МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____
"_____"_____2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: « Програмний засіб перевірки безпеки інформаційних систем засобами нейромережі *GAN* »

Виконавець: студент Гарда Антон Сергійович

Керівник: доцент Кучерява Ольга Миколаївна

Нормоконтролер: Тупота Є.В.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп’ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Литвиненко О. Є.

«_____»_____2020 р.

ЗАВДАННЯ
на виконання дипломної роботи (проекту)

Гарди Антона Сергійовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проекту): Програмний засіб перевірки безпеки інформаційних систем засобами нейромережі *GAN*

затверджена наказом ректора від «_____»_____202__ р. №___

2. Термін виконання роботи (проекту): з _____ по _____

3. Вихідні дані до роботи (проекту): мова програмування Python, фреймворк *TensorFlow*, дистрибутиву *Anaconda*, пакет *Matplotlib*, алгоритм *Improved Training of Wasserstein GAN*, метод дивергенції Йенсена-Шеннона.

4. Зміст пояснювальної записки: _____

5. Перелік обов'язкового графічного (ілюстративного) матеріалу: структурна схема програмного засобу, зв’язки між рівнями програмного засобу, послідовність виконання функцій програмного засобу, схема алгоритму, робота процедури дискримінатора та генератора.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання

7. Дата видачі завдання: “_____"_____ 2020 р.

Керівник дипломної роботи _____Кучерява О.М.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____Гарда А.С.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Програмний засіб перевірки безпеки інформаційних систем засобами нейромережі *GAN*”: -- с., -- рис., -- літературних джерел.

Ключові слова: *GAN, TENSORFLOW, SYSTEM SECURITY, DISCRIMINATOR, GENERATOR*

Об’єкт дослідження – модуль перевірки безпеки інформаційних систем.

Предмет дослідження – засоби безпеки інформаційних систем.

Мета дипломної роботи – розробка програмного засобу перевірки безпеки інформаційних систем засобами нейромережі *GAN*.

Методи дослідження – набір інструментів та команд мови програмування *Python* за допомогою фреймворку *TensorFlow*. Віртуальне середовище створено на основі дистрибутиву *Anaconda*. Для візуалізація процесу тренування використано пакет *Matplotlib*. Для методу виміру двух схожих ймовірностей використано метод дивергенції Йенсена-Шеннона. Для оптимізації навчання обрано метод *Adam*. Для структури програмного засобу використано покращений *GAN*, а саме – *Improved Training of Wasserstein GAN*.

Прогнозні припущення щодо розвитку об’єкта дослідження – створення програмного продукту та використання його для перевірки безпеки інформаційних систем.

Результати дипломної роботи рекомендується використовувати при розробці нових програмних засобів, призначених для перевірки надійності паролів користувачів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

OpenCV	–	*Open-Source Computer Vision Library;*
API	–	*Application Programming Interface*;
LBPH	–	*Local Binary Patterns Histogram*;
OpenBR	–	*Open-Source Biometric Recognition*;
ЕОМ	–	Електронна обчислювальна машина;
FLDA	–	*Fisher's Linear Discriminant Analysis*;
RGB	–	Червоний, зелений, синій (з англ. *Red, Green, Blue*) - адитивна колірна модель;
SVM	–	Метод опорних векторів (з англ. *support vector machines*);
SURF	–	Прискорені стійкі ознаки (англ. *Speeded up Robust Features*);
ЗМІ	–	Засоби масової інформації.

ВСТУП

Актуальність теми. Проблема інформаційної безпеки набула особливої значущості в сучасних умовах широкого застосування автоматизованих інформаційних систем, заснованих на використанні комп’ютерних і телекомунікаційних засобах.

У зв’язку із зростаючою роллю інформаційних ресурсів у житті сучасного суспільства, а також через реальність численних загроз проблема інформаційної безпеки вимагає до себе постійної і значної уваги. Останні десятиліття порушення захисту інформації прогресує з використанням програмних засобів і через глобальну мережу Інтернет. Досить поширеною загрозою інформаційної безпеки є зараження комп’ютерних систем вірусами. Інформаційна безпека – це стан захищеності інформаційного середовища, захист інформації являє собою діяльність щодо запобігання витоку інформації, що захищається, несанкціонованих і ненавмисних впливів на інформацію, що захищається, тобто процес, спрямований на досягнення цього стану. Загрози інформаційній безпеці – це можливі дії або події, які можуть вести до порушень інформаційної безпеки. Види загроз інформаційній безпеці дуже різноманітні і мають безліч класифікацій. При розробці засобів, методів і заходів, що забезпечують захист інформації, необхідно враховувати велику кількість різних факторів.

При забезпеченні інформаційної безпеки комп'ютерних систем необхідно враховувати, що обмін інформацією є найпершою умовою життєдіяльності кожної організації. Для протидії злочинам в інформаційній сфері або хоча б зменшення шкоди необхідно грамотно вибирати заходи і засоби забезпечення захисту комп'ютерної інформації від навмисного руйнування, крадіжки, псування, несанкціонованого доступу, несанкціонованого читання і копіювання. Важливою проблемою забезпечення безпеки інформаційних ресурсів комп'ютерних систем є задача обмеження кола осіб, що мають доступ до конкретної інформації і захисту її від несанкціонованого доступу.

В інформаційній сфері історично сформувалися два напрямки захисту від несанкціонованого доступу. У системах фізичного захисту вони називаються системами управління доступом, а в комп'ютерній сфері – системами ідентифікації і аутентифікації.

Ідентифікацію та аутентифікацію можна вважати основою програмнотехнічних засобів безпеки тому, що інші сервіси розраховують на обслуговування іменованих суб'єктів. Ідентифікація та аутентифікація – це перша лінія захисту, «вхід» інформаційного простору комп'ютерної системи. Саме від правильності рішення цих двох завдань залежить, чи можна дозволити доступ до ресурсів комп'ютерної системи конкретного користувача. Засоби ідентифікації (розпізнавання користувача за пред'явленим ідентифікатором) і аутентифікації (встановлення автентичності ідентифікованого користувача) відносяться до категорії класичних механізмів управління доступом користувачів та інформаційної безпеки комп'ютерних систем і мереж. Система захисту виконує ідентифікацію та аутентифікацію на основі певної унікальної інформації, яка характеризує конкретного користувача системи.

Є декілька підходів до вирішення завдання ідентифікації і аутентифікації користувачів. Парольний підхід – використовує унікальне знання, наприклад, логін-пароль. Апаратний або електронний – використовує унікальний предмет, наприклад, смарт-карти, магнітні карти, токени. Біометричний – використовує унікальні характеристики людини, такі як відбитки пальців, сітківка ока, голос.

У кожного з підходів є свої переваги і недоліки. Однозначного рішення не існує, користувачам доводиться самостійно вибирати, який спосіб ідентифікації реалізовувати в своїх інформаційних комп'ютерних системах. Система ідентифікації і аутентифікації є одним з ключових елементів інфраструктури захисту від несанкціонованого доступу до будь-якої інформаційної комп’ютерної системи.

На сьогоднішній день парольна ідентифікація та аутентифікація є найпоширенішим способом визначення особи користувача. Даний спосіб найбільш простий як у реалізації, так і у використанні. Суть парольної ідентифікації або аутентифікації зводиться до наступного. Кожен зареєстрований

користувач системи одержує набір персональних реквізитів, наприклад, пароль та логін. При кожній спробі входу користувач повинен вказати свою інформацію. Оскільки інформація унікальна для кожного користувача, то на підставі її система робить висновок про особу та ідентифікує її. Головна перевага парольної ідентифікації – це простота реалізації. Введення парольної ідентифікації не вимагає витрат, даний процес реалізований у більшості програмних продуктів. Таким чином, система захисту інформації виявляється простою і доступною. Тепер перейдемо до недоліків. Головний недолік це залежність надійності ідентифікації від самих користувачів, точніше, від обраних ними паролів. Більшість людей використовують ненадійні ключові слова, які легко підбираються. До них відносяться занадто короткі паролі, загальновідомі сполучення символів та інше. Тому деякі фахівці в області інформаційної безпеки радять використати довгі паролі, що складаються з випадкового сполучення букв, цифр і різних символів. Оскільки володіння вірним паролем майже завжди гарантує зловмиснику владу над інформаційною системою, атаки на парольні системи є найбільш поширеними формами дій інформаційних порушників. Для досягнення цієї мети зловмисники часто звертаються до спеціальних програм, що спрямовані на «злам» парольної системи. При правильному використанні паролі можуть забезпечити прийнятний для багатьох організацій рівень безпеки. Проте, по сукупності характеристик їх слід визнати найслабкішим засобом перевірки достовірності. Саме слабкість парольного захисту є однією з основних причин уразливості комп'ютерних систем до спроб несанкціонованого доступу. Розробка дипломної роботи спрямована на перевірку безпеки паролів користувачів, що дозволить вчасно виявити слабкі паролі та замінити їх, упередити можливі наслідки від використання слабких паролів.

Мета і завдання виконання дипломної роботи. Мета дипломної роботи – розробка програмного засобу перевірки безпеки інформаційних систем засобами нейромережі *GAN*. Даний програмний засіб дає можливість компаніям, приватним особам на основі бази паролів виконати тренування моделі, згенерувати тестові дані для перевірки надійності паролів користувачів.

Для досягнення поставленої мети необхідно виконати наступні завдання: розробити модуль тренування моделі, розробити модуль генерації вихідних данних, розробити модуль підготовки вхідних даних, розробити архітектуру для моделі, розробити інтерфейс користувача, проаналізувати сучасні методи перевірки інформаційної безпеки. Проаналізувати архітектуру нейромереж для перевірки безпеки інформаційних систем. Детально оглянути мережу *GAN*, програмно реалізувати модель на основі нейромережі *GAN*, обрати оптимальні параметри для тренування моделі. Проаналізувати вхідні та вихідні дані, перевірити на тестовому наборі хешів. Виконати перевірку вихідних даних з правилами. Порівняти з існуючими програмними засобами. Обрати середовище розробки, мови програмування, програмні інструменти та інші засоби для розробки програмного продукту.

Об’єкт і предмет дослідження. Об’єкт дослідження – модуль перевірки безпеки інформаційних систем. Предмет дослідження – засоби безпеки інформаційних систем.

Методи дослідження. В ході розробки програмного засобу для досягнення поставленої мети використовується набір інструментів та команд мови програмування *Python* за допомогою фреймворку *TensorFlow*. Віртуальне середовище створено на основі дистрибутиву *Anaconda*. Для візуалізація процесу тренування використано пакет *Matplotlib*. Для методу виміру двух схожих ймовірностей використано метод дивергенції Йенсена-Шеннона. Для оптимізація навчання обрано метод *Adam*. Для структури програмного засобу використано покращений *GAN*, а саме – ***Improved Training of Wasserstein GAN***.

Наведені нижче функції не входять до сфери застосування, оскільки неможливо включити їх до проекту, оскільки це було б занадто великим, щоб кодувати їх усі. Детальний аналіз зв’язків внутрішньої структури моделі, фільтрування вхідних даних, тренування на основі декількох моделей, детальне математичне пояснення досліджуваних алгоритмів.

Наукова новизна отриманих результатів. У роботі представлено техніку вгадування пароля, засновану на генеративних змагальних мережах (*GAN*). Програмний засіб призначений для вивчення інформації про розповсюдження

паролів через витоки паролів. Як результат, на відміну від поточних інструментів вгадування паролів, програмний засіб не покладається на будь-яку додаткову інформацію, таку як явні правила, або припущення щодо марковійської структури вибраних користувачем паролів. Даний підхід до вгадування паролів є революційним, оскільки програмний засіб генерує паролі без втручання користувача – таким чином, не вимагаючи знання предметної області щодо паролів, а також ручного аналізу витоків бази даних паролів. Результати показують, що програмний засіб конкурує з найсучаснішими інструментами генерації паролів. Програмний засіб завжди міг генерувати таку ж кількість збігів, як і інші інструменти вгадування паролів. Однак, в даний час вимагає виведення більшої кількості паролів порівняно з іншими інструментами. Ця вартість є незначною, враховуючи переваги запропонованої техніки. Крім того, навчання на більшому наборі даних дозволяє використовувати більш складні нейромережеві структури та більш всебічне навчання. Як результат, базовий *GAN* може виконати більш точну оцінку щільності, тим самим зменшуючи кількість паролів, необхідних для досягнення певної кількості збігів.

Розроблений програмний засіб можливо використовувати в системах перевірки інформаційної безпеки, або як окремий програмний засіб, наприклад, для перевірки безпеки політики паролів в організації.

РОЗДІЛ 1
ОГЛЯД МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ ПЕРЕВІРКИ БЕЗПЕКИ ІНФОРМАЦІЙНИХ СИСТЕМ ЗАСОБАМИ НЕЙРОМЕРЕЖ

1.1. Актуальність і доцільність технології перевірки безпеки інформаційних систем

Інформаційна безпека – це стан захищеності систем обробки і зберігання даних, при якому забезпечено конфіденційність, доступність і цілісність інформації, використання й розвиток в інтересах громадян або комплекс заходів, спрямованих на забезпечення захищеності інформації особи, суспільства і держави від несанкціонованого доступу, використання, оприлюднення, руйнування, внесення змін, ознайомлення, перевірки запису чи знищення.

Поняття інформаційної безпеки не обмежується безпекою технічних інформаційних систем чи безпекою інформації у чисельному чи електронному вигляді, а стосується усіх аспектів захисту даних чи інформації незалежно від форми, у якій вони перебувають.

Метою реалізації інформаційної безпеки будь-якого об'єкта є побудова системи забезпечення інформаційної безпеки даного об'єкту.

1.1.1 Області застосування інформаційної безпеки

В залежності від контексту термін «інформаційна безпека» вживається в різних значеннях. У найширшому сенсі поняття має на увазі захист конфіденційних відомостей, виробничого процесу, інфраструктури компанії від навмисних або випадкових дій, які призводять до фінансових збитків або втрати репутації.

Принципи захисту інформації. У будь-якій галузі базовий принцип інформаційної безпеки полягає в дотриманні балансу інтересів громадянина,

суспільства і держави. Труднощі дотримання балансу полягає в тому, що інтереси суспільства і громадянина нерідко конфліктують. Громадянин прагне зберегти в секреті подробиці особистого життя, джерела і рівень доходу. Суспільство, навпаки, зацікавлена в тому, щоб «розсекретити» інформацію про нелегальні доходи, факти корупції, злочинні діяння. Держава створює і керує стримуючим механізмом, який охороняє права громадянина на нерозголошення персональних даних та одночасно регулює правовідносини, пов'язані з розкриттям злочинів і притягненням винних до відповідальності.

