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:

Алгоритми екстраградієнтного типу та їх застосування в машинному навчанні

Виконала студентка 4-го курсу

Кісленко Олена Сергіївна _____

Науковий керівник:

доктор фіз.-мат. наук, професор

Семенов Володимир Вікторович _____

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студентка _____

Роботу розглянуто й допущено до захисту на засіданні кафедри обчислювальної математики

«__29__» _____травня_____ 2023 р.,

протокол № __8__

Завідувач кафедри

С. І. Ляшко _____

Київ — 2023

РЕФЕРАТ

Обсяг роботи 40 сторінок, 12 ілюстрацій, 13 джерел посилань, 1 додаток.

Ключові слова: ЕКСТРАГРАДІЄНТНІ АЛГОРИТМИ ОПТИМІЗАЦІЇ, МАШИННЕ НАВЧАННЯ, ГЕНЕРАТИВНІ ЗМАГАЛЬНІ МЕРЕЖІ.

Об'єктом роботи є порівняння застосування екстраградієнтних методів оптимізації для тренування генеративних змагальних мереж.

Метою роботи є визначення кращих екстраградієнтних алгоритмів оптимізації для генерування зображень за допомогою генеративних змагальних мереж.

Інструменти розроблення: Google Colaboratory, мова програмування Python, середовище програмування PyCharm, бібліотека Tensorflow.

Результати роботи: Було наведено формулювання екстраградієнтних алгоритмів, протестовано і порівняно їх роботу на генеративній змагальній мережі. Для описаних моделей з екстраградієнтними оптимізаторами також були знайдені деякі чисельні оцінки якості і зроблено висновок, що вони не завжди точно характеризують якість згенерованих зображень.

ЗМІСТ

РОЗДІЛ 1
ВСТУП

Машинне навчання є однією з найактуальніших галузей сучасної науки, яка займається розвитком алгоритмів та моделей, здатних вчитися та самостійно вдосконалюватися на основі даних. Одним з важливих напрямків машинного навчання є генерувальні моделі. Найбільш поширеними моделями для таких задач є варіаційні автоенкодери (VAE) і генеративні змагальні мережі. В даній роботі саме другі будуть розглядатися детальніше.

GAN - це модель, що складається з двох нейронних мереж, які називаються генератором і дискримінатором. Під час тренування моделі GAN головною метою є досягнення рівноваги між генератором і дискримінатором, де генератор здатний генерувати реалістичні зразки, які важко відрізнити від реальних даних, а дискримінатор вміє ефективно розрізняти між справжніми та згенерованими зразками. Це досягається через мінімізацію функції втрати, яка включає функцію втрати генератора і функцію втрати дискримінатора. Зазвичай дві складові навчаються по черзі, тобто відбувається чергування між тренуванням генератора і тренуванням дискримінатора.

Оптимізація GAN є складною задачею через структуру самої моделі, яка включає дві конкуруючі нейронні мережі. Базовим алгоритмом оптимізації, що можна застосовувати до нейронних мереж, є стохастичний градієнтний спуск. Проте, він не завжди збігається до розв'язку оптимізаційної задачі. Тому було запропоновано використовувати екстраградієнтні алгоритми.

Об'єктом дослідження даної роботи є екстраградієнтні алгоритми і їх застосування в генеративних змагальних мережах. Метою роботи є порів-

няти результати застосування екстраградієнтних алгоритмів зі стандартними для тренування моделей машинного навчання, і встановити їх ефективність для таких задач.

РОЗДІЛ 2
ЕКСТРАГРАДІЄНТНІ АЛГОРИТМИ ОПТИМІЗАЦІЇ

2.1. Формулювання задачі. Варіаційні нерівності.

Нехай $C \in \mathbb{R}^n$ – непорожня множина, оператор $A : C \to \mathbb{R}^n$. Варіаційною нерівністю називається задача на знаходження елементу $x \in C$ такого, що

$$\langle Ax, y - x\rangle \geqslant 0 \quad \forall y \in C. \tag{2.1}$$

Опуклі екстремальні задачі можна подати у вигляді варіаційних нерівностей. Дійсно, нехай заданий критерій оптимальності

$$f(x) \longrightarrow min, \quad x \in C,$$

і множина C є замкненою і опуклою, а f – диференційовна і опукла функція. Тоді

$$f(x^*) = \min_{y \in C} f(y) \Longleftrightarrow x^* \in C, \ \langle \nabla f(x^*), y - x^*\rangle \geqslant 0 \quad \forall y \in C.$$

При цьому, постановка варіаційної нерівності описує більш широкий клас задач, ніж екстремальні задачі.

Наведемо деякі визначення, які знадобляться для подальшого розгляду.

Визначення 2.1. Оператор $A : C \to \mathbb{R}^n$ називається монотонним, якщо $\langle Ax - Ay, x - y\rangle \geqslant 0 \ \ \forall x, y \in C$.

Визначення 2.2. Оператор $A : C \to \mathbb{R}^n$ називається ліпшицевим з константою Ліпшиця L, або L-ліпшицевим на множині C, якщо $\|Ax - Ay\| \leqslant R\|x - y\| \quad \forall x, y \in C$.

Визначення 2.3. Метричною проекцією гільбертового простору H на замкнену і опуклу підмножину $C \in H$ називають оператор P_C такий, що

$$\|P_C x - x\| = \min_{y \in C} \|y - x\|.$$

2.2. Основні алгоритми для розв'язання варіаційних нерівностей.

Розглянемо відомі методи розв'язання варіаційних нерівностей вигляду (2.1).

Надалі будемо вважати, що виконані такі умови:

1. Множина C є замкненою та опуклою.
2. Оператор $A : C \to \mathbb{R}^n$ є L-ліпшицевим на множині C.
3. Існує хоча б один розв'язок (2.1).

Найпростішим методом пошуку розв'язку варіаційної нерівності є градієнтний метод, визначений таким чином:

Алгоритм 2.1 (Градієнтний метод)**.**

$$z_{k+1} = P_C(z_k - \alpha A z_k),$$

де $z_0 \in C$ – довільна точка. Проте, існують задачі з монотонними та ліпшицевими операторами, для яких запропонований алгоритм не збігається до розв'язку. Для збіжності треба накласти більш сильні умови, наприклад, сильну монотонність або обернену сильну монотонність [2].

Далі був розроблений екстраградієнтний алгорим, представлений у [4], а також його узагальнення.

Алгоритм 2.2 (Екстраградієнтний алгоритм)**.**

$$z_{k+1/2} = P_C(z_k - \alpha A z_k)$$
$$z_{k+1} = P_C(z_k - \alpha A z_{k+1/2}),$$

де $\alpha \in \left(0, \frac{1}{L}\right)$, L – константа Ліпшиця оператора A, $z_0 \in C$.

Або, в іншому вигляді, крок екстраградієнтного алгоритму можна записати у такому вигляді:

$$z_{k+1} = P_C(z_k - \alpha A(P_C(z_k - \alpha A z_k))).$$

Алгоритм екстраполяції з минулого є модифікацією екстраградієнтного алгоритму, яка дозволяє проводити всього одне обчислення образу оператора на кожному кроці.

Алгоритм 2.3 (Метод екстраполяції з минулого)**.**

$$z_{k+1/2} = P_C(z_k - \alpha A z_{k-1/2})$$
$$z_{k+1} = P_C(z_k - \alpha A z_{k+1/2}),$$

де $\alpha \in \left(0, \frac{1}{3L}\right)$, L – константа Ліпшиця оператора A, $z_0 \in C$. Після виконання обчислення значення $Az_{k+1/2}$ можна зберегти для наступної ітерації.

2.3. Модифікації алгоритмів.

У задачах, які виникають у машинному навчанні, застосування звичайних алгоритмів оптимізації може займати багато часу через необхідність обчислювати градієнт в усіх точках.

Надалі будемо вважати, що задана задача без обмежень, тобто $C = \mathbb{R}^n$.

Розглянемо задачу з функцією втрат, що можна представити у вигляді суми

$$F(x) = \frac{1}{n}\sum_{i=1}^{n} F_i(x). \tag{2.2}$$

Зазвичай $F_i(x)$ можна поставити у відповідність i-му спостереженню з наявних даних (зокрема, $F_i(x)$ може бути функцією втрат i-го спостереження).

Для таких задач використовують декілька підходів, що дозволяють значно зменшити кількість обчислень, обчислюючи градієнти не в усіх точках (спостереженнях), а тільки в частині з них.

Розглянемо ці підходи детальніше.

