми, при цьому, навіть, збільшуючи кількість епох, цей шум не зменшується. З цього можна зробити висновок, що для роботи із зображеннями згорткові шари все ж дають більш «чистий» результат. Загалом, обидві моделі ефективно впоралися із завданням генерації зображенням і з суб'єктивної точки зору більшість згенерованих зображень цифр можна впізнати.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямку. В ході виконання даного дослідження набув розвитку напрямок побудови, навчання та застосування моделі, яка реалізує концепцію машинного мистецтва, на базі сервісу хмарних обчислень AWS. Був проведений порівняльний аналіз попередніх робіт, в яких були винайдені найбільш ефективні на даний момент методи та підходи для побудови такої моделі. Також був проведений порівняльний аналіз чотирьох конфігурацій системи, серед яких була визначена найбільш оптимальна, а саме CGAN на базі GPU-машини ml.g4dn.xlarge. Були виважені всі аспекти, які дозволили знайти такий варіант, який тримав баланс між швидкодією та якістю.

Подальші перспективи дослідження можна окреслити в таких категоріях:

1. Дослідження моделі WGAN в цьому ж контексті;
2. Дослідження рекурентних нейронних мереж для генерації, наприклад, музики в цьому ж контексті;
3. Дослідження генетичних алгоритмів для оптимізації процесу навчання також у контексті порівняння з іншими конфігураціями системи;

Також є практичне побажання – автоматизація конфігурації ядра мови програмування Julia на машинах AWS SageMaker, адже ця мова є однією з найбільш перспективних в сфері машинного навчання та науки про данні.

ЛІТЕРАТУРА:

1. Стаття авторів моделі GAN. [Electronic resource] – Access mode: https://arxiv.org/pdf/1406.2661.pdf
2. Функція активації maxout, яка була застосована в першій публікації про модель GAN [Electronic resource] – Access mode: https://arxiv.org/pdf/ 1302.4389v4.pdf
3. Стаття авторів моделі DCGAN [Electronic resource] – Access mode: https://arxiv.org/pdf/1511.06434.pdf
4. Свідчення спеціалістів, які підтвердили ефективність застосування функції активації LeakyReLU в моделі GAN [Electronic resource] – Access mode: https://www.reddit.com/r/MLQuestions/comments/c96mci/why_do_gans_only_work_with_leaky_relu/
5. Розділ книги Яна Гудфелоу, присвячений згортковим нейронним мережам [Electronic resource] – Access mode: https://www.deeplearningbook.org /contents/convnets.html
6. Стаття автора ідеї про додатковий шар BatchNorm в нейронній мережі [Electronic resource] – Access mode: https://arxiv.org/pdf/1502.03167.pdf
7. Стаття автора ідеї про додатковий шар Dropout в нейронній мережі [Electronic resource] – Access mode: https://www.cs.toronto.edu/~rsalakhu /papers/srivastava14a.pdf
8. Опис розрахунку функції помилки binary cross-entropy [Electronic resource] – Access mode: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a
9. Стаття автора алгоритму зворотного розповсюдження помилки та градієнтного спуску [Electronic resource] – Access mode: https://www.nature.com /articles/323533a0
10. Стаття автора методу оптимізації алгоритму навчання ADAM [Electronic resource] – Access mode: https://arxiv.org/pdf/1412.6980.pdf

REFERENCES:

1. https://arxiv.org/pdf/1406.2661.pdf
2. https://arxiv.org/pdf/1302.4389v4.pdf
3. https://arxiv.org/pdf/1511.06434.pdf
4. https://www.reddit.com/r/MLQuestions/comments/c96mci/why_do_gans_only_work_with_leaky_relu/
5. https://www.deeplearningbook.org/contents/convnets.html
6. https://arxiv.org/pdf/1502.03167.pdf
7. https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf
8. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3a-c6025181a
9. https://www.nature.com/articles/323533a0
10. https://arxiv.org/pdf/1412.6980.pdf