Важливе значення в сучасних умовах принцип правового забезпечення інформаційної безпеки набуває, коли нормативне супроводження не встигає за розвитком галузі інформаційної безпеки. Прогалини в законодавстві дозволяють не тільки уникнути відповідальності за кіберзлочини, але і перешкоджають впровадженню передових технологій захисту даних.

Принцип глобалізації, або інтеграції систем інформаційного захисту зачіпає всі галузі: політичну, економічну, культурну. Розвиток міжнародних комунікаційних систем потребує узгоджене забезпечення безпеки даних.

Згідно з принципом економічної доцільності, ефективність заходів щодо забезпечення інформаційної безпеки повинна відповідати або перевершувати витрачені ресурси.

Принцип гнучкості систем інформаційного захисту означає усунення будь-яких режимних обмежень, які заважають генерувати і впроваджувати нові технології. Сувору регламентацію конфіденційної, а не відкритої інформації передбачає принцип несекретності.

Чим більше різних апаратних і програмних інструментів безпеки застосовують для захисту даних, тим більше різнобічних знань і навичок потрібно зловмисникам, щоб виявити уразливості і обійти захист. На посилення інформаційної безпеки спрямований принцип різноманітності захисних механізмів інформаційних систем.

Принцип простоти управління системою безпеки заснований на ідеї, що чим складніше система інформаційної безпеки, тим важче перевірити узгодженість роботи окремих компонентів і реалізувати центральне

адміністрування.

Запорукою лояльного ставлення персоналу до ІБ є постійне навчання правилам інформаційної безпеки і чіткі роз'яснення наслідків недотримання прави. Принцип лояльності адміністраторів систем безпеки даних і всього персоналу компанії пов'язує забезпечення безпеки з мотивацією співробітників. Якщо співробітники, а також контрагенти і клієнти сприймають інформаційну безпеку як непотрібне або навіть вороже явище, гарантувати безпеку інформації в компанії не під силу навіть найкращим системам.

Перераховані принципи – основа забезпечення інформаційної безпеки в усіх галузях, яка доповнюється елементами в залежності від специфіки галузі.

Банківська сфера. Розвиток технологій кібератак змушує банки впроваджувати нові і постійно вдосконалювати базові системи безпеки. Мета розвитку інформаційної безпеки в банківській сфері - виробити такі технологічні рішення, які здатні убезпечити інформаційні ресурси і забезпечити інтеграцію новітніх продуктів в ключові бізнес-процеси фінансово-кредитних установ.

Необхідність слідувати різним законам і стандартам пов'язана з тим, що банки здійснюють безліч різних операцій, ведуть діяльність за різними напрямками, які потребують власні інструменти забезпечення безпеки. Наприклад, забезпечення інформаційної безпеки при дистанційному банківському обслуговуванні включає створення інфраструктури безпеки, куди входять засоби захисту банківських додатків, контролю потоків даних. моніторингу банківських транзакцій і розслідування інцидентів. Багатокомпонентний захист інформаційних ресурсів забезпечує мінімізацію загроз, пов'язаних з шахрайством при використанні сервісів дистанційного банківського обслуговування, а також захист репутації банку.

Інформаційна безпека банківської сфери, як і інших галузей, залежить від кадрового забезпечення. Особливість ІБ в банках полягає в підвищеній увазі до фахівців з безпеки на рівні регулятора.

Енергетика. Енергетичний комплекс належить до числа стратегічних галузей, які потребують особливих заходів забезпечення інформаційної безпеки. Якщо на робочих місцях в адміністраціях і управліннях достатньо стандартних

засобів ІБ, то захист на технологічних ділянках генерації енергії і доставки кінцевим користувачам потребує підвищеного контролю. Головним об'єктом захисту в енергетичній сфері є не інформація, а технологічний процес. Система безпеки в такому випадку повинна забезпечити цілісність технологічного процесу і автоматизованих систем управління.

Значимість інформаційної безпеки в енергетичній сфері визначається наслідками реалізації інформаційних кіберзагроз. Це не тільки матеріальні збитки або удар по репутації, але перш за все - шкода здоров'ю громадян, підрив екології, порушення інфраструктури міста або регіону.

Проектування системи інформаційного захисту в сфері енергетики починається з прогнозування та оцінки ризиків безпеки. Основний метод оцінки – моделювання можливих загроз, яке допомагає раціонально розподілити ресурси при організації системи безпеки і попередити реалізації кіберзагроз. Крім того, оцінка ризиків безпеки в енергетиці відрізняється безперервністю: аудит в процесі експлуатації системи ведеться постійно, щоб своєчасно змінювати налаштування для забезпечення максимального ступеня захисту і підтримки системи в актуальному стані.

Засоби масової інформації. Головне завдання інформаційної безпеки в ЗМІ полягає в захисті національних інтересів, включаючи інтереси громадян, суспільства і держави. Діяльність засобів масової інформації в сучасних умовах зводиться до створення інформаційних потоків у вигляді новин та журналістських матеріалів, які надходять, обробляються і видаються кінцевим споживачам: читачам, глядачам, відвідувачам сайтів.

Забезпечення і контроль безпеки в сфері масової інформації реалізується за кількома напрямками і включає:

- розробку рекомендацій щодо антикризових процедур на випадок реалізації загрози інформаційного нападу;
- навчальні програми з інформаційної безпеки для співробітників редакцій ЗМІ, прес-служб, відділів по зв'язках з громадськістю;
- тимчасове зовнішнє адміністрування організації, які зазнали інформаційного нападу.

Ще однією проблемою безпеки інформації в ЗМІ є необ'єктивність. Для забезпечення об'єктивного висвітлення подій потрібен механізм захисту, який захищав би журналістів від тиску з боку представників влади, керівництва або власника ЗМІ, та страхував сумлінні бізнес-структури від дій нечесних представників ЗМІ.

Ще одним наріжним каменем забезпечення безпеки інформації в сфері ЗМІ є обмеження доступу до даних. Проблема в тому, щоб обмеження доступу до відомостей з метою попередити інформаційні загрози не стало «прикриттям» для цензури. Рішення, яке зробить роботу ЗМІ більш прозорою і допоможе уникнути заподіяння шкоди інтересам національної безпеки, міститься в проекті Конвенції про доступ до інформаційних ресурсів, яка очікує голосування в Європейському союзі. Норми документа припускають, що держава забезпечує рівний доступ до всіх офіційних документів, створюючи відповідні реєстри в інтернеті, і встановлює обмеження досітупу, які не можуть бути змінені. Винятків, які дозволять скасувати обмеження доступу до інформаційних ресурсів, всього два:

– суспільна користь, можливість оприлюднити навіть ті дані, які не підлягають поширенню в звичайних умовах;
– національний інтерес, якщо приховування відомостей завдасть збитків державі.

1.1.2. Напрямки розвитку інформаційної безпеки

Забезпечення інформаційної безпеки стає складнішим й більш актуальним в міру розвитку цифрового світу. Нові напрямки захисту інформації від розробників переходять в сферу практичного застосування.

Складові інформаційної безпеки. Захист інформації актуальний і для держави в цілому, і для окремих галузей економіки або компаній. З практичної точки зору поняття інформаційної безпеки змінюється в міру змін в світі, деякі напрямки захисту інформації відходять у минуле, виникають нові. Окремим завданням стає протидія прямим інформаційним втручанням в діяльність країни або бізнесу. Наукові дослідження в галузі захисту інформації перестають бути

теоретичними.

Завдання ускладнюються, наукові дослідження проводять у нових сферах, серед яких:

– геополітичне домінування;
– розробка кіберзброї;
– криптовалюта і маркери;
– телемедицина;
– середовище віртуалізації.

Вітчизняні розробники не завжди встигають за розвитком технологій, але щороку на ринку з'являються нові продукти, що дозволяють вирішити завдання інформаційної безпеки.

Захист інформації не є завданням лише компанії. Головною метою Доктрини інформаційної безпеки названа спільна охорона інтересів особистості, суспільства і країни в інформаційній сфері. Діяльність організацій в області захисту інформації спирається на створену державою міцну основу у вигляді контролю за сектором Інтернету, розвитку вітчизняного ринку програмного забезпечення та електронного обладнання. У цих умовах з'являються нові основні напрямки інформаційної безпеки. Експерти, що працюють в цій сфері, кажуть, що в найближчі п'ять років ситуація зміниться ще більш радикально. З урахуванням швидкого розвитку в сфері захисту інформації зараз виділяється кілька актуальних напрямків.

Хмарна безпека. Багато компаній відмовляються від зберігання інформації на серверах і переміщають її в хмару. У хмарі найчастіше знаходиться інформація, що обробляється при роботі програмних продуктів для малого бізнесу, бухгалтерських та *CRM*-систем. При розміщенні в хмарних системах персональних даних, виникає питання про їх захищеність відповідно до вимог законодавства та можливості несанкціонованого доступу до інформації. Це завдання поки не вирішується на законодавчому рівні і стає питанням приватної домовленості між підприємством і власником хмарної системи зберігання.

У цій сфері існує безліч невирішених питань:

– нормативно-правове регулювання;

– технічне забезпечення, розробка нових засобів захисту інформації;

– організаційна взаємодія.

Хмарні додатки надзвичайно динамічні і потребують автоматизованої системи безпеки. Технологія безпечного доступу (*SASE, Secure Access Service Edge*) дозволяє підприємствам краще захищати мобільних співробітників і хмарні додатки шляхом маршрутизації трафіку через хмарні рішення, ніж в «класичному» варіанті обробки вхідного трафіку у власному дата-центрі. Регулювання захисту інформації в хмарі і забезпечення безпеки даних має стати одним з найважливіших напрямків розвитку інформаційної безпеки для корпоративного сектора найближчим часом.

Розробка кіберзброї. Захист інформації базується не тільки на створенні декількох рівнів безпеки, а й на розробці превентивних заходів, які знизять ризик посягання на критично важливі системи. Розробка власного кіберзброї гарантує забезпечення безпеки, так як ризик удару може виявитися серйозним. Тому при наявності власних інструментів нападу кількість внутрішніх і зовнішніх загроз може знизитися.

Безпека медичних систем. Новим рішенням в охороні здоров'я стала телемедицина або надання послуг в режимі телеконференції. Передані дані не завжди захищені належним чином, що викликає необхідність вирішити це завдання.

Новий напрямок безпеки торкнеться не тільки захисту персональних даних громадян, а й систем управління медичними установами, каналами зв'язку, каналами передачі документованої медичної інформації, наприклад, між медичним центром і лабораторією, яка проводить дослідження.

Комплексна безпека медичних систем стає нагальною потребою при охороні інтересів особистості.

Безпека мобільних пристроїв. Багато громадян не передбачають, що їх смартфон – це інформаційна бомба, яка містить повну і вичерпну інформацію про людину, його пересування, купівлі, зв'язки, захоплення. Не тільки потрапляння телефону в чужі руки, але і проникнення в нього вірусу загрожують конфіденційності інформації.

Основна частка ринку щодо захисту мобільних пристроїв доводиться на великих гравців, але окремі антивірусні рішення і засоби захисту інформації розробляють невеликими компаніями або науково-дослідними організаціями.

Безпека в фінансово-кредитній сфері. Фішинг та інші шахрайські методи списання коштів з платіжних карт громадян стали повсякденною реальністю. Інформаційні системи банків постійно страждають від хакерських атак.

Розвиток інформаційної безпеки у фінансовій сфері актуальний і для банків, і для компаній, що бажають додатково убезпечити активи і масиви конфіденційної інформації, що містять архів фінансових транзакцій. Ці ж заходи, технічні та апаратні засоби, можуть бути використані для захисту електронної комерції.

Протидія загрозам передбачає:

– захист каналів віддаленого доступу, трафіку передачі фінансової конфіденційної інформації;
– створення довіреного середовища на апаратних засобах клієнта за допомогою *TrustScreen* або *Mac*-токенів;
– розробку і впровадження процесів боротьби з шахрайством, спрямованим на розкрадання коштів;
– моніторинг всіх транзакцій.

Інформаційна безпека криптовалют, токенів та смарт-контрактів. Майнинг криптовалют став повсякденним явищем. Токени випускають і фермерські господарства, ігрові проекти. З'являється національне нормативно-правове регулювання сфери. Широкий розвиток цього напрямку вимагає паралельного розвитку і засобів захисту. Поки більшість електронних гаманців знаходиться за кордоном, але з розвитком національної інфраструктури захист інформації в області електронних активів стане істотною потребою.

1.2. Використання машинного навчання для інформаційної безпеки

Машинне навчання має вирішальне значення для захисту даних при масштабних порушеннях:

Пошук мережевих загроз. Постійно відстежуючи основи даних на наявність аномалій або порушень, алгоритми машинного навчання можуть ефективно виявляти та стримувати загрози. Здатність машинного навчання обробляти дані в режимі реального часу надзвичайно корисна, оскільки дозволяє виявляти загрози, інсайдерські порушення та зловмисне програмне забезпечення в міру їх виникнення, запобігаючи величезним втратам.

Захист хмарних даних. Організації все частіше переносять свої бази даних у хмару, щоб зменшити навантаження на зовнішні сервери та клопоти з обслуговуванням. Машинне навчання може допомогти захистити дані, що зберігаються в хмарі, шляхом виявлення та аналізу підозрілих входів у хмару та проведення аналізу *IP*-адрес та їх репутації.

Шифрування даних. Гомоморфне шифрування – це процес, за допомогою якого алгоритми машинного навчання виконують обчислення наявних зашифрованих даних без необхідності їх дешифрування. Додатковою перевагою цього процесу є те, що отримані результати також містяться в зашифрованому тексті, але при розшифровці показують ті самі результати, які вони мали б, якби операцію виконували над розшифрованими даними.

Ухилення від хакерських атак. Використовуючи такі методології, як аналіз поведінки та розпізнавання шаблонів, машинне навчання може допомогти запобігти порушенням даних заздалегідь - перехід від скремблювання до відшкодування збитків після порушення. Це допомагає організаціям бути на крок попереду хакерів, щоб заздалегідь компенсувати потенційні атаки та посилити захист.

Сприяння безпеці кінцевих точок. Машинне навчання може бути використано для підготовки налаштувань безпеки кінцевих точок для виявлення аномалій та шкідливих дій на основі набутих знань. Оскільки машинне навчання процвітає на великих наборах данних, безпеку кінцевих точок можна постійно посилювати проти нових загроз на основі минулих даних та сховищ.