2.3.1. Стохастичні алгоритми.

У традиційних алгоритмах оптимізації зазвичай використовується весь набір даних або функцій для обчислення градієнта і пошуку оптимального розв'язку. Проте, у великих і складних задачах такий підхід може бути непрактичним через обчислювальну складність або обмежену доступність даних. Стохастичні алгоритми оптимізації пропонують альтернативу, де замість використання повного набору даних або функцій, вони використовують підмножину даних або випадковий вибір даних для оцінки функції втрат або обчислення градієнта. Цей підхід дозволяє знизити обчислювальну складність і швидше знаходити наближений оптимальний розв'язок.

Для заданої у вигляді (2.2) функції втрат градієнт буде мати вигляд

$$\nabla F(x) = \frac{1}{n} \sum_{i=1}^{n} \nabla F_i(x).$$

Для зменшення кількості обчислень дану суму можна наблизити значенням $\nabla F_i(x)$ для одного випадкового i, отримавши стохастичний алгоритм. Тобто, основна ідея стохастичних алгоритмів оптимізації полягає у тому, що в них параметри моделі на кожній ітерації оновлюються з використанням випадково обраних екземплярів.

Зокрема, стохастичний варіант градієнтного методу для функції, представленої у вигляді (2.2), має такий вигляд:

$$z_{k+1} = z_k - \alpha \nabla F_i(z_k).$$

Аналогічно можна отримати стохастичні варіанти інших алгоритмів.

Оскільки використовується градієнт тільки одного спостереження, то загальна функція втрат не обов'язково спадатиме на кожному кроці і може не досягти мінімума функції. Проте, даний алгоритм дозволяє отримати певне наближення.

2.3.2. Міні-батч алгоритми.

Для міні-батч алгоритмів наявні спостереження розбивають на міні-групи (має бути більше одної непорожньої міні-групи). В кожній з таких груп обраховується середнє значення градієнту в ній, і результат оновлюється на ці середні значення помножені на крок алгоритму.

Для даних алгоритмів можна використати векторизацію, що додасть певне пришвидшення. За маленьких розмірах міні-груп процес відносно швидко збігається, але збільшується кількість шуму. Навпаки, при великих розмірах міні-груп процес збігається довше, але можна отримати більш точні значення.

Міні-батч алгоритми є рекомендованими варіантами алгоритмів для задач машинного навчання.

2.4. Adam.

Алгоритм Adam (Adaptive Moment Estimation) є одним з найбільш часто застосованих алгоритмів для тренування нейронних мереж та інших моделей машинного навчання.

Однією з головних переваг даного алгоритму є його здатність працювати з великими наборами даних та моделями з великою кількістю параметрів. Він добре пристосовується до широкого спектру завдань машинного навчання, демонструє стабільну та ефективну роботу.

Основною ідеєю алгоритму Adam є використання наближених значень першого та другого моментів градієнтів для оновлення кожного параметра моделі, звідки й походить назва Adaptive Moment Estimation.

Через α позначимо крок алгоритму, $\beta_1, \beta_2 \in [0, 1)$ – порядки розпаду моментів. Початкові значення задаються таким чином:

$$m_0 = 0, \quad v_0 = 0, \quad t = 0,$$

де t – час (номер ітерації), m_0 і v_0 перший та другий моменти відповідно.

Алгоритм має такий вигляд:

Алгоритм 2.4 (Adam)**.**

for $t = 0, 1, ..., T-1$ **do**

$g_t = \nabla_\theta f_t(z_{t-1})$;

$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$;

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

$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$;

$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$;

$z_t = z_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$;

2.4.1. ExtraAdam.

Як модифікацію звичайного алгоритму Adam можна розглянути Adam з екстраполяційним кроком (ExtraAdam), що представлений у [3].

Нехай задані крок α, порядки розпаду β_1, β_2 та початкові значення z_0, m_0, v_0. Алгоритм виглядає таким чином:

Алгоритм 2.5 (ExtraAdam)**.**

for $t = 0, 1, ..., T-1$ **do**

$g_t = \nabla f(z_{t-1/2})$;

$m_{t-1/2} = \beta_1 m_{t-1} + (1-\beta_1) g_t$;

$v_{t-1/2} = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$;

$\hat{m}_{t-1/2} = \frac{m_{t-1/2}}{1-\beta_1^{2t-1}}$;

$\hat{v}_{t-1/2} = \frac{v_{t-1/2}}{1-\beta_2^{2t-1}}$;

$z_{t-1/2} = z_t - \alpha \frac{\hat{m}_{t-1/2}}{\sqrt{\hat{v}_{t-1/2}}+\epsilon}$;

$g_{t+1/2} = \nabla f(z_{t+1/2})$;

$m_t = \beta_1 m_{t-1/2} + (1-\beta_1) g_{t+1/2}$;

$v_t = \beta_2 v_{t-1/2} + (1-\beta_2) g_{t+1/2}^2$;

$\hat{m}_t = \frac{m_t}{1-\beta_1^{2t}}$;

$\hat{v}_t = \frac{v_t}{1-\beta_2^{2t}}$;

$z_{t+1} = z_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$;

РОЗДІЛ 3
ГЕНЕРАТИВНІ ЗМАГАЛЬНІ МЕРЕЖІ

3.1. Огляд структури GAN.

Генеративні змагальні мережі (Generative Adversarial Networks, GANs) – клас алгоритмів машинного навчання, ціллю яких є генерування результату, що міг би бути взятий з вхідного набору даних. Вони були запропоновані у 2014 році вченим Іаном Гудфеллоу [1]. Наразі GANs є одним з найпопулярніших підходів для генерації реалістичних зображень, звуку та тексту.

Генеративні Змагальні Мережі складаються з двох нейронних мереж, що називаються генератором та дискримінатором.

— Метою генератору є власне генерація нових даних, основуючись на вхідному наборі даних. На вхід генератор отримує випадковий вектор з так названого прихованого простору.

— Дискримінатор є класифікатором, що отримує на вхід набір екземплярів даних. Метою дискримінатора є максимально правильно сказати про кожен зразок, чи він є справжнім, чи створений генератором.

Нейронні мережі генератора та дискримінатора тренуються сумісно і разом утворюють спільну мережу GAN. В результаті, вже натренований генератор можна використовувати для створення нових екземплярів даних, як-от зображень.

3.2. Формальний опис GAN.

Нехай є заданий розподіл даних p_{data}. Позначимо параметри моделі генератора через $\theta_g \in \mathbb{R}^g$, а параметри моделі дискримінатора через $\theta_d \in \mathbb{R}^d$. Тоді моделі генератора та дискримінатора можна представити у вигляді функцій

$$G(z;\theta_g) : \mathbb{R}^g \to X; \quad D(x;\theta_d) : X \to [0,1].$$

Генератор отримує на вхід шум z, який має розподіл p_z, і переводить їх у дані з певного простору X. В свою чергу, дискримінатор отримує екземпляр даних з цього простору X і перетворює його на число з $[0,1]$ – ймовірність того, що вхідні дані належали справжньому розподілу p_{data}.

Нехай на вхід генератору подається шум $p_z(z)$. Генератор G неявно задає розподіл p_g як розподіл $G(z)$, отриманий при $z \sim p_z$. GAN прагне до того, щоб отримати хороше наближення p_{data} – реального розподілу даних.

Для тренування генератора використовується така функція помилки:

$$\mathcal{L}_D = \mathbb{E}[\log(D(G(z)))] + \mathbb{E}[\log(1 - D(x))] \to \max_D .$$

В той час, дискримінатор тренують з застосуванням функції помилки такого вигляду:

$$\mathcal{L}_G = -\mathbb{E}[\log(D(G(z)))] \to \min_G .$$

Таким чином, можна сказати, що генератор та дискримінатор є двома гравцями гри на мінімакс з такою цільовою функцією:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}(x)} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D_{\theta_d}(G_{\theta_d}(z))) \right]. \qquad (3.1)$$

При цьому, під час тренування одної з мереж друга залишається фіксованою. Тобто, під час тренування генератора параметри дискримінатора не змінюються, і навпаки.

Оптимізацію генеративної мережі можна також представити у вигляді варіаційної нерівності, що детально розглядається в [3].