1.2.1. Огляд існуючих програмних рішень

Cybereason. Це платформа для аналізу кібербезпеки, яка забезпечує моніторинг загроз та аналіз. Це надає компаніям та організаціям більшої видимості в середовищі безпеки, а також здатності випереджати загрози. Технологія полювання, що працює на основі штучного інтелекту *Cybereason,* визначає, чи зазнає атаки організація чи ні. Полювання на загрози, як правило, вимагає значних ресурсів, але *Cybereason* автоматизує роботу, завдяки чому команди безпеки будь-якого розміру та рівня кваліфікації можуть отримати користь.

Versive. Рішення допомагає компаніям та організаціям визначати найважливіші загрози, допомагаючи командам економити час, який інакше може бути витрачений на розслідування сповіщень, які не потребують негайної уваги. *Versive Security Engine* (*VSE*) використовує штучний інтелект для відокремлення критичних ризиків від рутинної мережевої діяльності, виявляючи ланцюжки дій, що призводять до атак, та допомагаючи командам безпеки перевершити ці атаки.

Crowdstrike. *Crowdstrike* надає надає власне програмне забезпечення для захисту кінцевих точок. Платформа компанії *Falcon* пропонує запобігання, видимість усіх кінцевих точок та активне полювання на загрози для клієнтів у таких галузях, як фінанси, охорона здоров'я та роздрібна торгівля. Працюючи за рамками простого виявлення, платформа *Falcon* автоматично досліджує загрози, проводить аналіз та вилучає.

PerimeterX пропонує рішення для виявлення ботів на основі машинного навчання для електронної комерції, торгівлі, медіа та корпоративних клієнтів. Платформа виявлення аналізує дані датчиків за допомогою машинного навчання та аналізу поведінки, щоб створити оцінку ризику для кожного користувача. Продукт має комплексний набір інтеграції, що забезпечує просте розгортання для різноманітних інфраструктур.

Shape Security пропонує програмне забезпечення для боротьби з імітаційними атаками, такими як підроблені акаунти, заповнення облікових даних та шахрайство з кредитними додатками для підприємств у роздрібній

торгівлі, фінансах, уряді, техніці та подорожах. Моделі машинного навчання *Shape* отримують доступ до даних, що нагадують зловмисників, це дозволяє системі дізнатися, як виглядає людська діяльність проти шахрайства. Рішення компанії, *Enterprise Defense* та *Blackfish*, використовують програмний продукт для виявлення відмінностей між реальними та штучними користувачами, а потім блокують, переспрямовують або позначають шахрайське джерело.

Deep Instinct. Платформа запобігання загрозам *Deep Instinct* використовує глибоке навчання для запобігання як файловим, так і файловим кібератакам. Захист компанії можна застосувати до будь-якої платформи (мобільні пристрої, сервери, кінцеві точки тощо) та запобігає атакам на основі сценаріїв, включаючи *JavaScript* та *HTML*.

Lastline Defender. Протокол мережевої безпеки *Lastline Defender* дає уявлення про загрози в корпоративній хмарі або гібридному середовищі. Компанія використовує свою глобальну розвідувальну мережу загроз для сканування метаданих та інформації про дорожній рух. Рішення використовує штучний інтелект для виявлення аномалій та попередження адміністраторів про потенційну шкідливу діяльність. Інструмент може миттєво визначити незвичний трафік в мережі: включаючи видобуток біткойнів, віддалене виконання файлів, для забезпечення безпеки в компанії.

1.3. Огляд архітектур нейромереж для перевірки безпеки інформаційних систем

Підходи до глибокого навчання можна класифікувати на дві моделі, а саме глибокі дискримінаційні моделі та генеративні (неконтрольовані) моделі. Моделі глибокої дискримінації включають три підходи, а саме рекурентні нейронні мережі, глибокі нейронні мережі, згорткові нейронні мережі. Генеративні (неконтрольовані) моделі включають чотири підходи: глибокі автокодери, обмежена машина Больцмана, глибокі машини Больцмана та мережі глибоких переконань. Залежно від використання, ці методи можна класифікувати на три категорії наступним чином: глибокі підходи для контрольованого навчання,

глибокі підходи для неконтрольованого або генеративного навчання та гібридні глибинні підходи.

1.3.1. *Deep neural networks*

Deep neural networks (*DNN*). Глибока нейронна мережа – це багатошарові персептрони (*MLP*) з кількістю шарів, що перевищують три. *MLP* - це клас штучної нейронної мережі з прямим зв'язком, що визначається *n* рівнями, що складають її та змінюють один одного, як представлено в нейронній мережі, таким чином зробивши розпізнавання простим у реалізації (рис. 1.1).

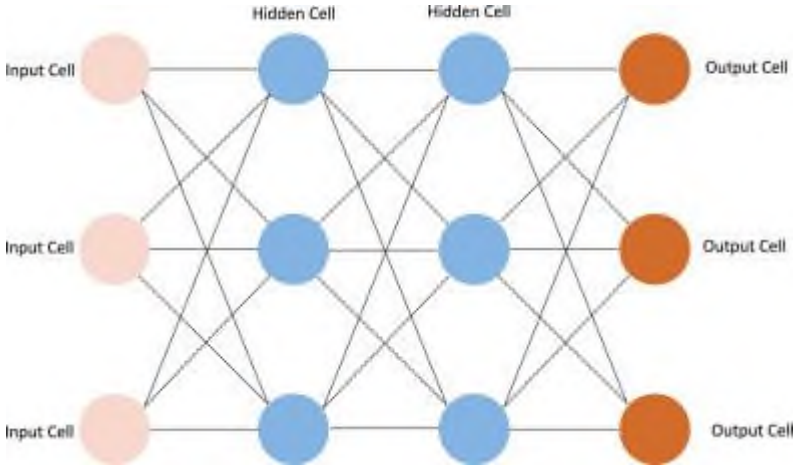

Рис. 1.1. Глибока нейронна мережа (*DNN*)

Вхідні дані передаються нейронам на першому шарі, далі нейронам на наступному шарі і так до фінального виходу. Вихід може являти собою прогноз, на кшталт «Так» або «Ні», поданий за допомогою ймовірностей. На кожному шарі може бути один або безліч нейронів, кожен з яких обчислює невелику функцію, функцію активації. Ця функція імітує передачу сигналу наступним, пов'язаних з попередніми, нейронам. Якщо результат вхідних нейронів перевищує поріг, вихідне значення ігнорується і передається далі. Зв'язок між двома нейронами сусідніх шарів має вагу. Вага визначає вплив вхідних даних на вихід для наступного нейрона і наступний фінальний вихід. Початкові ваги нейромережі випадкові, проте в процесі навчання моделі вони постійно оновлюються і навчаються передбачати вірне вихідне значення. У процесі аналізу нейромережі можна виявити кілька логічних структурних елементів (нейрон, шар, вага, вхід, вихід, функція активації і нарешті механізм навчання,

або оптимізатор), які допомагають їй поступово замінювати ваги (спочатку з випадковими значеннями) на більш підходящі для точного прогнозу виходу.

Для більш ясного розуміння розглянемо, як людський мозок вчиться розрізняти людей. Природа процесу навчання людського мозку досить очевидна. Замість того щоб для впізнавання людей вивчати структуру особи, ми вивчаємо відхилення від типового особи, тобто те, наскільки сильно відрізняються очі конкретної людини від типових очей. Далі ця інформація перетворюється в електричний сигнал певної сили. Подібним же чином вивчаються відхилення всіх інших частин обличчя від типових. Всі ці відхилення в результаті збираються в нові ознаки і дають вихідне значення. Все описане відбувається за частки секунди, і ми просто не встигаємо зрозуміти, що сталося в нашій підсвідомості.

Як показано вище, нейромережа намагається імітувати той же процес, використовуючи математичний підхід. Вхідні дані приймаються нейронами першого шару, і в кожному нейроні обчислюється функція активації. На основі простого правила нейрон передає вихідне значення наступного нейрона, подібно до того, як людський мозок вивчає відхилення. Чим більше вихід нейрона, тим більше значення має відповідна вхідна ознака. На наступному шарі ці ознаки об'єднуються в нові, які мають поки незрозумілу для форму, але система навчається їм інтуїтивно. Повторений безліч разів цей процес призводить до формування складної мережі зі зв'язками.

Коли структура нейромереж зрозуміла, розберемося, як відбувається навчання. З вхідних даних, які ми надаємо мережі, на виході виходить передбачення, яке може бути вірним або невірним. Залежно від виходу ми можемо вимагати від мережі більш точних прогнозів, і система буде навчатися, змінюючи значення ваг для нейронних зв'язків. Щоб правильно дати мережі зворотний зв'язок і визначити наступний крок для внесення змін, ми використовуємо математичний алгоритм. Повторення процесу крок за кроком кілька разів з наростаючим обсягом даних дозволяє нейромережі оновлювати ваги відповідним чином і створює систему, в якій мережа може робити прогноз на основі створених нею через ваги і зв'язки правил. Назва «глибокі нейронні

мережі» пішла від використання безлічі прихованих шарів, які і роблять нейронну мережу «глибокою», здатною навчатися більш складним паттернам.

1.3.2. *Recurrent neural networks*

Recurrent neural networks (*RNN*) – це тип штучної нейронної мережі, яка використовує послідовні дані або дані часових рядів (рис. 1.2). Ці алгоритми глибокого навчання зазвичай використовуються для порядкових або часових проблем, таких як переклад мови, обробка природної мови, розпізнавання мови та субтитри до зображень; вони включені в такі популярні програми, як *Siri*, голосовий пошук та *Google Translate*. Як і прямі та згорткові нейронні мережі (*CNN*), періодичні нейронні мережі використовують навчальні дані для навчання. Вони відрізняються своєю «пам'яттю», оскільки беруть інформацію з попередніх входів, щоб впливати на поточний вхід і вихід. У той час як традиційні глибинні нейронні мережі припускають, що входи та виходи не залежать один від одного, вихід повторюваних нейронних мереж залежить від попередніх елементів у послідовності. Хоча майбутні події також могли б допомогти у визначенні результату даної послідовності, однонаправлені рекурентні нейронні мережі не можуть враховувати ці події у своїх прогнозах.

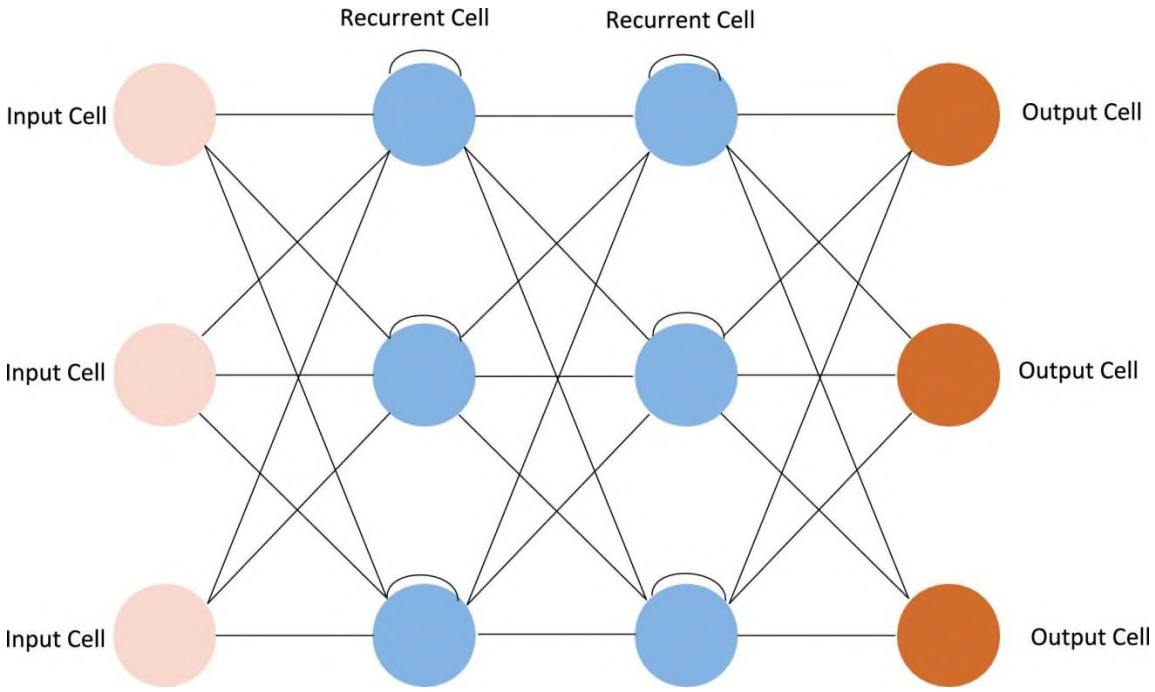

Рис. 1.2. Recurrent neural networks (RNN)

Іншою відмінною характеристикою періодичних мереж є те, що вони діляться параметрами на кожному рівні мережі. Завдяки цьому процесу *RNN*, як правило, стикаються з двома проблемами, відомими як вибухові градієнти та зникаючі градієнти. Ці проблеми визначаються розміром градієнта, який є нахилом функції втрат уздовж кривої помилки. Коли градієнт занадто малий, він продовжує зменшуватися, оновлюючи вагові параметри, доки вони не стануть незначними близькими до нуля. Коли це відбувається, алгоритм більше не навчається. Вибухові градієнти виникають, коли градієнт занадто великий, створюючи нестійку модель. У цьому випадку ваги моделі зростуть занадто швидко, і вони врешті-решт будуть представлені як *NaN*. Одним із вирішень цих питань є зменшення кількості прихованих шарів у нейронній мережі, усуваючи деякі складності в моделі *RNN*.

1.3.3. *Convolutional neural networks*

Convolutional neural networks (*CNN*) – це нейронна мережа, яка має один або кілька згорткових шарів і використовується головним чином для обробки зображень, класифікації, сегментації (рис. 1.3).

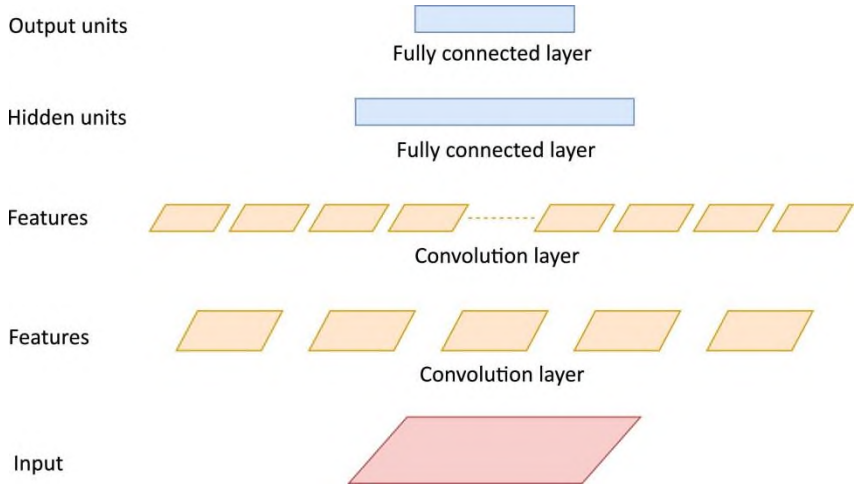

Рис. 1.3. Convolutional neural networks (CNN)

Найпоширенішим використанням *CNN* є класифікація зображень, наприклад, ідентифікація супутникових зображень, що містять дороги, або класифікація рукописних букв і цифр. Є й інші загальнодоступні завдання, такі як сегментація зображень та обробка сигналів, для яких *CNN* добре підходить.

1.3.4. *Generative Adversarial Network*

Generative Adversarial Network (GAN) – алгоритм машинного навчання без учителя, побудований на комбінації з двох нейронних мереж, одна з яких (мережа *G*) генерує зразки, а інша (мережа *D*) намагається відрізнити правильні («справжні») зразки від неправильних (рис. 1.4). Так як мережі *G* і *D* мають протилежні цілі – створити зразки і відбракувати зразки – між ними виникає антагоністична гра.

Принцип змагань в мережі *GAN* нерідко описується через метафори. Наприклад, генеративна мережа уподібнюється фальшивомонетнику або підроблювач картин, а дискримінатор експерту який прагне розпізнати підробку.

GAN складається з двох частин. Генератор вчиться генерувати правдоподібні дані. Дискримінатор вчиться розрізняти дані підроблені генератором від реальних даних. Дискримінатор карає генератор за отримання неправдоподібних результатів.

Коли починається навчання, генератор видає фальшиві дані, і дискримінатор швидко вчиться відрізняти фальшиву інформацію.

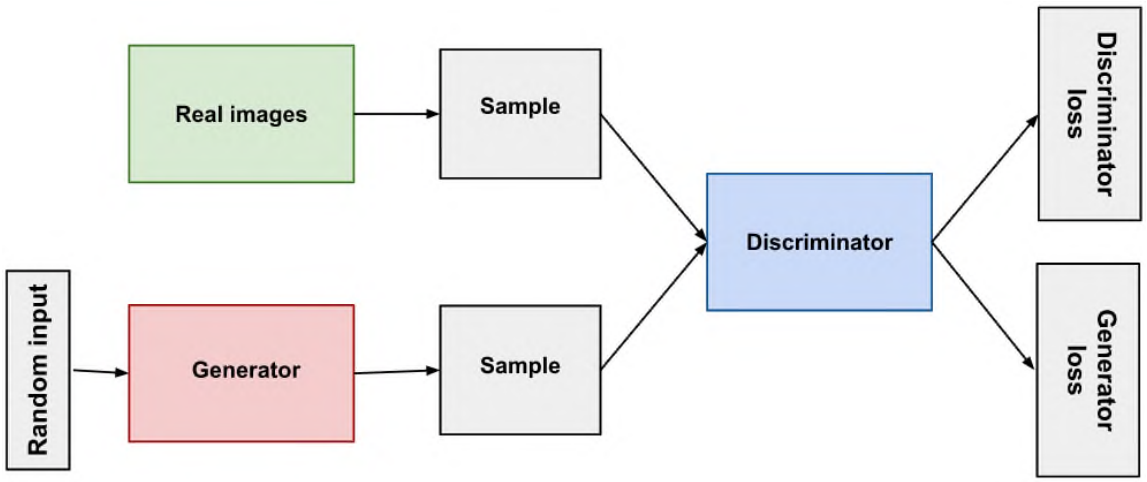

Рис. 1.4. Generative Adversarial Network (GAN)

Генератор та дискримінатор є нейронними мережами. Вихід генератора підключений безпосередньо до входу дискримінатора. Мета генератора – брехати, не бути спійманим. Мета дискримінатора – викрити генератор у брехні.

Дискримінатор в *GAN* – це класифікатор. Дискримінатор намагається відрізнити реальні дані від даних, створених генератором. Класифікатор може

використовувати будь-яку мережеву архітектуру, що відповідає типу даних, які класифікує (рис. 1.5).

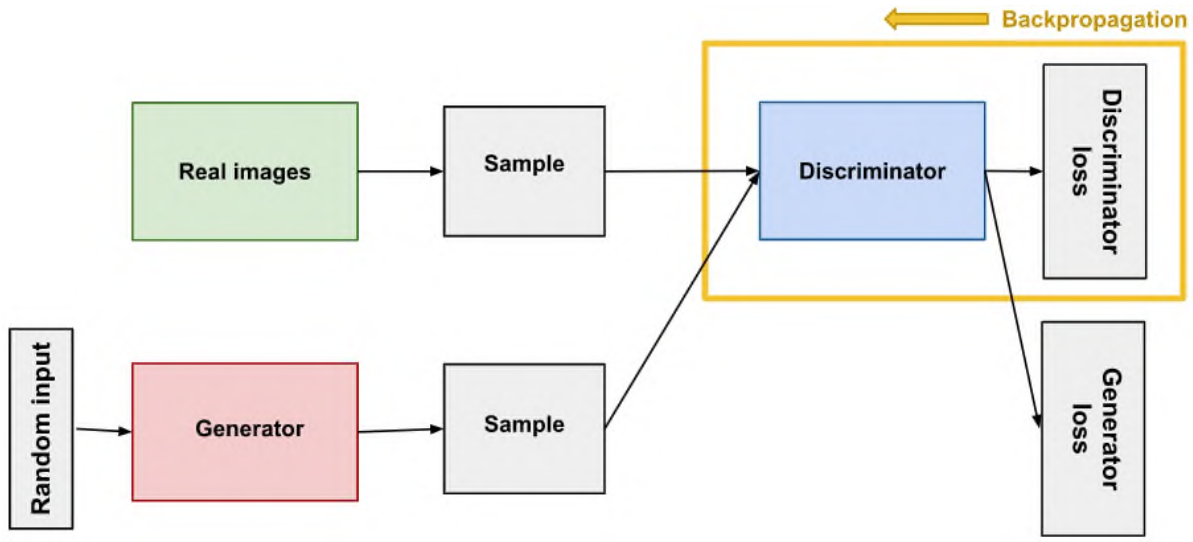

Рис. 1.5. Дискримінатор *GAN*

Дані навчання дискримінатора отримані з двох джерел:

– реальні екземпляри даних, такі як реальні паролі користувачів. (дискримінатор використовує ці приклади в якості позитивних прикладів під час навчання);

– помилкові екземпляри даних, створені генератором. (дискримінатор використовує ці приклади в якості негативних прикладів під час навчання).

На рисунку два поля «*Sample*» представляють ці два джерела даних (*real, fake*), що надходять в дискримінатор. Під час навчання дискримінатора генератор не тренується. Його ваги залишаються постійними, в той час як він дає приклади для навчання дискримінатора.

Навчання дискримінатора. Дискримінатор підключається до двох функцій втрат. Під час навчання дискримінатор ігнорує втрату генератора і використовує втрату дискримінатора. Під час навчання дискримінатор класифікує як реальні дані, так і помилкові дані з генератора. Втрата дискримінатора (*discriminator loss*) карає дискримінатор (*discriminator*) за неправильну класифікацію реального примірника як підробленого або підробленого примірника як реального. Дискримінатор оновлює свої ваги за допомогою зворотного поширення від втрати дискримінатора через мережу дискримінатора. Функція зворотного

поширення коригує в правильному напрямку, розраховуючи вплив ваги на результат – як зміниться результат при зміні ваги.

Навчання генератора вимагає більш тісної інтеграції між генератором і дискримінатором, ніж вимагає навчання дискримінатора (рис. 1.6). Частина *GAN*, яка навчає генератор, включає в себе:

– випадкове введення;

– мережу генератора, яка перетворює випадковий вхід в екземпляр даних;

– мережу, яка класифікує згенеровані дані;

– вихід дискримінатора;

– втрата генератора, яка карає генератор за нездатність обдурити дискримінатор;

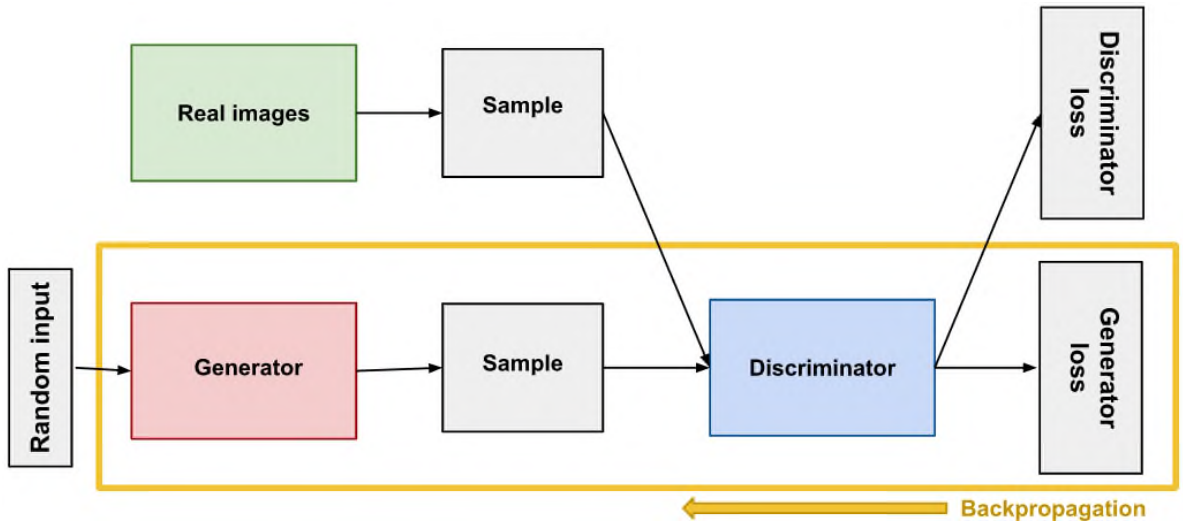

Рис. 1.6. Генератор *GAN*

Нейронні мережі потребують певної форми введення. У своїй основній формі *GAN* приймає випадковий шум в якості входу. Потім генератор перетворює цей шум в значимий вивід. Вводячи шум, ми можемо змусити *GAN* виробляти широкий спектр даних, вибираючи їх з різних місць в цільовому розподілі.

Експерименти показують, що розподіл шуму не має великого значення, тому ми можемо вибрати щось, з чого легко вибрати, наприклад, рівномірний розподіл. Для зручності простір, з якого відбирається шум, зазвичай має меншу розмірність, ніж розмірність вихідного простору.

1.4. Висновки до розділу

В розділі описано актуальність і доцільність технології перевірки безпеки інформаційних систем. Описано області застосування технології. Наведено приклади програмних продуктів, їх характеристики. Описано шляхи використання машинного навчання для інформаційної безпеки. Наведено архітектури нейромереж які використовуються в області інформаційної безпеки.

Абсолютна точність для тренування моделі не може бути досягнута з наявними програмними та апартними засобами. До завдань відноситься програмно реалізувати модель на основі нейромережі *GAN* з максимально можливою вірогідністю вихідних даних.

РОЗДІЛ 2
ВИКОРИСТАНІ ТЕХНОЛОГІЇ ПРИ РОЗРОБЦІ МОДУЛЯ

2.1. Дистрибутив *Anaconda*

Anaconda – це вільно та відкрито розповсюджуваний дистрибутив різних програмних продуктів, зокрема, мов програмування *Python* та *R*. Платформа спеціалізується на "наукових обчисленнях" (*scientific computing*): наука про дані, застосуванні методів машинного навчання, широкомасштабна обробка даних, передбачувальна аналітика тощо. Використання платформи має на меті спрощення управління пакетами та їх розгортання. Версіями пакунків керує система управління пакетами *Conda*.

Дистрибутив Anaconda використовується понад 15 мільйонами користувачів і містить більше 1500 популярних пакетів наукових даних, придатних для *Windows*, *Linux* та *MacOS*, наприклад, *NumPy*, *SciPy* та *Ggplot2*.

Дистрибутив Анаконда має широкі можливості під'єднання модулів (більше 1500), зокрема, власним *Conda* та менеджером віртуального середовища. Графічний інтерфейс, *Anaconda Navigator* слугує графічною альтернативою інтерфейсові командного рядка (*CLI*) (рис. 2.1).

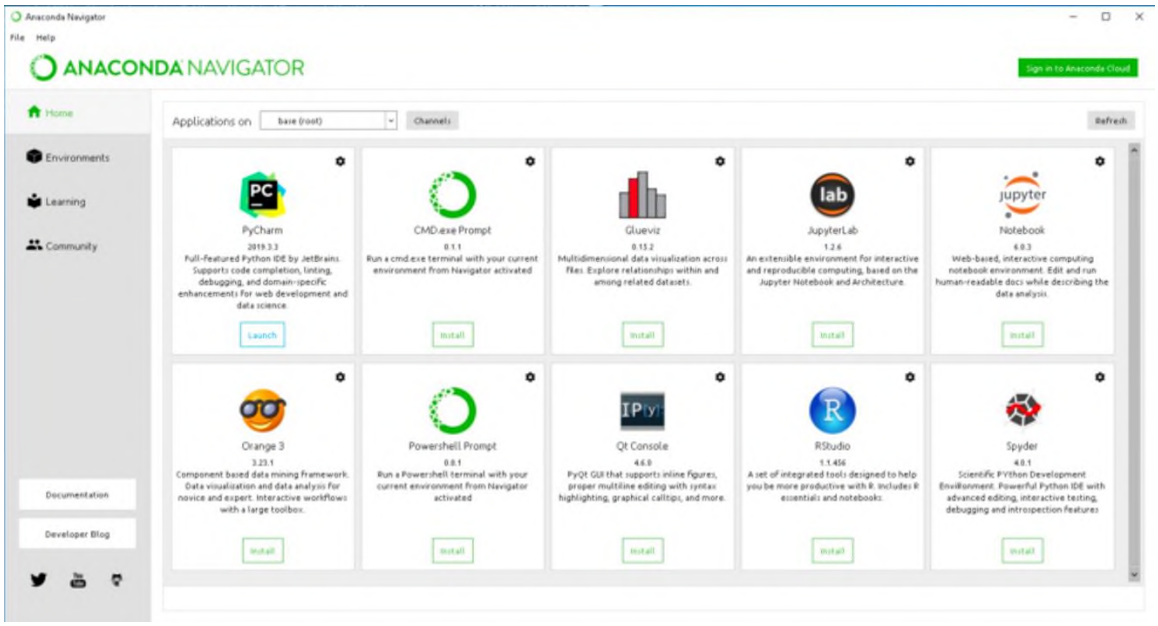

Рис. 2.1. Anaconda Navigator

Велика різниця між *Conda* та менеджером пакетів *Pip* полягає в тому, як управляти залежностями під'єднуваних пакунків, що є суттєвим викликом при роботі з *Data Science* у *Python* та головною причиною появи *Conda*.