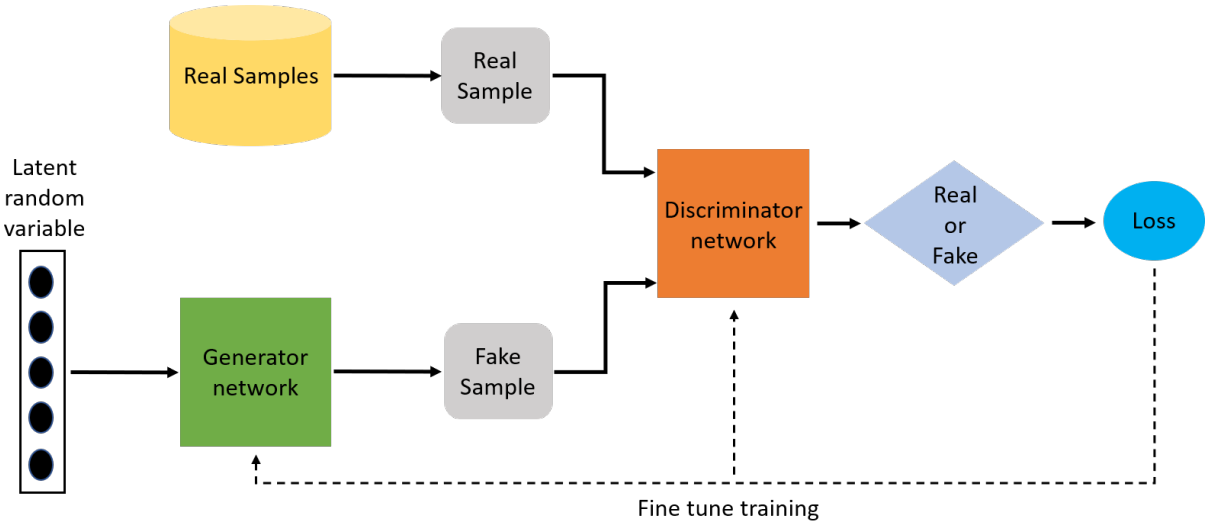


Рис. 3.1: Схема GAN, [12].

В оригінальній статті [1], де вперше запропонували модель GAN, був представлений алгоритм для тренування GAN з використанням стохастичного градієнтного спуску.

Алгоритм 3.1 (Міні-батч стохастичний градієнтний спуск)**.**

for N ітерацій **do**

for k кроків **do**

- Обираємо міні-батч з m екземплярів $\{z^{(1)}, z^{(2)}, ..., z^{(m)}\}$ з шуму.
- Обираємо міні-батч з m екземплярів $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ з наявних даних $p_{data}(x)$.
- Оновлюємо дискримінатор (його параметри), збільшуючи на стохастичний градієнт:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(x^{(i)}) + \log \left(1 - D(G(z^{(i)})) \right) \right].$$

end for

- Обираємо міні-батч з m екземплярів $\{z^{(1)}, z^{(2)}, ..., z^{(m)}\}$ з шуму.
- Тепер оновлюємо генератор зменшенням на стохастичний

градієнт

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \Big(1 - D(G(z^{(i)}))\Big).$$

3.3. Теоретичні результати.

Наведемо деякі теоретичні результати, пов'язані з генеративними змагальними мережами. Відповідні доведення можна знайти в [1].

Лема 3.1. *Якщо зафіксувати параметри θ_g генератора G, то оптимальним дискримінатором буде*

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}.$$

Тоді задачу на мінімакс 3.1 можна переформулювати в задачу з такою цільовою функцією:

$$C(G) = \mathbb{E}_{x \sim p_{data}(x)} \log D^*_{\theta_d}(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D^*_{\theta_d}(G_{\theta_d}(z))) =$$
$$= \mathbb{E}_{x \sim p_{data}(x)} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \right] + \mathbb{E}_{x \sim p_g(z)} \left[\log \frac{p_g(x)}{p_{data}(x) + p_g(x)} \right]. \quad (3.2)$$

Теорема 3.1. *Глобальний мінімум функції $C(G)$, визначеної в 3.2, досягається тоді і тільки тоді, коли $p_g = p_{data}$, причому тоді виконується рівність $C(G^*) = -\log 4$.*

Теорема 3.2 (Збіжність алгоритму.)**.** *Якщо на кожному кроці 3.1 параметри дискримінатора досягають своїх оптимальних значень за фіксованих параметрів генератора, а p_g оновлюється згідно до критерію 3.2, то p_g збігається до p_{data}.*

3.4. Оцінки якості GAN.

Важливою є також можливість оцінити якість генеративної змагальної мережі. В першу чергу зазвичай оцінюють отримані результати візуально. Тим не менш, хотілося б також мати певні чисельні характеристики

моделі, які будуть більш об'єктивними. Для цього було запропоновано декілька підходів, які працювали б для моделей GAN. Проте, немає одного універсального методу, який був би загальноприйнятим. У [6] та [9] можна знайти докладний огляд пропонованих методів оцінки генеративних змагальних мереж.

Найбільш поширеними методами оцінки генеративних змагальних мереж є Inception Score та Fréchet Inception Distance, як сказано в [9]. Проте, вони обидві використовують мережу Inception v3, яка завчасно навчена на наборі даних ImageNet з зображеннями середнього розміру 469x387, які в більшості випадків обрізаються під розмір 256x256, в той час як у даній роботі через обмеження обчислювальних можливостей було використано значно менші зображення. Тому дані способи оцінки якостей будуть давати дуже викривлені результати на наборах даних, що розглядаються. Проте, включимо і їх в загальний огляд методів оцінки генеративних змагальних мереж.

3.4.1. KL-розбіжність.

Розбіжність Кульбака-Лейблера, KL-розбіжність, або відносна ентропія, є статистичною мірою відстані між розподілами.

Нехай задані два розподіли ймовірностей: $p(x)$ та $q(x)$. KL-розбіжність $q(x)$ від $p(x)$ визначається таким чином:

$$D_{KL}(p\|q) = \int\limits_{x} p(x) \log \frac{p(x)}{q(x)} dx.$$

Чим більш схожі розподіли $p(x)$ та $q(x)$, тим менша їх KL-розбіжність. Нерівність $D_{KL}(p\|q) \geqslant 0$ виконується завжди. При цьому $D_{KL}(p\|q) = 0$ тоді і тільки тоді, коли $p = q$.

Дана міра є досить чутливої до такої проблеми у тренуванні генеративних змагальних мереж як mode collapse, і допомагає її виявляти.

3.4.2. JS-розбіжність.

JS-розбіжність, тобто розбіжність Дженсена-Шеннона, називається також радіусом інформації. У цій оцінці використовується KL-розбіжність, проте, на відміну від останньої, JS-розбіжність є симетричною відносно своїх аргументів.

Обчислити її можна таким чином:

$$D_{JS}(p\|q) = \frac{1}{2}D_{KL}\left(p\|\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q\|\frac{p+q}{2}\right).$$

При цьому $\sqrt{D_{JS}(p\|q)}$ буде метрикою, яка називається відстанню Дженсена-Шеннона.

3.4.3. Метрика Васерштейна.

k-відстанню Васерштейна (Wasserstein-k) між розподілами p і q називають

$$W_1(p,q) = \left(\inf_{\gamma\in\Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}\|x-y\|^k\right)^{1/k},$$

де $\Pi(p,q)$ – це множина відповідних спільних розподілів.

Чим нижча дана метрика, тим ближчі розподіли.

3.4.4. Inception Score.

Inception Score (або скорочено IS), що був запропонований в 2016 році у [5], є одною з найбільш поширених оцінок результату роботи моделі GAN.

При обчислюванні Inception Score враховується два фактори:

- Зображення різноманітні, тобто чи не отримуємо ми одне і те ж або дуже схожі зображення кожного разу.
- Кожне з зображень є достатньо якісним, чітко зображує якийсь з бажаних об'єктів (не є розмитим).

Inception Score буде досягати більших значень якщо обидві ці умови виконуються.

Inception Score використовує попередньо натреновану модель Inception v3. Дана модель була натренована на наборі даних ImageNet, що складається з більш ніж 10 мільйонів зображень. Для цього набору даних, кожне з більш ніж 100,000 понять з WordNet називають набором синонімів. ImageNet має в середньому близько по 1000 зображень, що відповідають кожному набору синонімів.

Inception v3 є згортковою нейронною мережею, що може класифікувати зображення на 1000 класів з ImageNet.

Для розрахунку Inception Score замість Inception v3 можна використовувати іншу нейронну мережу, яка навчена на даних з такими ж категоріями як і дані, що генерує GAN. Проте, Inception v3 є найбільш поширеним вибором, так як ця мережа працює з великою кількістю класів і є вже попередньо навченою.

Далі алгоритм обчислення Inception Score виглядає таким чином:

1. Генеруємо зображення за допомогою побудованої моделі GAN.
2. Згенеровані зображення класифікуються за допомогою попередньо навченої моделі класифікації (наприклад, Inception v3). Для кожного зображення отримується вектор ймовірностей, який вказує, наскільки ймовірно, що зображення належить до кожного класу.
3. Інформація про ймовірності класифікації зображень використовується для обчислення Inception Score.

Формула для обчислення Inception Score виглядає таким чином:

$$IS = exp\left[\mathbb{E}_{x\sim p_g}\left(D_{KL}(p(y|x)\|p(y))\right)\right].$$

Вищий Inception Score вказує на кращу якість та різноманітність зображень, що генерує GAN. Якщо генератор генерує лише один клас (наприклад, лише обличчя людини), то Inception Score буде обмежуватися лише розподілом ймовірностей цього класу. У такому випадку максимальне значення Inception Score буде наближатися до 1.

Згідно з [10], у Inception Score є декілька недоліків:

- У ньому використовується вже навчена нейронна мережа Inception v3, яка охоплює хоч і велику кількість класів, але є і не включені до неї. Моделі для інших класів можуть отримувати високі значення IS, хоча це не обов'язково відповідатиме гарній якості зображення.
- Inception Score не враховує всі аспекти якості зображень, такі як реалістичність деталей, колір, контрастність тощо.
- Inception Score може бути чутливим до розмірів зображень.

Проте, Inception Score є відносно простим в обчисленні і дозволяє відстежити як якість, так і різноманітність зображень. Тому він є поширеним варіантом оцінки якості моделі GAN.

3.4.5. Fréchet Inception Distance (FID).

Оцінка FID основана на відстані Фреше, яка є мірою несхожості між двома багатовимірними гауссівськими розподілами. При застосуванні її до генеративних змагальних мереж, обчислюється відстань Фреше між справжнім та згенерованим розподілами.

У статті [8], де був вперше запропонований даний показник, стверджують, що він відображає схожість чи різницю між справжніми та згенерованими зображеннями краще, ніж Inception Score, і є більш стійким. Ідеєю FID є порівняння статистик розподілів, а саме математичного сподівання та коваріації.

Для обчислення FID вище описана мережа Inception v3 використовується для виділення окремих ознак. Далі, обчислюються математичне сподівання та коваріація і для справжніх, і для згенерованих даних. Після чого обчислюється відстань Фреше, або Васерштейн-2 відстань.

Таким чином, формула для обчислення Fréchet Inception Distance виглядає таким чином:

$$FID(r, g) = \|m_r - m_g\|_2^2 + Tr\left(C_r + C_g - 2\left(C_r C_g\right)^{1/2}\right),$$

де m_r, m_g – математичні сподівання, а C_r, C_g – коваріації реальних та згенерованих даних відповідно.

При цьому чим нижчий показник FID, тим більш схожі порівнювані розподіли.

3.5. Застосування моделей GAN.

Розглянемо декілька варіантів застосувань Генеративних Змагальних Мереж.

- Частіше всього GANs застосовують для генерування зображень. Наприклад, навчивши мережу на наборі даних з зображень мультиплікаційних персонажів, можна отримати зображення нових, ще не існуючих персонажів.
- Перетворення фото в інше фото. Наприклад, за допомогою GANs з фото, знятого вночі, можна отримати аналогічне фото, але зняте вдень. Також можна змінити стиль картини і побачити, як би виглядала картина Моне, якби її автором був Ван Гог.
- За допомогою GANs можна з фото з низькою роздільною здатністю отримати фото у високій.
- Створення зображень по текстових описах.
- Старіння обличчя за допомогою мережі Age-cGAN.

3.6. Складнощі в тренуванні GANs

При тренуванні генеративних змагальних мереж може виникати декілька складнощів, які треба враховувати.

Перш за все, під час тренування треба зберігати баланс між поточною якістю моделей генератора і дискримінатора. Якщо якась з моделей є значно кращою, то навчання другої може ускладнитися. Наприклад, якщо модель дискримінатора є вже досить хорошою і практично завжди відрізняє

згенеровані зображення від справжніх, то генератор у якості зворотнього зв'язку завжди буде отримувати одну й ту ж відповідь ("зображення згенеровані"), градієнт його функції втрат практично зануляється, що не даватиме йому інформації для тренування. Тому в будь-який момент треба підтримувати складові мережі на приблизно однаковому рівні якості.

Ще однією поширеною проблемою при тренуванні генеративних змагальних мереж є так званий mode collapse, згаданий зокрема в [7]. Він трапляється, коли генератор починає породжувати приблизно однакові зображення, або невеликий їх набір. Наприклад, якщо мережа, що має генерувати рукописні цифри починає створювати тільки зображення з цифрою 1, не намагаючись також робити й інші. Оскільки одною з цілей GAN є генерування саме різноманітних зображень, таке явище може бути проблемою.

Також, часто генеративні змагальні мережі можуть не збігатися до оптимального стану. Коли генератор стає кращим, дискримінатор вже має труднощі з розрізненням справжніх та згенерованих зображень, а в точці рівноваги стверджує, що ймовірність кожного з варіантів 50%. Тому з часом значущість відгуку дискримінатора спадає, що заважає тренуванню генератора, яке основане на цьому відгуку.

РОЗДІЛ 4
ОБЧИСЛЮВАЛЬНИЙ ЕКСПЕРИМЕНТ

Для порівняння роботи екстраградієнтних алгоритмів для тренування генеративних змагальних мереж використаємо мережі, побудовані для набору даних MNIST, який є відносно простим і потребує менше обчислювальних можливостей, ніж багато інших. Будемо застосовувати такі алгоритми, як градієнтний та екстраградієнтний методи, екстраполяцію з минулого, екстраградієнтний варіант алгоритму Adam і власне Adam, який широко використовується для тренування нейронних мереж і, зокрема, генеративних змагальних мереж.

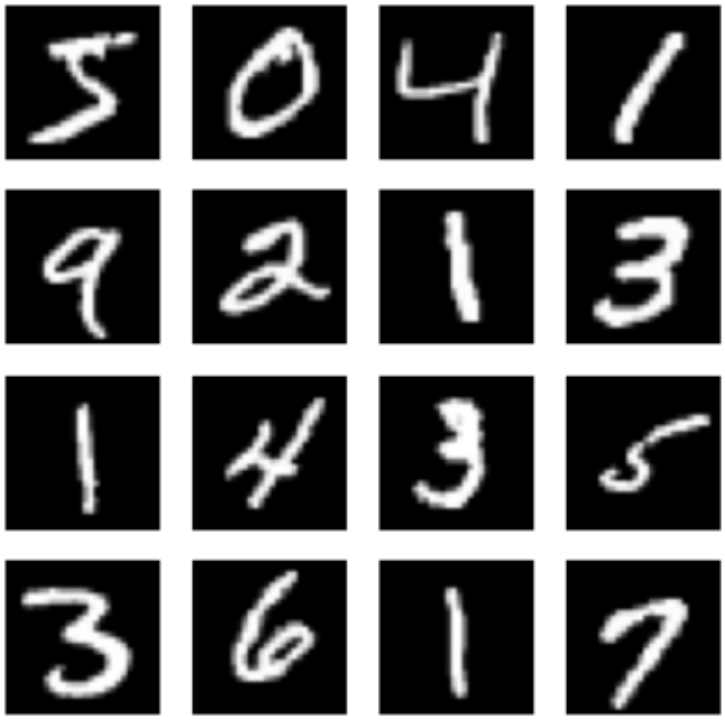


Рис. 4.1: Приклади зображень з MNIST.

4.1. Опис набору даних.

MNIST (від Modified National Institute of Standards and Technology database) – це набір даних, що складається з зображень рукописних цифр. Для кожного зображення також зберігається позначка того, яка цифра на ньому зображена. Усі зображення мають розмір 28x28 пікселей. Даний набір даних часто використовується для різних задач машинного навчання, зокрема для класифікації за допомогою нейронних мереж згортки (CNN), так як є відносно простим. У даному випадку будемо будувати генеративну змагальну мережу, яка зможе генерувати нові, аналогічні до наявних зображення з рукописними цифрами.

4.2. Архітектура GAN

У даній роботі була використана архітектура моделей генератора і дискримінатора, що описана в [11].

І генератор, і дискримінатор є нейронними мережами згортки. Архітектура кожної з них детально представлена на схемах нижче: 4.2 і 4.3 відповідають моделі генератора, а 4.4 і 4.5 – дискримінатора.