Коли *Pip* встановлює деякий потрібний клієнтові пакет, то він автоматично встановлює весь перелік залежних від нього пакетів *Python*, не перевіряючи при цьому, чи вони будуть конфліктувати з раніше встановленими пакунками. Через це користувач із коректно встановленим, наприклад, *Google Tensorflow*, може виявити, що той різко перестає працювати: *Pip* при інсталяції нового пакета встановить інакшу версію *NumPy* (якого потребують одночасно і встановлюваний пакунок і вже наявний *Tensorflow*), наприклад 3.6, тоді як *Tensorflow* коректно працюватиме лише з 3.5. У деяких випадках може спочатку здаватися, що новий пакет працює як слід, але насправді він даватиме відмінні від правильних результати, що буде видно лише у деяких подробицях.

На відміну від цього, *Conda* структурно аналізує все поточне програмне середовище і вже після цього встановлює новий пакет, враховуючи всі обмеження сумісностей, версій різних пакунків, вірніше пропонує поєднуваний набір пакетів. У деяких випадках *Conda* попередить користувача про неможливість одночасного використання деяких пакетів. Як наслідок, тепер користувач може мати, наприклад, *Tensorflow* саме версії 2.0 або новішої, обираючи зручний для себе варіант враховуючи розуміння особливостей версій кожного з пакетів.

Пакети з відкритим кодом можуть бути встановлені індивідуально як зі сховища *Anaconda*, так і з *Anaconda Cloud* чи з вашого власного сховища чи дзеркала, використовуючи команду *conda install*. *Anaconda Inc* компілює та створює всі пакунки у самому сховищі *Anaconda* та надає бінарні файли для *Windows*, *Linux* та *MacOS*. Все, що доступне на *PyPI*, може бути встановлено в середовищі *Conda* за допомогою *Pip*, і *Conda* буде відслідковувати, що саме було встановлено самим пакунком і що саме встановлено за допомогою *Pip*.

Окремі, власні пакети можна створити за допомогою *conda build*, ними можна поділитись з іншими, завантаживши їх у будь-яке сховище, як-от: *Anaconda Cloud* або *PyPI*.

Установка *Anaconda*2 за замовчуванням тягне за собою *Python* 2.7, а *Anaconda*3 включає *Python* 3.7. Однак за допомогою *Conda* завжди можна створити середовища, задані по-новому, щоб містити будь-яку версію *Python*.

Anaconda Navigator – це графічний інтерфейс користувача настільних ПК (*GUI*), що входить у дистрибутив *Anaconda*, який дозволяє користувачам запускати пов'язані програми та керувати пакетами, середовищами та каналами *Conda* без використання часто менш зручного командного рядка. Навігатор може шукати пакети в *Anaconda Cloud* або в локальному сховищі *Anaconda*, встановлювати їх у середовищі, запускати пакети та оновлювати їх. Працює у *Linux, macOS* та *Windows*.

Conda – вільно та відкрито розповсюджуваний крос-платформний та безвідносний до мови програмування (*language-agnostic*) менеджер пакетів та система управління середовищем *Anaconda* яка встановлює, запускає та оновлює пакети та їх залежності. Від початку створений для програм *Python*, сьогодні може пакувати та розповсюджувати програмне забезпечення для дуже широкого переліку мов, в тому числі для багатомовних проектів. Пакет Conda та менеджер навколишнього середовища включений у всі версії *Anaconda, Miniconda* та *Anaconda Repository*.

Anaconda Cloud – це послуга управління пакетами від *Anaconda*, де ви можете знаходити, отримувати доступ, зберігати та ділитися загальнодоступними та приватними *notebook* (*notebook interface*), середовищами та пакетами *Conda* та *PyPI*. Розміщені у хмарі *notebook* та середовища застосовуються для вирішення широкого спектра задач, при цьому Вам не потрібно мати обліковий запис *Anaconda Cloud*, шукати загальнодоступні пакунки серед платних, постійно завантажувати та встановлювати їх.

2.2. Фреймворк *TensorFlow*

TensorFlow – відкрита програмна бібліотека для машинного навчання цілій низці задач, розроблена компанією *Google* для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для виявлення та

розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди. Наразі застосовують як для досліджень, так і для розробки продуктів *Google*.

Існує три основні конструкції для операцій *TensorFlow*: вектори, масиви (матриці), тензори. Вектор – це математичний об'єкт, який має напрямок і величину. За його допомогою знаходять положення однієї точки в просторі відносно іншої точки. Масив – це розташування або ряд елементів, таких як символи, цифри або вирази. Масиви можуть бути *n*-мірними, тому матриця – це масив із 2 вимірами. Тензор – це об'єкт, що описує лінійний зв'язок між скалярами, векторами та іншими тензорами (рис. 2.2). Іншими словами, це укладання декількох масивів для створення вищих розмірних структур. Практичним прикладом тензора в дії є зображення. Коли обробляється зображення *RGB*, це тривимірний тензор із шарами для кожного кольору в розмірах висоти та ширини.

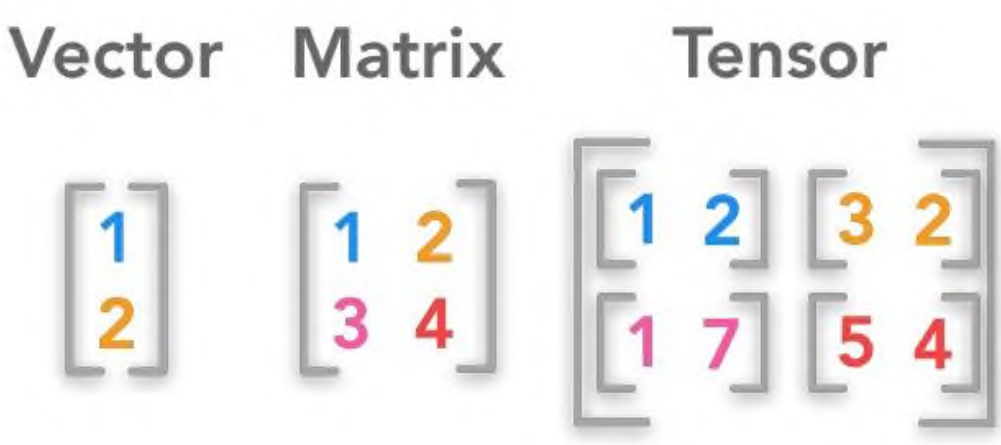

Рис. 2.2. Основні конструкції для операцій *TensorFlow*

TensorFlow є крос-платформним. Він працює майже на всьому: графічні процесори та центральні процесори, мобільні та вбудовані платформи, блоки обробки тензорів, які є спеціалізованим обладнанням для обчислення тензорної математики.

Програми для глибокого навчання складні, оскільки навчальний процес вимагає великих обчислень. Це займає багато часу через великий обсяг даних, і включає кілька ітераційних процесів, математичні обчислення, множення матриць тощо. Якщо виконувати ці дії на звичайному центральному процесорному блоці, зазвичай це займе багато часу.

Блоки графічної обробки (*GPU*) популярні в контексті ігор, де екран і зображення повинні мати високу роздільну здатність. Для цієї мети спочатку були розроблені графічні процесори. Однак вони також використовуються для розробки програм глибокого навчання.

Однією з головних переваг *TensorFlow* є те, що він підтримує графічні процесори, а також центральні процесори. Він також має швидший час компіляції, ніж інші бібліотеки глибокого навчання, такі як *Keras* і *Torch*.

2.3. Алгоритм машинного навчання *GAN*

Для написання дипломної роботи обрано алгоритм машинного навчання GAN (*Generative Adversarial Network*). Це клас алгоритмів штучного інтелекту, який використовується в навчанні без викладача, реалізована система двох штучних нейронних мереж, яка змагається одна з одною. Одна мережа генерує кандидатів (генератор), а інша оцінює їх (дискримінатор). Як правило, генеративна мережа навчається будувати відповідності з латентного простору до певного розподілу даних, тоді як дискримінаційна мережа розрізняє представників справжнього розподілу даних та кандидатів, вироблених генератором. Метою тренувальної мережі є збільшення частоти помилок дискримінаційної мережі (тобто «обдурити» дискримінатор шляхом створення нових синтезованих екземплярів, які повинні походити на представників справжнього розподілу даних).

Заздалегідь відомий набір даних використовують як початкові навчальні данні для дискримінатора. Навчання дискримінатора передбачає забезпечення його зразками з набору даних, доки він не досягне певного рівня точності. Зазвичай генератор на початку отримує випадково відібрані дані із заздалегідь визначеного латентного простору (наприклад, за допомогою багатовимірного нормального розподілу. Після цього зразки, синтезовані генератором, оцінюються дискримінатором. Метод зворотного поширення помилки застосовується в обох мережах, так що генератор створює кращі зображення, тоді як дискримінатор стає більш кваліфікованим при визначенні синтезованих

зображень. Генератор, як правило, є деконволюційною нейронною мережею, а дискримінатор – згортковою нейронною мережею.

Ідея вивести моделі в конкурентному середовищі (модель проти дискримінатора) була запропонована Лі, Гаучі та Гросом в 2013 році. Метод використовується для висновків поведінки. Це називається навчання по Тюрінгу, оскільки цей параметр схожий на тест Тюрінга. Навчання по Тюрінгу є узагальненням генеративної змагальної мережі. Можуть розглядатись і моделі відмінні від нейронних мереж. Крім того, дискримінаторам дозволяється впливати на процеси, з яких отримані набори даних, що робить їх активними учасниками, як у тесті Тюрінга. Ідею змагального навчання можна знайти й у більш ранніх роботах, таких як стаття Шмідхубера 1992 року.

Принцип змагальності в мережі *GAN* нерідко описується через метафори. Наприклад, генеративна мережа уподібнюється фальшивомонетнику або підроблювач картин, а дискримінатор експерту який прагне розпізнати підробку. *GAN* складається з двох частин:

Генератор вчиться генерувати правдоподібні дані.

Дискримінатор вчиться розрізняти дані підроблені генератором від реальних даних. Дискримінатор карає генератор за отримання неправдоподібних результатів.

Коли починається навчання, генератор видає явно фальшиві дані (рис. 2.3), і дискримінатор швидко вчиться говорити, що це фальшива інформація:

Рис. 2.3. Приклад генерація фальшивих даних

В процесі навчання генератор стає ближче до отримання вихідних даних (рис. 2.4), які можуть обдурити дискримінатор:

Рис. 2.4. Приклад генерація фальшивих даних

Якщо тренування генератора проходить добре, дискримінатору стає складно відрізнити реальне від фальшивого (рис. 2.5). Дискримінатор починає класифікувати підроблені дані як реальні, та їх точність зменшується.

Рис. 2.5. Приклад генерації натренованої моделі

GAN це нова архітектура некерованої нейронної мережі, здатна досягти набагато кращої продуктивності порівняно з традиційними мережами. Точніше кажучи, *GAN* – це новий спосіб тренування нейронної мережі.

Картина всієї системи виглядає наступним чином (рис. 2.6):

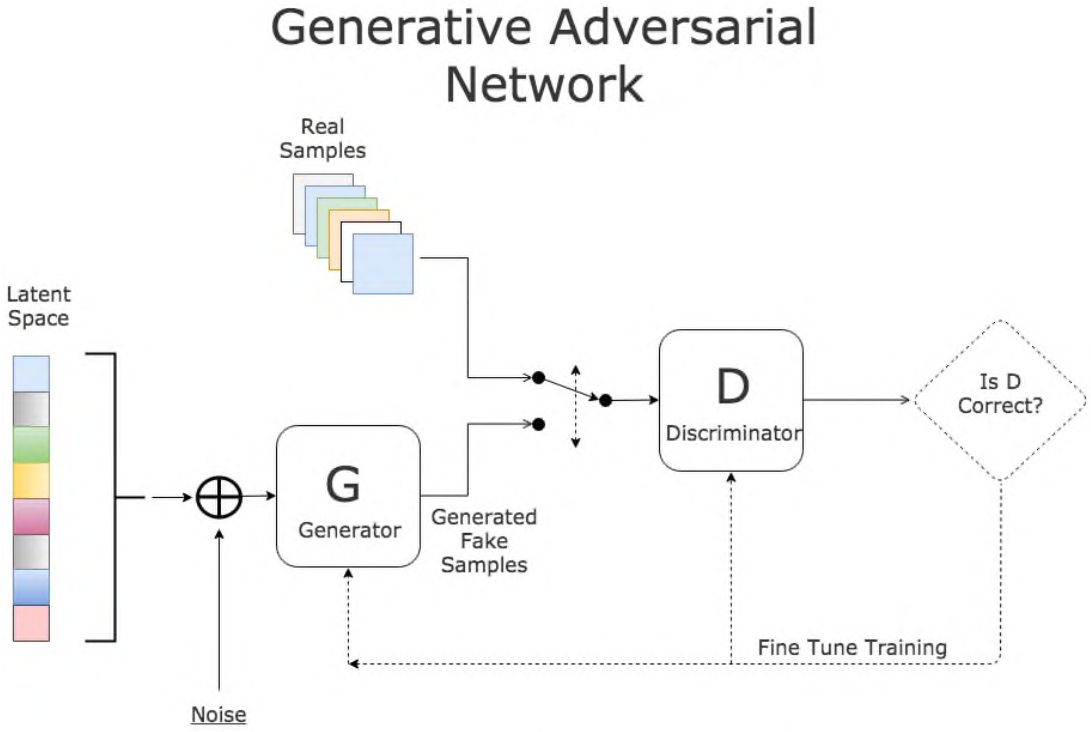

Рис. 2.6. Архітектура нейромережі *GAN*

І генератор, і дискримінатор є нейронними мережами. Вихід генератора підключений безпосередньо до входу дискримінатора.

GAN схожа на протидію фальшивомонетника і поліцейського в грі в кішки-мишки, де фальшивомонетник вчиться передавати фальшиві замітки, а поліцейський вчиться їх виявляти. Обидва є динамічними, поліцейський теж навчається і кожна сторона постійно вивчає інші методи. В дипломній роботі

генератор – генерує фальшиві паролі, а дискримінатор намагається звинуватити генератор у брехні.

Дискримінатор в *GAN* – це класифікатор. Дискримінатор намагається відрізнити реальні дані від даних, створених генератором. Класифікатор може використовувати будь-яку мережеву архітектуру, що відповідає типу даних, які класифікує (рис. 2.7).