Дані розбивалися на міні-батчі по 256 зображень.

Для тренування генератора та дискримінатора було використано такі алгоритми: Екстраградієнтний метод Корпелевич, Екстраполяція з минулого, ExtraAdam, Adam.

Алгоритм Adam було використано вбудований, з бібліотеки keras, tensorflow.

Усі інші методи, екстраградієнтні, було реалізовано як класи, що наслідуються від загального класу оптимізаторів OptimizerV2 в бібліотеці keras, tensorflow. Для цього потрібно перевизначити метод $\_resource\_apply\_dense$, який відповідає за крок алгоритму, якщо тензор градієнту є щільним. Код для використаних класів можна знайти в додатку до роботи.

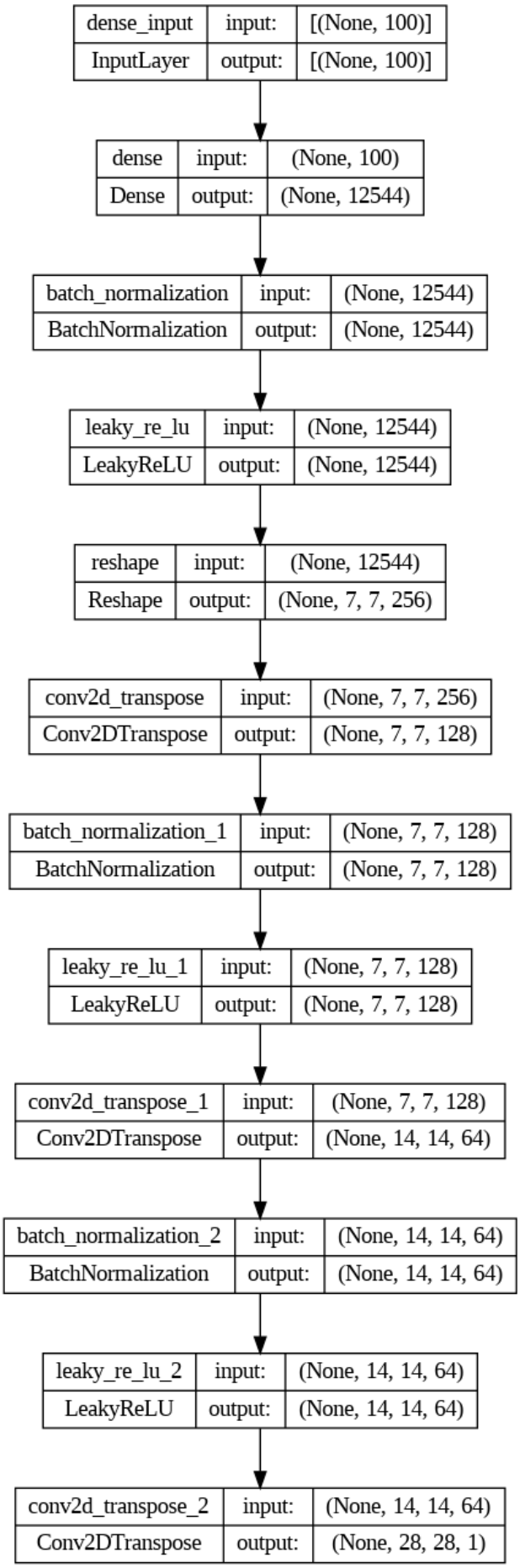


Рис. 4.2: Архітектура моделі генератора.

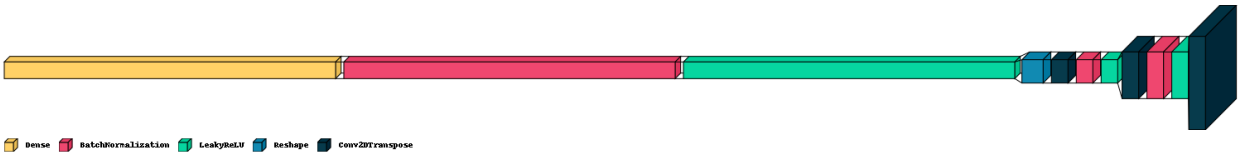


Рис. 4.3: Архітектура моделі генератора.

Для всіх методів використовувалося значення кроку $learning\_rate = 0.001$.

Також, в якості функції втрат в обох випадках була використана бінарна крос-ентропія:

$$loss = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right].$$

4.3. Результати роботи програми.

Результати після 100 епох тренування представлені на Рис. 4.6а-4.6д.

Бачимо, що за допомогою алгоритмів ExtraAdam і Adam були отримані більш правдоподібні зображення. Тим не менш, відмітимо, що хоча за допомогою алгоритму ExtraAdam вийшли хороші результати, проте тренування моделі з даним оптимізатором на кожній епосі займало майже вдвічі більше часу, ніж інші.

Подивимося тепер також на чисельні характеристики, пов'язані з якістю моделей.

Графіки функцій втрат для різних оптимізаторів зображені на Рис. 4.7.

Помітимо, що функція втрат генератора моделі з оптимізатором ExtraAdam завжди є більшою за інші. Тим не менш, зображення, згенеровані цією моделлю, є одними з найкращих. Отже, маємо підтвердження тому факту, що якість згенерованих зображень не завжди корелює зі значенням функції втрат. Також відмітимо, що у генератора моделі з оптимізатором Adam на останніх епохах значення функції втрат значно нижче за значення інших. В даному випадку, він також генерує відносно правдоподібні зображення.

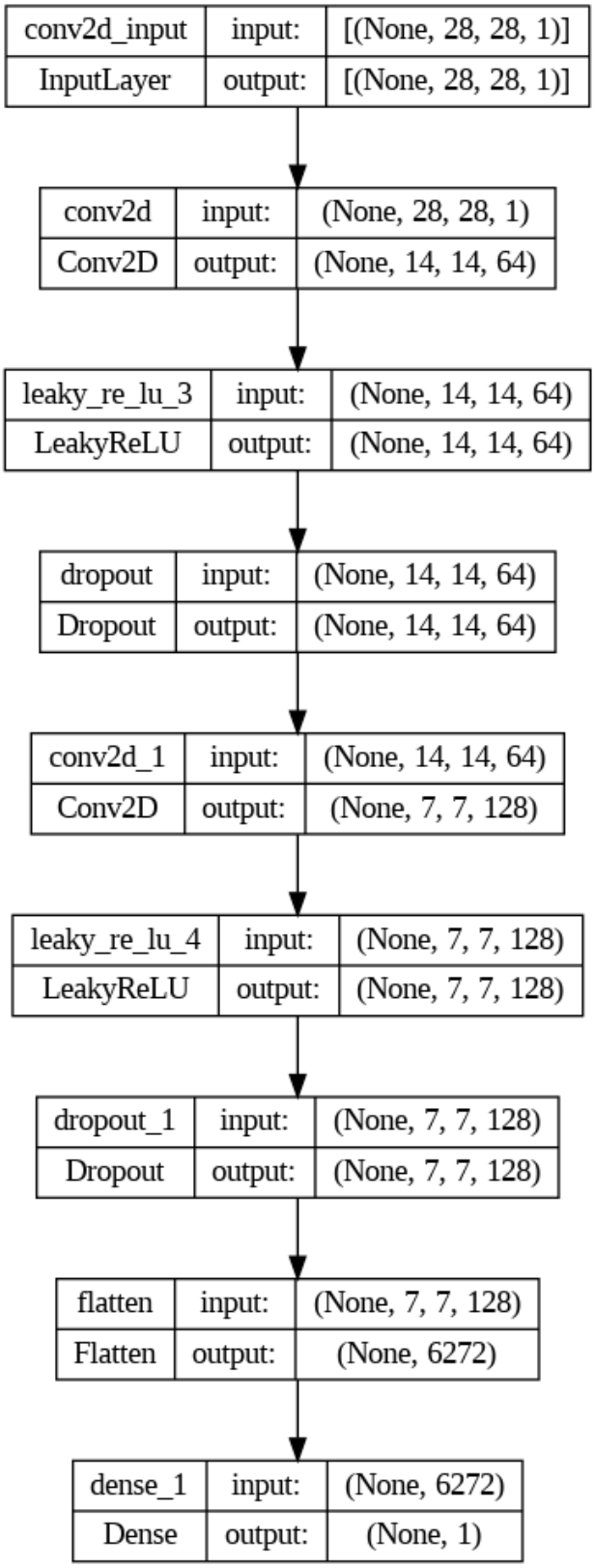


Рис. 4.4: Архітектура моделі дискримінатора.