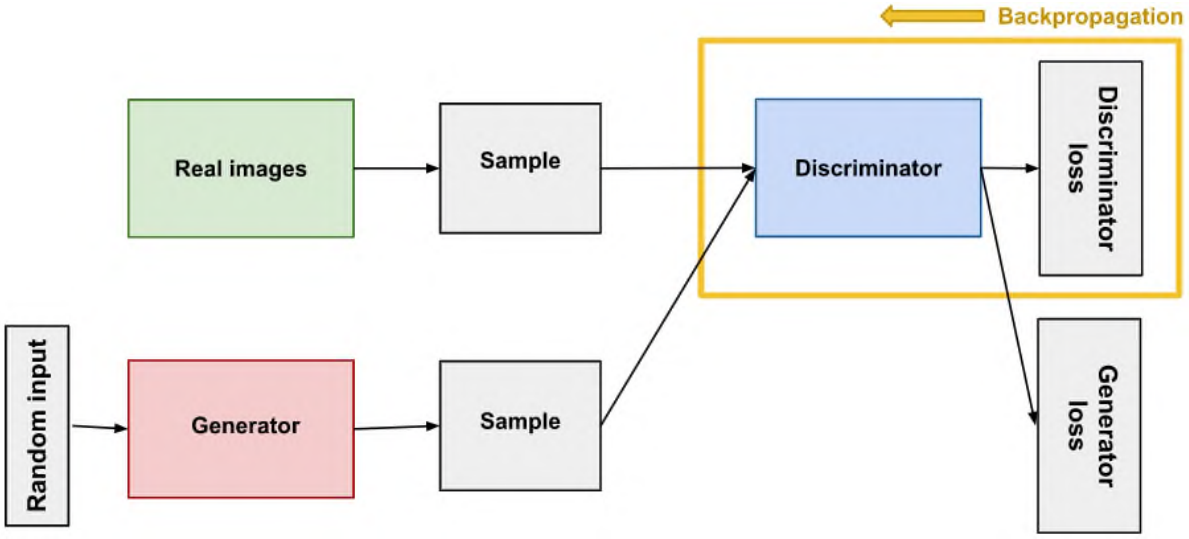

Рис. 2.7. Дискримінатор *GAN*

Дані навчання дискримінатора отримані з двох джерел:

– реальні екземпляри даних, такі як реальні паролі користувачів. (дискримінатор використовує ці приклади в якості позитивних прикладів під час навчання);

– помилкові екземпляри даних, створені генератором. (дискримінатор використовує ці приклади в якості негативних прикладів під час навчання).

На рисунку два поля «*Sample*» представляють ці два джерела даних (*real, fake*), що надходять в дискримінатор. Під час навчання дискримінатора генератор не тренується. Його ваги залишаються постійними, в той час як він дає приклади для навчання дискримінатора.

Навчання дискримінатора. Дискримінатор підключається до двох функцій втрат. Під час навчання дискримінатор ігнорує втрату генератора і використовує втрату дискримінатора. Під час навчання дискримінатор класифікує як реальні дані, так і помилкові дані з генератора. Втрата дискримінатора (*discriminator loss*) карає дискримінатор (*discriminator*) за неправильну класифікацію реального

примірника як підробленого або підробленого примірника як реального. Дискримінатор оновлює свої ваги за допомогою зворотного поширення від втрати дискримінатора через мережу дискримінатора. Функція зворотного поширення коригує в правильному напрямку, розраховуючи вплив ваги на результат – як зміниться результат при зміні ваги. Метод зворотного поширення помилки (*backpropagation*) – метод обчислення градієнта, який використовується при оновленні ваг багатошарового перцептрона. Це ітеративний градієнтний алгоритм, який використовується з метою мінімізації помилки роботи багатошарового перцептрона і отримання бажаного виходу.

Основна ідея цього методу полягає в поширенні сигналів помилки від виходів мережі до її входів, в напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи.

Для можливості застосування методу зворотного поширення помилки передавальна функція нейронів повинна бути диференційована. Метод є модифікацією класичного методу градієнтного спуску.

Частина генератора *GAN* вчиться створювати помилкові дані, використовуючи зворотний зв'язок від дискримінатора. Він вчиться змушувати діскримінатор класифікувати свій висновок як реальний.

Навчання генератора вимагає більш тісної інтеграції між генератором і дискримінатором, ніж вимагає навчання дискримінатора (рис. 2.8). Частина *GAN*, яка навчає генератор, включає в себе:

– випадкове введення;

– мережу генератора, яка перетворює випадковий вхід в екземпляр даних;

– мережу, яка класифікує згенеровані дані;

– вихід дискримінатора;

– втрата генератора, яка карає генератор за нездатність обдурити дискримінатор;

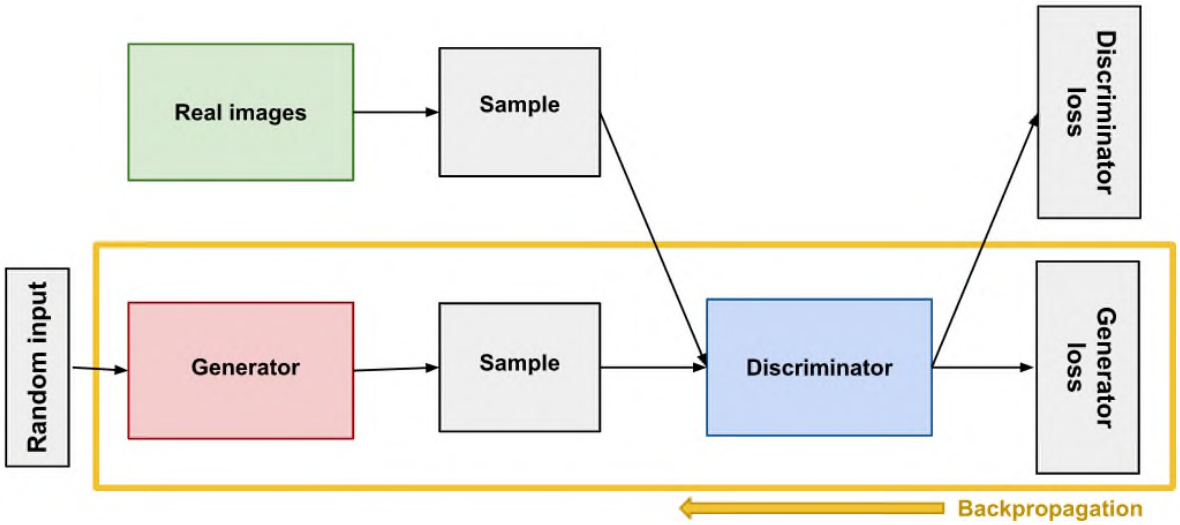

Рис. 2.8. Генератор *GAN*

Нейронні мережі потребують певної форми введення. У своїй основній формі *GAN* приймає випадковий шум в якості входу. Потім генератор перетворює цей шум в значимий вивід. Вводячи шум, ми можемо змусити *GAN* виробляти широкий спектр даних, вибираючи їх з різних місць в цільовому розподілі.

Експерименти показують, що розподіл шуму не має великого значення, тому ми можемо вибрати щось, з чого легко вибрати, наприклад, рівномірний розподіл. Для зручності простір, з якого відбирається шум, зазвичай має меншу розмірність, ніж розмірність вихідного простору.

Використання дискримінатора для навчання генератора. Щоб навчити нейронну мережу, потрібно змінювати вагу мережі, щоб зменшити помилку або її втрату. Однак в *GAN* генератор не пов'язаний безпосередньо з втратою, на яку ми намагаємося вплинути. Генератор надходить в мережу дискримінатора, а дискримінатор видає результат, на який ми намагаємося вплинути. Втрати генератора штрафують генератор за створення вибірки, яку мережу дискримінатора класифікує як неправдиву.

Цей додатковий шматок мережі повинен бути включений в зворотне поширення. Функція зворотного поширення коригує зміни в правильному напрямку, розраховуючи вплив ваги на результат – як зміниться результат при зміні ваги. Але вплив ваги генератора залежить від впливу ваги дискримінатора,

в який він подається. Таким чином, зворотне поширення починається на виході і тече назад через дискримінатор в генератор.

У той же час, не потрібно, щоб дискримінатор змінювався під час навчання генератора.

Отже, тренування генератора відбувається за такою процедурою:

– зразок випадкового шуму;

– створення вихідного сигналу генератора з дискретного випадкового шуму.

– отримати дискримінатором «Реальна» або «Підроблена» класифікацію для виходу генератора;

– розрахувати втрати з класифікації дискримінатора;

– дорога назад через дискримінатор і генератор для отримання градієнтів;

– використовуємо градієнти, щоб змінити тільки вага генератора.

Вище розглянуто одну ітерацію навчання генератора.

GAN Навчання. Оскільки *GAN* містить дві окремо навчені мережі, алгоритм навчання має враховувати дві складності:

– *GAN* повинен маніпулювати двома різними типами навчання (генератор і дискримінатор);

– конвергенцію *GAN* важко визначити.

Мінливе тренування. Навчання *GAN* проходить наступним чином:

– дискримінатор тренується протягом однієї або декількох епох;

– генератор тренується протягом однієї або декількох епох;

– повторення кроків 1 та 2, для продовження навчання мереж генератора і дискримінатора.

Генератор зберігається постійним на етапі навчання дискримінатора. Оскільки тренування дискримінатора намагається з'ясувати, як відрізнити реальні дані від фальшивих, потрібно навчитись розпізнавати недоліки генератора. Це абсолютно інша проблема для ретельно навченого генератора, ніж для непідготовленого генератора, який видає випадковий вихід.

Зберігаємо дискримінатор постійним під час фази навчання генератора. В іншому випадку генератор буде намагатися вразити рухому ціль і може ніколи не зійтися.

Це дозволяє *GAN* вирішувати задачі, які складно вирішити, проблеми генерації. Це складна проблему, почавши з набагато більш простої задачі класифікації.

Конвергенція. Оскільки одночасно навчаються дві мережі, проблема конвергенції *GAN* є однією з найперших і, можливо, однією з найскладніших проблем з моменту її створення.

В більшості випадків важко досягти утопічної ситуації, коли обидві мережі стабілізуються і дають стабільний результат. Одне проблема полягає в тому, що в міру того, як генератор стає кращим з наступними епохами, дискримінатор працює гірше, оскільки дискримінатор не може легко відрізнити реальну від фальшивої.

Якщо генератор постійно досягає успіху, дискримінатор має половину точності, подібну до точності гортання монети. Це створює загрозу для зближення *GAN* в цілому.

На зображенні нижче показано цю проблему (рис. 2.9):

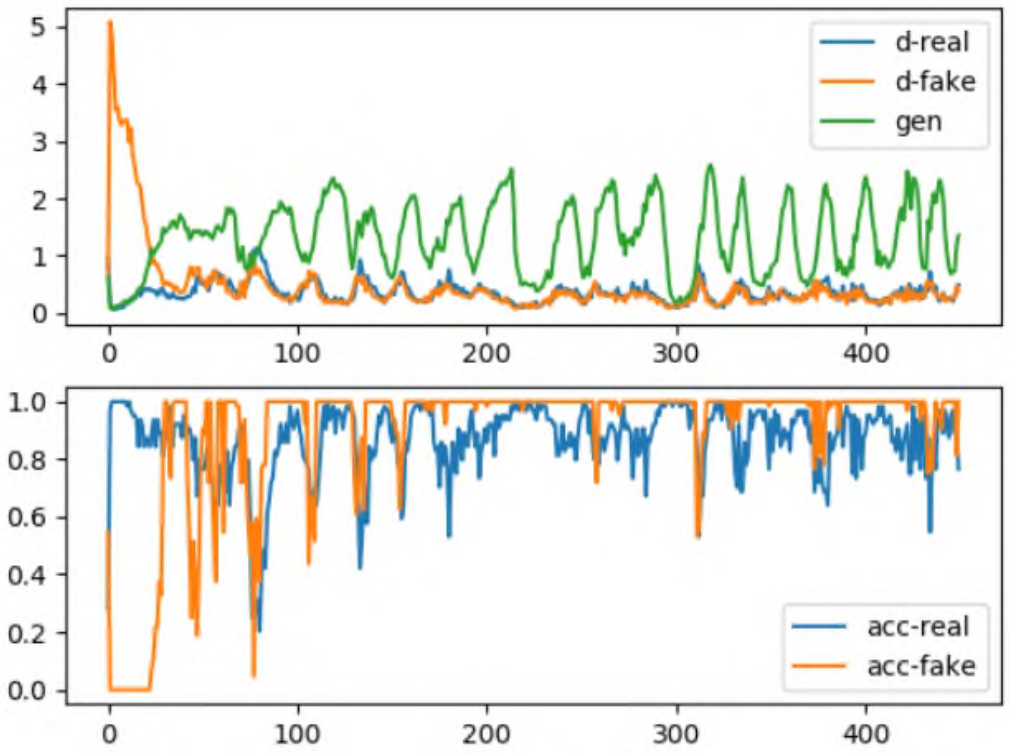

Рис. 2.9. Конвергенція *GAN*

Прогресія створює проблему для зближення *GAN* в цілому: зворотний зв'язок дискримінатора стає менш значущим з плином часу. Якщо *GAN* продовжує навчання після точки, коли дискримінатор дає абсолютно випадковий зворотний зв'язок, тоді генератор починає тренуватися на небажаному зворотному зв'язку, точність та якість може погіршитись (рис. 2.10).

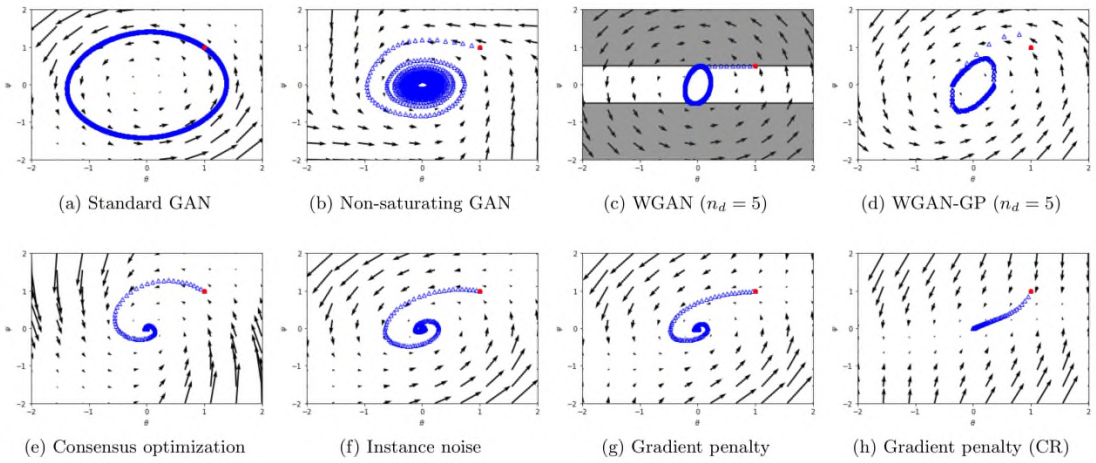

Рис. 2.10. Властивості збіжності різних навчальних алгоритмів *GAN*

Для *GAN* конвергенція часто є скороминущим, а не стабільним станом.