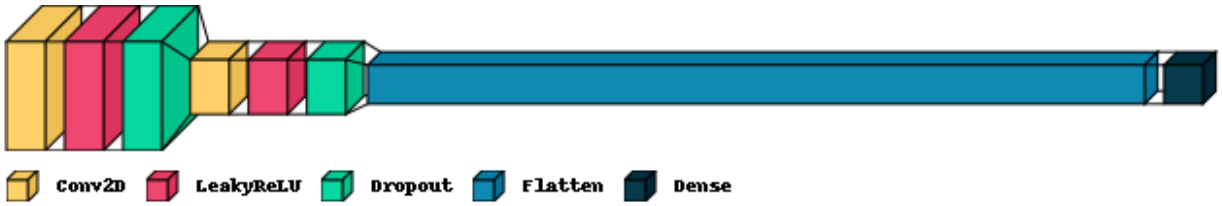


Рис. 4.5: Архітектура моделі дискримінатора.

Також, за допомогою класу $tf.keras.metrics.KLDivergence()$ були отримані наступні значення оцінки KL-розбіжності:

Градієнтний метод	0.82793444
Екстраградієнтний метод	0.93778235
Екстраполяція з минулого	0.98290235
ExtraAdam	1.0319617
Adam	0.86198443

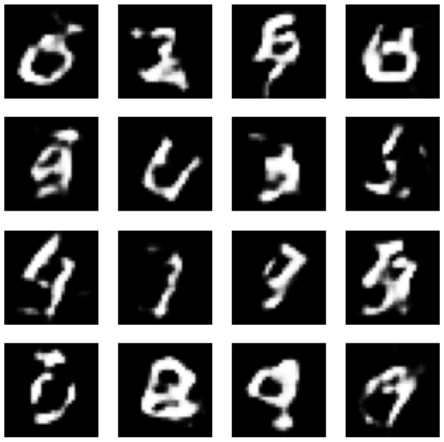
Помітимо, що ExtraAdam знову має найвищий показник, незважаючи на хорошу якість зображень. Також, найнижчого значення KL-розбіжності набуває модель з градієнтним методом, результати якої візуально не є найкращими.

Розглянемо також Васерштейн-1 відстань між кожним зі згенерованих розподілів та справжнім. Для цього використаємо вбудовану у бібліотеку scipy функцію $scipy.stats.wasserstein\_distance$.

Градієнтний метод	0.025472238659858704
Екстраградієнтний метод	0.025016184896230698
Екстраполяція з минулого	0.0167342871427536
ExtraAdam	0.0029144883155822754
Adam	0.004948500543832779

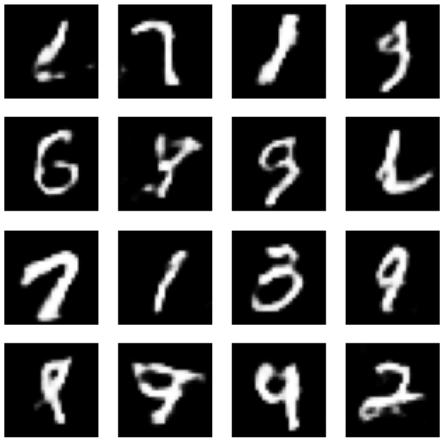
Значення цього показника вже краще узгоджуються з інтуїтивною оцінкою результатів: у ExtraAdam і Adam показники помітно нижчі, ніж у перших трьох методів. Тому схоже, що для даного набору даних Васерштейн-1 відстань є найбільш достовірною з розглянутих. При цьому помітимо, що все ж найменше значення досягається саме для моделі з оптимізаційним

(а) Зображення, отримані з GAN з градієнтним методом.

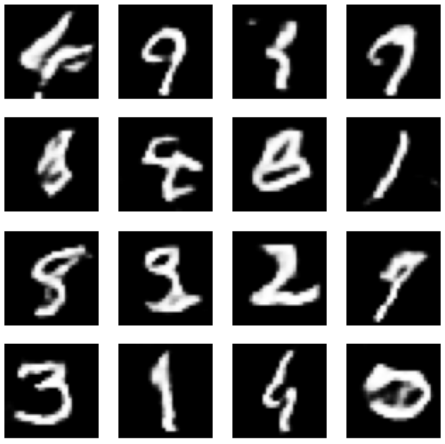


(б) Зображення, отримані з GAN з екстраградієнтним методом.

(в) Зображення, отримані з GAN з екстраполяцією з минулого.



(г) Зображення, отримані з GAN з ExtraAdam.



(д) Зображення, отримані з GAN з Adam.

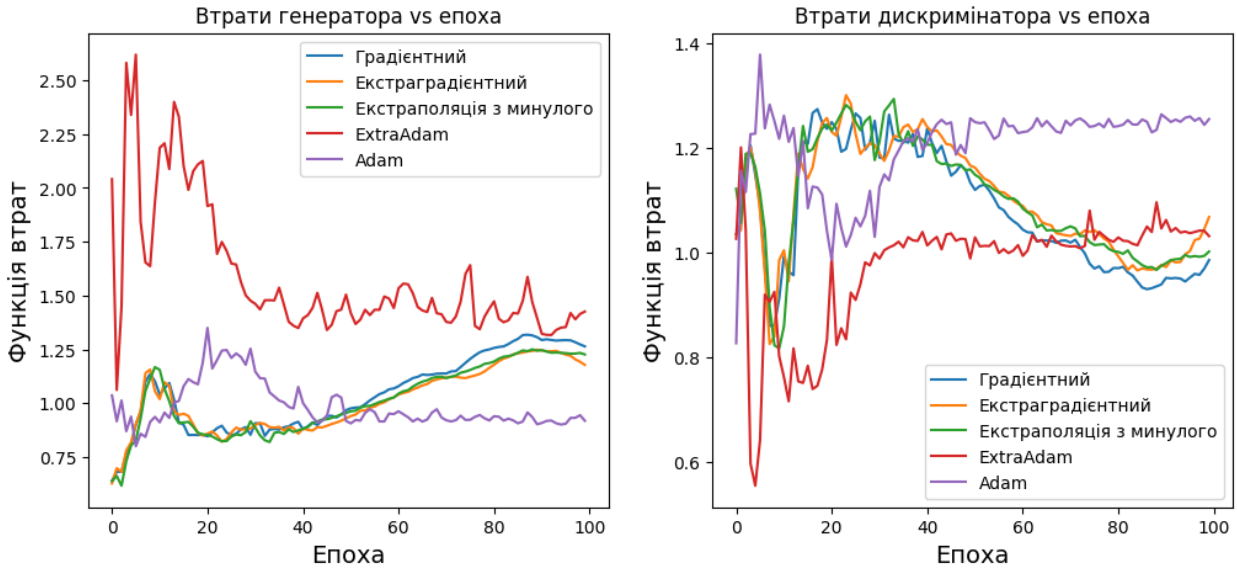


Рис. 4.7: Графіки функції втрат генератора і дискримінатора.

алгоритмом ExtraAdam.

Бачимо, що оскільки поняття "якісне зображення"досить складно описати і виразити чисельно, і найбільш важливі характеристики можуть відрізнятися залежно від конкретного набору даних, то оцінки якості моделей GAN не завжди відображають справжній рівень якості згенерованих зображень. Тому зазвичай рекомендують використовувати декілька з них.

РОЗДІЛ 5
ВИСНОВКИ

Генеративні змагальні мережі мають широкий спектр застосування і є хорошим інструментом для генерування реалістичних даних, зокрема зображень, звуку або відео.

При навчанні GAN можна використовувати різні алгоритми оптимізації, найпоширенішими з яких є стохастичний градієнтний спуск, Adam, RMSProp. Хорошим варіантом алгоритмів, що можна застосовувати, є екстраградієнтні алгоритми. Серед використаних екстраградієнтних алгоритмів найкращі результати вдалося отримати за допомогою ExtraAdam, в той час як результати від градієнтного, екстраградієнтного методів та екстраполяції з минулого були помітно гірші. Проте, навіть незважаючи на це, можна побачити, що за більшої кількості епох результати мали б покращитися. Також хороші результати отримали за допомогою стандартного методу Adam, без використання екстраполяції. Деякі числові оцінки були кращі для нього, ніж для ExtraAdam.

Проте, на розглянутому прикладі також вдалося побачити, що чисельні оцінки якості генеративних мереж не завжди добре характеризують справжню якість зображень. Це правда зокрема тому, що в різному контексті якість зображення означає різні речі (наприклад, це може бути пов'язане з кольором, чи чіткістю). Кожна з запропонованих раніше оцінок охоплює небагато таких ознак і має свої обмеження. Тому краще проводити аналіз якості моделей з використанням декількох різних оцінок якості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio: Generative Adversarial Networks. arXiv:1406.2661 (2014).
2. В.В.Семенов: Варіаційні нерівності: теорія та алгоритми : підручник. К.: ВПЦ "Київський університет"(2021).
3. Gauthier Gidel, Hugo Berard, Gaëtan Vignoud, Pascal Vincent, Simon Lacoste-Julien: A Variational Inequality Perspective on Generative Adversarial Networks. arXiv:1802.10551 (2020).
4. Korpelevich, G.M.: An extragradient method for finding saddle points and for other problems. Matecon. 12(4), 747-756 (1976).
5. Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen: Improved Techniques for Training GANs. arXiv:1606.03498 (2016).
6. Ali Borji: Pros and Cons of GAN Evaluation Measures. arXiv:1802.03446 (2018).
7. Ian Goodfellow: NIPS 2016 Tutorial: Generative Adversarial Networks. arXiv:1701.00160 (2017).
8. Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Sepp Hochreiter: GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. arXiv:1706.08500 (2018).
9. Ali Borji: Pros and Cons of GAN Evaluation Measures: New Developments. arXiv:2103.09396 (2021).
10. Mack D.: A simple explanation of the Inception Score. Medium. URL: https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a.

11. Deep Convolutional Generative Adversarial Network. TensorFlow. URL: https://www.tensorflow.org/tutorials/generative/dcgan.
12. Ankan Dash, Junyi Ye, Guiling Wang: A review of Generative Adversarial Networks (GANs) and its applications in a wide variety of disciplines - From Medical to Remote Sensing. arXiv:2110.01442 (2018).

Додаток А

Код програми.

```
import tensorflow as tf
tf.config.run_functions_eagerly(True)

from keras import backend_config
from keras.optimizers.legacy import optimizer_v2

class Extragradient(optimizer_v2.OptimizerV2):
    def __init__(
        self,
        learning_rate=0.001,
        name="Extragradient",
        **kwargs
    ):
        super().__init__(name, **kwargs)
        self._set_hyper("learning_rate", kwargs.get("lr",
        learning_rate))
        self.pvs_dict = {}
        self.num_dict = {}

    def _resource_apply_dense(self, grad, var, apply_state=None):
        var_dtype = var.dtype.base_dtype
        lr_t = self.learning_rate
        var_copy = var
```

```
            if var.name in self.num_dict:
              self.num_dict[var.name].assign_add(1)
            else:
              self.num_dict[var.name] = tf.Variable(1)

            if self.num_dict[var.name].numpy() % 2:
              var.assign_sub(lr_t * grad)
            else:
              var.assign(self.pvs_dict[var.name] - lr_t * grad)

            self.pvs_dict[var.name] = var_copy
            return var

        def _resource_apply_sparse(self, grad, var, indices,
        apply_state=None):
            raise NotImplementedError("_resource_apply_sparse
            is not implemented")


    class ExtrPast(optimizer_v2.OptimizerV2):
        def __init__(
            self,
            learning_rate=0.001,
            name="Extragradient",
            **kwargs
        ):
            super().__init__(name, **kwargs)
            self._set_hyper("learning_rate", kwargs.get("lr",
            learning_rate))
```

```
    self.pvs_dict = {}
    self.num_dict = {}
    self.prev_odd_grad = {}

def _resource_apply_dense(self, grad, var, apply_state=None):
    var_dtype = var.dtype.base_dtype
    lr_t = self.learning_rate
    var_copy = var

    if var.name in self.num_dict:
      self.num_dict[var.name].assign_add(1)
    else:
      self.num_dict[var.name] = tf.Variable(1)

    if self.num_dict[var.name] % 2:
      if var.name in self.prev_odd_grad:
          var.assign_sub(lr_t * self.prev_odd_grad[var.name])
      else:
          var.assign_sub(lr_t * grad)
    else:
       var.assign(self.pvs_dict[var.name] - lr_t * grad)

    self.pvs_dict[var.name] = var_copy

    return var

def _resource_apply_sparse(self, grad, var, indices,
apply_state=None):
    raise NotImplementedError("_resource_apply_sparse
```

```
        is not implemented")


class Gradient(optimizer_v2.OptimizerV2):
    def __init__(
        self,
        learning_rate=0.001,
        name="Extragradient",
        **kwargs
    ):
        super().__init__(name, **kwargs)
        self._set_hyper("learning_rate", kwargs.get("lr",
        learning_rate))

    def _resource_apply_dense(self, grad, var, apply_state=None):
        lr_t = self.learning_rate
        var.assign_sub(lr_t * grad)
        return var

    def _resource_apply_sparse(self, grad, var, indices,
    apply_state=None):
        raise NotImplementedError("_resource_apply_sparse is
        not implemented")


class ExtraAdam(optimizer_v2.OptimizerV2):
    def __init__(
        self,
        learning_rate=0.001,
```

```
        beta1=0.9,
        beta2=0.9999,
        epsilon=1e-7,
        name="ExtraAdam",
        **kwargs
    ):
        super().__init__(name, **kwargs)
        self._set_hyper("learning_rate", kwargs.get("lr",
        learning_rate))
        self.num_dict = {}
        self.m = {}
        self.v = {}
        self.mhat = {}
        self.vhat = {}
        self.beta1 = tf.Variable(beta1)
        self.beta2 = tf.Variable(beta2)
        self.epsilon = tf.Variable(epsilon)
        self.w_int = {}
        self.w_half = {}

    def _resource_apply_dense(self, grad, var, apply_state=None):
        var_dtype = var.dtype.base_dtype
        lr = self.learning_rate
        eps = self.epsilon
        b1 = self.beta1
        b2 = self.beta2
        var_copy = var

        if var.name in self.num_dict:
```