Функція втрати. *GAN* намагається відтворити розподіл ймовірностей. Тому потрібно використовувати функції втрат, які відображають відстань між розподілом даних, що генеруються *GAN*, і розподілом реальних даних.

Це питання є областю активних досліджень, і було запропоновано багато підходів. Приклади функії втрат *minimax loss, Wasserstein loss* та інші.

GAN може мати дві функції втрат: одну для навчання генератора і одну для навчання дискримінатора.

У схемах втрат, втрати генератора і дискримінатора походять з однієї міри відстані між розподілами ймовірностей. Однак в обох цих схемах генератор може впливати тільки на один параметр у вимірі відстані: параметр, який відображає розподіл підроблених даних. Тому під час навчання генератора відкидається інший параметр, який відображає розподіл реальних даних.

В кінцевому підсумку втрати генератора і дискримінатора виглядають по-різному, навіть якщо вони отримані з однієї формули.

Параметри для тренування моделі**.** Наступні гіперпараметри можуть характеризувати модель:

– *batch size* (розмір пакета), представляє дані з навчального набору, які поширюються через *GAN* на кожному кроці оптимізатора;

– *number of iterations* (число ітерацій), вказує, скільки разів *GAN* викликає свій крок вперед і крок зворотного поширення. У кожній ітерації *GAN* виконує одну ітерацію генератора і одну або декілька ітерацій дискримінатора;

– *number of discriminator iterations per generator iteration* (число ітерацій дискримінатора на одну ітерацію генератора), яке вказує, скільки ітерацій дискримінатор виконує в кожній ітерації *GAN*;

– *model dimensionality* (розмірність моделі), яка представляє кількість вимірювань для кожного загорткового шару;

– *gradient penalty coefficient (λ)* (коефіцієнт штрафу градієнта (λ)), який визначає штраф, який застосовується до норми градієнта дискримінатора щодо його вхідних даних. Збільшення цього параметра призводить до більш стабільної тренуванні;

– *output sequence length* (довжина вихідний послідовності), яка вказує максимальну довжину рядків, що генеруються генератором (*G*);

– *size of the input noise vector (seed)* (розмір вектора вхідного шуму (початкового числа)), який визначає, скільки випадкових чисел з нормального розподілу подається в якості вхідних даних в *G* для генерації вибірок;

– *Maximum number of examples* (Максимальна кількість прикладів), яке представляє максимальну кількість навчальних елементів для завантаження.

Adam optimizer's hyper-parameters (гіпер-параметри оптимізатора *Adam*):

– швидкість навчання, як швидко коригуються ваги моделі;

– коефіцієнт *β1*, який вказує швидкість убування змінного середнього градієнта;

– коефіцієнт *β2*, який вказує швидкість убування змінного середнього квадрата градієнта.

При навчанні глибокої нейронної мережі помилка навчання зменшується зі збільшенням кількості шарів. Однак, після досягнення певної кількості шарів, помилка навчання знову починає збільшуватися.

2.4. Алгоритм оптимізації *Adam*

Вибір алгоритму оптимізації для моделі глибокого навчання може означати різницю між хорошими результатами в хвилинах, годинах і днях.

Алгоритм оптимізації *Adam* – це розширення стохастичного градієнтного спуску, яке нещодавно стало широко застосовуватися для програм глибокого навчання в комп'ютерному зорі та обробці природних мов. Це алгоритм оптимізації, який можна використовувати замість класичної стохастичної процедури градієнтного спуску для оновлення вагових коефіцієнтів мережі ітеративно на основі навчальних даних.

Adam був представлений *Diederik Kingma* з *OpenAI* та Jimmy Ba з Університету Торонто у документі *ICLR* 2015 року під назвою «*Adam: A Method for Stochastic Optimization*».

Переваги використання алгоритму *Adam* в неопуклих задачах оптимізації:

– простий в реалізації;

– обчислювальна ефективність;

– невеликі вимоги до пам’яті;

– інваріантний до діагонального масштабування градієнтів;

– добре підходить для великих проблем з точки зору даних або параметрів;

– підходить для нестаціонарних цілей;

– підходить для проблем із дуже шумними або рідкісними градієнтами;

– гіперпараметри мають інтуїтивну інтерпретацію і, як правило, вимагають незначне налаштування.

Адам відрізняється від класичного стохастичного градієнтного спуску.

Стохастичний градієнтний спуск підтримує єдину швидкість навчання для всіх оновлень ваги, і швидкість навчання не змінюється під час тренування.

Швидкість навчання підтримується для кожної ваги мережі (параметра) і окремо адаптується в міру розгортання навчання.

Метод обчислює індивідуальні показники адаптивного навчання для різних параметрів на основі оцінок першого та другого моментів градієнтів.

Автори описують *Adam* як поєднання переваг двох інших розширень стохастичного градієнтного спуску. Такі як *AdaGrad* та *RMSProp*.

Адаптивний алгоритм градієнта (*AdaGrad*), який підтримує швидкість навчання за параметром, що покращує ефективність роботи з проблемами рідкісних градієнтів (наприклад, проблеми з природною мовою та комп'ютерним зором).

Середньоквадратичне розмноження (*RMSProp*), також підтримує коефіцієнти навчання за параметрами, які адаптуються на основі середнього значення останніх величин градієнтів для ваги (наприклад, наскільки швидко вона змінюється). Це означає, що алгоритм добре справляється з інтерактивними та нестаціонарними проблемами (наприклад з шумом).

Adam втілив в собі переваги як *AdaGrad*, так і *RMSProp* (рис. 2.11).

Замість того, щоб адаптувати швидкість навчання параметрів на основі середнього першого моменту (середнього значення), як у *RMSProp*, *Adam* також використовує середнє значення інших моментів градієнтів (нецентрова дисперсія).

Зокрема, алгоритм обчислює експоненціальний середній градієнт та квадратичний градієнт, а параметри *beta1* та *beta2* контролюють швидкість спаду середніх величин.

Початкове значення середніх величин та значень *beta1* та *beta2*, близьких до 1,0 (рекомендовано), призводять до зміщення оцінок моменту до нуля. Це упередження долається спочатку обчисленням упереджених оцінок, а потім обчисленням, скоригованим із зміщенням.

Adam є популярним алгоритмом у галузі глибокого навчання, оскільки він швидко досягає хороших результатів.

Емпіричні результати демонструють, що *Adam* добре працює на практиці і вигідно порівнює з іншими методами стохастичної оптимізації. *Adam* продемонстрував, що конвергенція відповідає очікуванням теоретичного аналізу. Алгоритм *Adam* застосовано до алгоритму логістичної регресії на наборах даних розпізнавання цифр *MNIST* та аналізу настроїв *IMDB*, багатошарового алгоритму перцептрона на наборі даних *MNIST* та згорткових нейронних мереж на наборі

даних розпізнавання зображень *CIFAR*-10. Використовуючи великі моделі та масиви даних *Adam* може ефективно вирішувати практичні проблеми глибокого навчання.

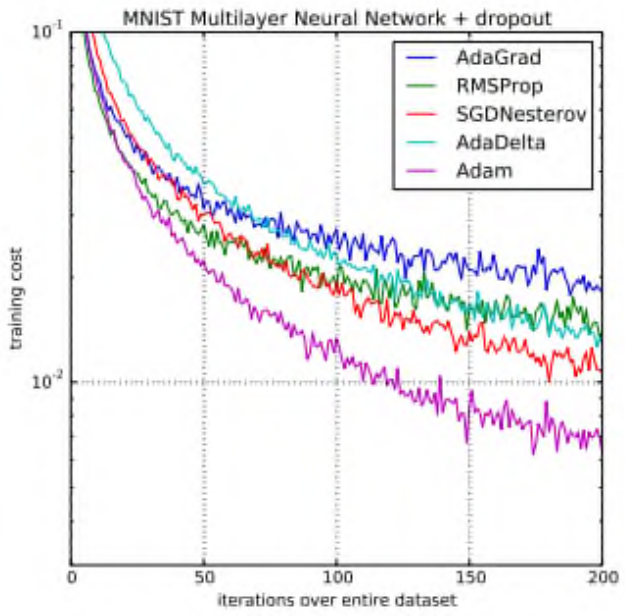

Рис. 2.11. Порівняння з іншими алгоритмами оптимізації

Параметри *Adam*:

– *alpha*. Швидкість навчання або розмір кроку. Відсоток оновлення ваг (наприклад, 0.001). Більші значення (наприклад, 0.3) призводять до швидшого початкового навчання перед оновленням. Менші значення (наприклад, 0.00001) сповільнюють навчання прямо під час тренування.

– *beta1*. Експоненціальна швидкість занепаду для першого моменту;

– *beta2*. Експоненціальна швидкість занепаду для оцінок другого моменту. Це значення слід встановити близько 1 для проблем з розрідженим градієнтом (наприклад, проблеми комп'ютерного зору);

– *epsilon*. Це маленьке число для запобігання діленню на нуль у реалізації.

Крім того, зниження швидкості навчання також можна використовувати з *Adam*. Хорошими налаштуваннями за замовчуванням для перевірених проблем машинного навчання є *alpha* = 0.001, *beta1*= 0.9, *beta2* = 0.999 та e *epsilon* = 10^{-8}

Документація *TensorFlow* пропонує налаштування *epsilon*. Значення за замовчуванням 10^{-8} для *epsilon* може бути невдалим за замовчуванням. Наприклад, при навчанні початкової мережі на *ImageNet* поточним хорошим

вибором є 1.0 або 0.1.

2.5. Висновки до розділу

В даному розділі описано технології, які використані при написання дипломної роботи. Оглянуто дистрибутив *Anaconda*, за допомогою якого можливо створити зручне віртуальне середовище для розробки проекту. Наведено опис фреймворку для вирішення задач з машинного навчання, а саме *TensorFlow*. Для вирішення поставлених задач обрано алгоритм машинного навчання *GAN*, описано роботу даного алгоритму. Наведено опис алгоритму оптимізації *Adam*, описано параметри.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ПЕРЕВІРКИ БЕЗПЕКИ ІНФОРМАЦІЙНИХ СИСТЕМ ЗАСОБАМИ НЕЙРОМЕРЕЖІ *GAN*

3.1. Постановка задачі

Завдання дипломної роботи полягає у розробці програмного засобу перевірки безпеки інформаційних систем засобами нейромережі *GAN*. Для досягнення поставленого завдання необхідно виконати наступні задачі:

– проаналізувати сучасні методи перевірки інформаційної безпеки;

– проаналізувати архітектуру нейромереж для перевірки безпеки інформаційних систем;

– розробити модуль тренування моделі;

– розробити модуль генерації вихідних данних;

– розробити модуль підготовки вхідних даних;

– детально оглянути роботу архітектури *GAN*;

– програмно реалізувати модель на основі нейромережі *GAN*;

– розробити архітектуру для моделі;

– розробити інтерфейс користувача;

– обрати оптимальні параметри для тренування моделі;

– проаналізувати вхідні та вихідні дані, перевірити на тестовому наборі хешів.

– виконати перевірку вихідних даних з правилами. Порівняти з існуючими програмними засобами;

– обрати середовище розробки, мови програмування, програмні інструменти та інші засоби для розробки програмного продукту.

Паролі є найбільш популярним методом аутентифікації, в основному тому, що вони прості в реалізації, не вимагають спеціального обладнання або програмного забезпечення і знайомі користувачам і розробникам. На жаль,

численні витоки з бази даних паролів показали, що користувачі, як правило, вибирають паролі які легко вгадати, в основному складаються з загальних рядків (наприклад, *password,* 123456, *iloveyou*) та їх варіантів.

Інструменти вгадування паролів надають цінний інструмент для виявлення слабких паролів, особливо коли вони зберігаються у вигляді хешу. Ефективність програмного забезпечення для підбору паролів залежить від можливості швидкого тестування великої кількості найбільш ймовірних паролів для кожного хеша пароля. Замість вичерпної перевірки всіх можливих комбінацій символів, інструменти вгадування паролів використовують слова зі словників і попередніх витоків паролів в якості можливих паролів. Сучасні інструменти для підбору пароля, такі як *HashCat*, *John the Ripper* та інші.

Можна зробити підхід ще на крок вперед, визначивши евристику для перетворення пароля, яка включає в себе комбінації декількох слів (наприклад, *iloveyou*123456), регістр змішаних букв (наприклад, *iLoVeyOu*) та дозволяє створювати (наприклад, il0v3you). Ця евристика в поєднанні з марковскими моделями дозволяє *HashCat, John the Ripper* генерувати велику кількість нових ймовірних паролів.

Хоча на практиці ці евристики досить успішні, вони носять випадковий характер і засновані на інтуїції про те, як користувачі вибирають паролі, а не на основі принципового аналізу великих наборів паролів. З цієї причини кожна техніка в кінцевому підсумку обмежується захопленням певного підмножини простору паролів, яке залежить від інтуїції, що стоїть за цією технікою. Крім того, розробка і тестування нових правил і евристик – трудомістке завдання, що вимагає спеціальних знань і, отже, обмежена масштабованість.

Щоб усунути ці недоліки, пропоную замінити вгадування паролів на основі правил, а також вгадування паролів на основі простих методів, керованих даними, таких як моделі Маркова, на новий підхід, заснований на глибокому навчанні. За своєю суттю ідея полягає в навчанні нейронних мереж автономно визначати характеристики і структури паролів і використовувати ці знання для створення нових зразків, які слідують тому ж розподілу. Припускаю, що глибокі нейронні мережі досить виразні, щоб охопити велику різноманітність

властивостей і структур, які описують більшість обраних користувачем паролів, в той же час нейронні мережі можна навчати без будь-яких апріорних знань або припущень про такі властивості і структури. Це різко контрастує з сучасними підходами, такими як марковські моделі і підходи, засновані на правилах (які можуть вгадувати тільки ті паролі, які відповідають доступним правилам). В результаті вибірки, що генеруються з використанням нейронної мережі, не обмежуються конкретним підмножиною простору паролів. Замість цього нейронні мережі можуть автономно кодувати широкий спектр знань з підбору паролів, які включають і перевершують те, що фіксується в створених людиною правилах і процесах генерації марковських паролів.