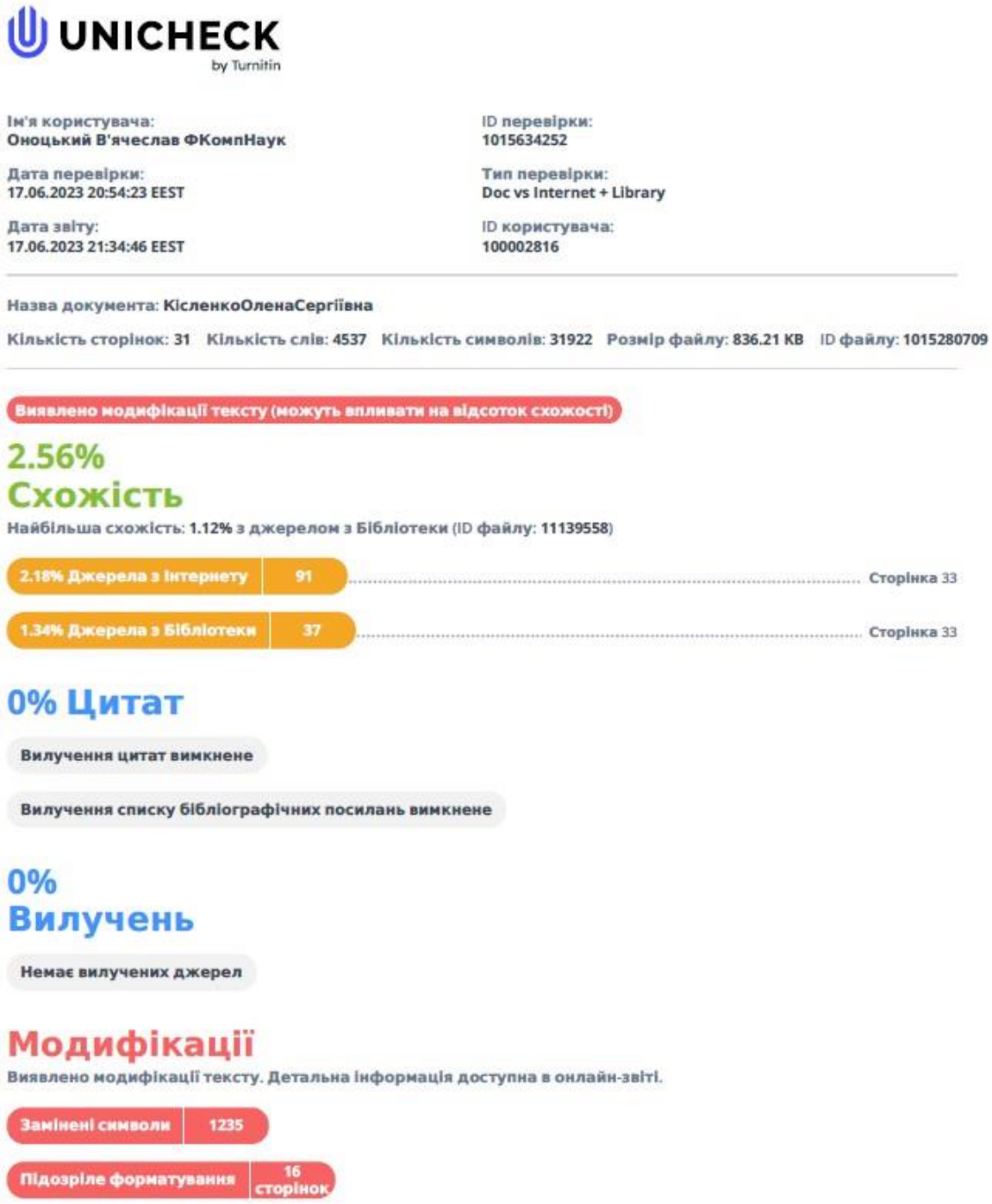
```
    self.num_dict[var.name].assign_add(1)
    self.m[var.name].assign(b1 * self.m[var.name] +
    (1 - b1) * grad)
    self.v[var.name].assign(b2 * self.v[var.name] +
    (1 - b2) * grad * grad)
    self.mhat[var.name] = tf.Variable(self.m[var.name] /
    (1 - tf.math.pow(tf.cast(b1, dtype=tf.float32),
    tf.cast(self.num_dict[var.name] - 1, dtype = tf.float32))))
    self.vhat[var.name] = tf.Variable(self.v[var.name] /
    (1 - tf.math.pow(tf.cast(b2, dtype=tf.float32),
    tf.cast(self.num_dict[var.name] - 1, dtype = tf.float32))))
  else:
    self.num_dict[var.name] = tf.Variable(1)
    self.m[var.name] = tf.Variable((1 - b1) * grad)
    self.v[var.name] = tf.Variable((1 - b2) * grad * grad)
    self.mhat[var.name] = tf.Variable(self.m[var.name] /
    (1 - b1))
    self.vhat[var.name] = tf.Variable(self.v[var.name] /
    (1 - b2))

  if self.num_dict[var.name].numpy() % 2:
    # n is odd
    if var.name in self.w_int:
      self.w_half[var.name] = tf.Variable(self.w_int[var.name] -
      lr * self.mhat[var.name] /
      (tf.math.sqrt(self.vhat[var.name]) + eps))
    else:
      self.w_half[var.name] = tf.Variable( - lr *
      self.mhat[var.name] / (tf.math.sqrt(self.vhat[var.name])
```

```
        + eps))
    else:
      # n is even
      if var.name in self.w_int:
        self.w_int[var.name] = tf.Variable(self.w_int[var.name] -
        lr * self.mhat[var.name] /
        (tf.math.sqrt(self.vhat[var.name]) + eps))
      else:
        self.w_int[var.name] = tf.Variable(- lr *
        self.mhat[var.name] / (tf.math.sqrt(self.vhat[var.name]) +
        eps))

    var.assign(self.w_half[var.name])
    return var

def _resource_apply_sparse(self, grad, var, indices,
apply_state=None):
    raise NotImplementedError("_resource_apply_sparse is not
    implemented")
```

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

СИСТЕМА ЗАПОБІГАННЯ ТА ВИЯВЛЕННЯ АКАДЕМІЧНОГО ПЛАГІАТУ

Довідка про оригінальність кваліфікаційної роботи за освітнім рівнем магістр



Експертна оцінка роботи науковим керівником :

Робота виконана самостійно та не містить відомостей без посилань на джерела.

Науковий керівник: _____ Семенов В.В.

(підпис) (ПІБ)

Оператор: _____ Оноцький В.В.

(підпис) (ПІБ)

Відгук на кваліфікаційну роботу бакалавра на тему:
«Алгоритми екстраградієнтного типу та їх застосування в машинному навчанні»
студентки 4-го курсу факультету комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка
Кісленко Олени Сергіївни

Робота Олени Кісленко присвячена дослідженню і порівнянню застосування екстраградієнтних алгоритмів оптимізації для навчання генеративних змагальних мереж (GANs). Ці методи беруть початок з екстраградієнтного алгоритму Корпелевич, запропонованого у 1976 році. Після цього багато вчених розвивали цей напрям, отримавши такі алгоритми як метод екстраполяції з минулого, в якому використовується тільки один градієнт на кожному кроці, зменшуючи складність обчислень, або метод Tseng'a.

Також, екстраградієнтні методи розглядаються як аналог звичайним градієнтним методам у навчанні генеративних змагальних мереж, які набирають популярність в останні роки. Зокрема, ідея екстраполяції в оптимізаційних алгоритмах була досліджена у статті Gauthier Gidel et al., опублікованій у 2020 році. В ній обговорюються переваги ідеї застосування екстраполяції і усереднення до відомих алгоритмів з метою позбутися коливань, присутніх при застосуванні звичайного стохастичного градієнтного методу. Через актуальність теми генерування даних, застосування екстраградієнтних алгоритмів до тренування моделей GAN також викликає інтерес.

У роботі студентка розглянула декілька класичних і екстраградієнтних методів оптимізації. У другому розділі був наведений теоретичний матеріал, що стосується генеративних змагальних мереж і способів виявити кращу з моделей. В останньому розділі було представлено порівняння результатів роботи GAN з використанням різних алгоритмів оптимізації, а також декількох їх метрик, що були отримані з обчислювального експерименту.

В процесі роботи студентка показала себе висококваліфікованим спеціалістом у галузі прикладної математики та інформатики.

Вважаю, що кваліфікаційна робота студентки відповідає вимогам, які висуваються до бакалаврських робіт, і заслуговує на оцінку «відмінно», а її автор заслуговує на присвоєння кваліфікації бакалавра.

Професор кафедри обчислювальної математики
факультету комп'ютерних наук та кібернетики
Київського національного університету
імені Тараса Шевченка,
доктор фізико-математичних наук, професор Володимир Семенов

Рецензія
на кваліфікаційну роботу бакалавра на тему:
«Алгоритми екстраградієнтного типу та їх застосування в машинному навчанні»
студентки 4-го курсу факультету комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка
Кісленко Олени Сергіївни

Метою рецензованої роботи є дослідження алгоритмів екстраградієнтного типу для навчання генеративних змагальних мереж (GAN).

Генеративні змагальні мережі були запропоновані у 2014 р. та використовуються для отримання зразків даних у відповідності з відомим розподілом. Задача навчання GAN зводиться до пошуку рівноваги Неша чи сідлової точки у спеціальних іграх.

Навчити мережу GAN за допомогою відомих у спільноті фахівців з машинного навчання варіантів градієнтного методу досить складно, оскільки зазвичай отримуються не збіжні траєкторії. Тому в окремих дослідницьких групах почали розглядати навчання GAN як варіаційну нерівність та експериментувати з методами екстраградієнтного типу. Рецензована робота вкладається у цей напрям досліджень.

Перша частина роботи є коротким оглядом поширених екстраградієнтних методів та їх модифікацій для навчання моделей GAN. У другій частині наведений опис генеративних змагальних мереж. Також присутній огляд різних метрик для оцінки моделі GAN. Остання частина присвячена обчислювальним експериментам з використанням відкритого набору даних MNIST, аналізу результатів і отриманих характеристик.

За змістом роботи можна зробити зауваження, але не вважаю його таким, що зменшує загальну позитивну оцінку роботи. У подальшій роботі варто також присвятити увагу теоретичним гарантіям збіжності алгоритмів за умовах, присутніх у реальних задачах навчання GAN.

Вважаю, що кваліфікаційна робота студентки Олени Кісленко відповідає вимогам, які висуваються до бакалаврських робіт, і заслуговує на оцінку «відмінно» (98), а її автор заслуговує на присвоєння кваліфікації бакалавра.

Доцент кафедри моделювання складних систем
факультету комп'ютерних наук та кібернетики
Київського національного університету
імені Тараса Шевченка,
доктор фіз.-мат. наук, доцент Андрій ШАТИРКО