Новий підхід для генерації парольних припущень, заснований на глибокому вивченні і генеруючих змагальних мережах (*GAN*).

GAN – це нещодавно представлені інструменти машинного навчання, призначені для оцінки щільності в багатовимірному просторі. *GAN* виконують неявне генеративне моделювання, навчаючи архітектуру глибокої нейронної мережі, яка отримує простий випадковий розподіл (наприклад, Гаусса або рівномірний), і генеруючи вибірки, які слідують за розподілом доступних даних.

Для навчання генеративної структури *GAN* використовують гру в кішки-мишки, в якій глибока генеративна мережа (*G*) намагається імітувати паролі користувачів, а дискримінаційна глибока нейронна мережа (*D*) намагається розрізнити вихідні навчальні вибірки (тобто «справжні вибірки») і вибірки, згенеровані *G* (тобто «підроблені вибірки»), змагальна процедура змушує *D* до витоку відповідної інформації для *G*, щоб ефективно імітувати вихідний розподіл даних.

3.2. Розпізнавання паролів

При атаці за допомогою підбору пароля зловмисник намагається ідентифікувати пароль одного або декількох користувачів шляхом багаторазового тестування декількох можливих паролів. Атаки за допомогою підбору пароля, ймовірно, настільки ж старі, як і самі паролі, а більш формальні дослідження датуються 1979 роком.

Два популярних сучасних засоби підбору паролів: *HashCat* та *john*. Обидва інструменти реалізують різні типи стратегій підбору пароля, включаючи: вичерпні атаки методом перебору, атаки по словнику, атаки на основі правил, які полягають в генерації здогадок з перетворень словникових слів, атаки по марковській моделі, в якому кожен символ пароля можна вибрати за допомогою стохастичного процесу, який враховує один або кілька попередніх символів, і який навчається на словниках незашифрованих паролів. *JTR* і *HashCat* особливо ефективні при вгадуванні паролів. Зокрема, було кілька випадків, коли більше 90% пароля, що просочився з онлайн-сервісів, були успішно відновлені.

Марковські моделі були вперше використані для генерації парольних припущень *Narayanan et al*. Їх підхід використовує правила паролів, які визначені вручну, наприклад, яка частина згенерованих паролів складається з букв і цифр. Ця методика була згодом поліпшена *Weir et al*, який показав, як «вивчити» ці правила з паролів. Ця рання робота була згодом розширена *Ma et al. і Durmuth et al*. Методи, засновані на марковських моделях, також використовувалися для реалізації оцінок надійності паролів в реальному часі і для оцінки надійності паролів в базах даних в незашифрованому вигляді.

Нещодавно *Melicher et al*. представили *FLA*, метод підбору пароля на основі періодичних нейронних мереж. За допомогою цієї техніки нейронна мережа навчається з використанням паролів, отриманих з декількох веб-сайтів. Під час генерації пароля нейронна мережа виводить один символ пароля за раз. Кожен новий символ (включаючи спеціальний символ кінця пароля) вибирається на основі його ймовірності, з огляду на поточний стан виведення, в тому, що по суті є марковским процесом. (Це властивість була використана в основному для оцінки надійності пароля в реальному часі). Основна мета *Melicher et al*. з *FLA* забезпечує швидку і точну оцінку надійності пароля, зберігаючи при цьому максимально легку модель і без шкоди для точності. Таким чином, при наявності навченої моделі *FLA* можна подати набір паролів в модель і витягти в якості вихідного файлу файл, який містить 6 полів, організованих таким чином:

– пароль;
– ймовірність паролю;

– передбачувана кількість вихідних припущень (надійність пароля);

– стандартне відхилення рандомізованого дослідження для цього пароля (в одиницях кількості припущень);

– кількість вимірювань для пароля;

– розрахунковий інтервал для числа припущень (в одиницях кількості припущень).

Оцінка показує, що методика перевершує марковські моделі і правила складання паролів, які зазвичай використовуються в *JTR* і *HashCat*, при тестуванні великої кількості припущень пароля (в діапазоні від 10^{10}до 10^{25}).

Відмінності між можливостями виведення *FLA* і створеного методу в дипломній роботі обумовлені марковською структурою процесу генерації пароля в *FLA*. Через ці властивості будь-яка характеристика пароля, що не охоплена в межах простору, може не кодуватися *FLA*. Наприклад, якщо значуще підмножина паролів з 10 символів будується як об'єднання двох слів (наприклад, *MusicMusic*), будь-який марковский процес з $n \leq 5$ не зможе правильно відобразити цю поведінку. З іншого боку, при достатній кількості прикладів нейронна мережа, яка використовується в дипломній роботі, зможе вивчити ці властивості. В результаті, в той час як *FLA* привласнила паролю *pookypooky* ймовірність $p = 10^{-33}$ (з передбачуваним числом спроб вгадати близько 10^{29}), це було вгадано після приблизно 10^{8} спроб в дипломній роботі.

3.3. Структура програмного засобу

Дипломна робота розроблена на основі *IWGAN*, а генератор та дискримінаторо використовує *CNN*. На фото нижче показана архітектура програмного засобу (рис. 3.1):

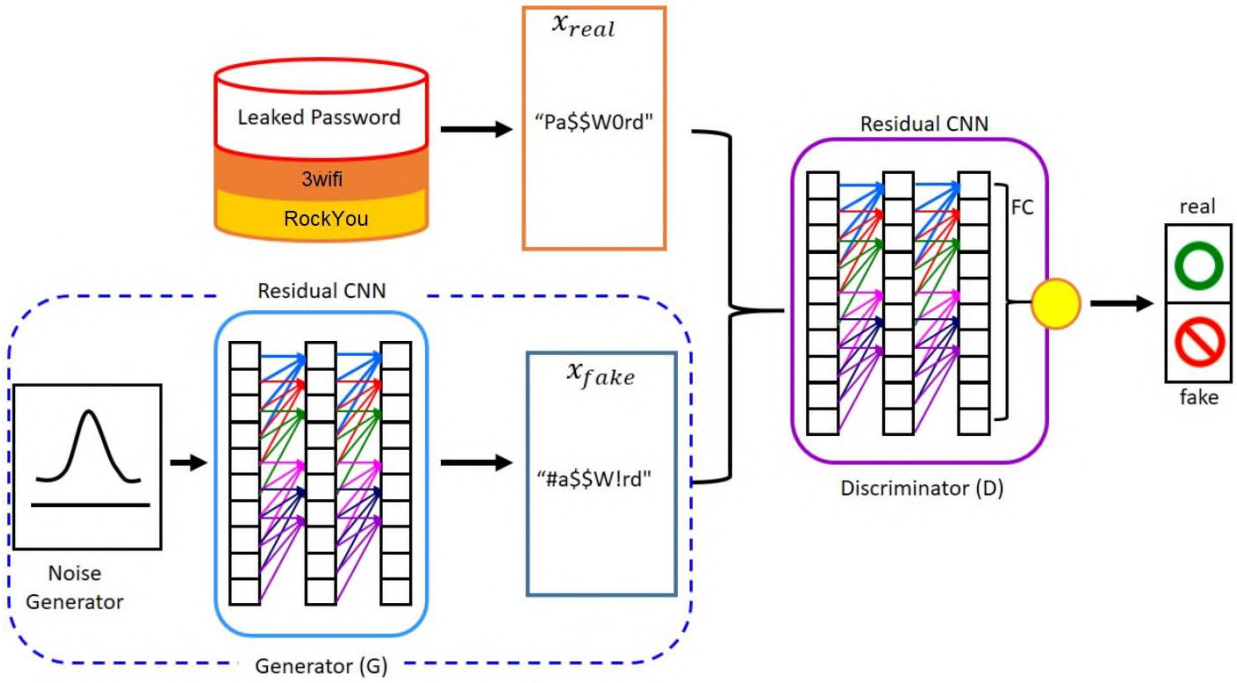

Рис. 3.1. Структурна схема програного засобу

CNN складається з шести *Residual Block* (залишкові блоки). На рисунку 3.2 показана структура одного залишкового блоку.

Експерименти проводились із використанням реалізації *IWGAN TensorFlow*. Всі експерименти проводились на робочій станції під управлінням *Windows* 10, 32 гігабайти оперативної пам'яті, процесор *Ryzen* 3600 та графічним процесором *NVIDIA GeForce GTX* 1080 *Ti*. Будівельними блоками, що використовуються для побудови *IWGAN* є залишкові блоки, приклад яких наведено на рисунку 3.2. Вони є центральним компонентом залишкових мереж. При тренуванні глибокої нейронної мережі спочатку помилка тренування зменшується із збільшенням кількості шарів. Однак після досягнення певної кількості шарів похибка навчання знову починає зростати.

На відміну від інших глибоких нейронних мереж, *ResNet* включає «швидке з'єднання» між рівнями. Це можна розглядати як обгортку для цих шарів і реалізується як функція ідентифікації (залишковий блок). Використовуючи кілька послідовних залишкових блоків, *ResNet* постійно зменшує похибку навчання, оскільки кількість шарів збільшується.

Залишкові блоки складаються з двох одновимірних загорткових шарів, з'єднані між собою функціями активації випрямлених лінійних одиниць (*ReLU*), (рис. 3.2). Вхід блоку є функцією ідентифікації і збільшується на 0.3 помножене

на вихід згорткових шарів для отримання вихідних даних блоку.

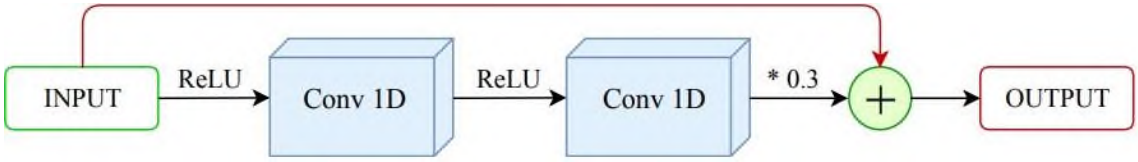

Рис. 3.2. Структура *Residual Block*

Фрагмент програмного коду *Residual Block*:

```
def ResidualBlock(name, inputs, dim):
    # print("- Creating ResidualBlock -")
    output = inputs
    output = tf.nn.relu(output)
    output = lib.ops.conv1d.Conv1D(name+'.1', dim, dim, 6, output)
    # print("After conv:", output)
    output = tf.nn.relu(output)
    output = lib.ops.conv1d.Conv1D(name+'.2', dim, dim, 6, output)
    return inputs + (0.3*output)
```

Так як генератор та дискримінатор основані на *CNN* в їх склад входить по шість залишкових блоків (рис. 3.3).

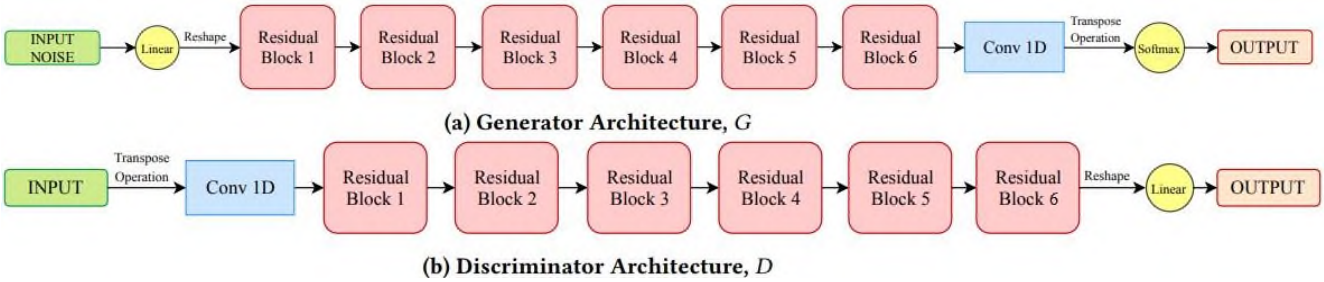

Рис. 3.3. Структура дискримінатора та генератора

Під час тренувальної процедури дискримінатор *D* обробляє паролі з набору навчальних даних, а також зразки паролів, створені генератором *G*. На основі зворотного зв’язку від *D*, *G* налаштовує свої параметри для отримання зразків паролів, які розподіляються аналогічно зразкам у навчальному наборі. У дипломній роботі, після завершення навчальної процедури, використано *G*, щоб генерувати вгадування пароля, а на вхід дискримінатора подається вхідний шум. Фрагмент програмного коду дискримінатора та генератора:

```
def Generator(n_samples, seq_len, layer_dim, output_dim, prev_outputs=None):
    # print("- Creating Generator -")
```

```
    output = make_noise(shape=[n_samples, 128])
    # print("Initialized:", output)
    output = lib.ops.linear.Linear('Generator.Input', 128, seq_len * layer_dim, output)
    # print("Lineared:", output)
    output = tf.reshape(output, [-1, seq_len, layer_dim,])
    # print("Reshaped:", output)
    output = ResidualBlock('Gen.1', output, layer_dim)
    output = ResidualBlock('Gen.2', output, layer_dim)
    output = ResidualBlock('Gen.3', output, layer_dim)
    output = ResidualBlock('Gen.4', output, layer_dim)
    output = ResidualBlock('Gen.5', output, layer_dim)
    output = ResidualBlock('Gen.6', output, layer_dim)
    output = lib.ops.conv1d.Conv1D('Gen.Output', layer_dim, output_dim, 1, output)
    output = softmax(output, output_dim)
    return output

def Discrimitator(inputs, seq_len, layer_dim, input_dim):
    output = inputs
    output = lib.ops.conv1d.Conv1D('Discriminator.Input', input_dim, layer_dim, 1,
output)
    output = ResidualBlock('Discriminator.1', output, layer_dim)
    output = ResidualBlock('Discriminator.2', output, layer_dim)
    output = ResidualBlock('Discriminator.3', output, layer_dim)
    output = ResidualBlock('Discriminator.4', output, layer_dim)
    output = ResidualBlock('Discriminator.5', output, layer_dim)
    output = ResidualBlock('Discriminator.6', output, layer_dim)
    output = tf.reshape(output, [-1, seq_len * layer_dim])
    output = lib.ops.linear.Linear('Discriminator.Output', seq_len * layer_dim, 1,
output)
    return output
```

3.4. Параметри для тренування моделі

Щоб використовувати здатність GAN ефективно оцінювати розподіл паролів з навчального набору, проведено експерименти з різними параметрами. В цьому розділі описано про вибір гіперпараметрів.

Наступні гіперпараметри характеризують модель:

– розмір пакета (*batch size)*, представляє кількість паролів з навчального набору, які поширюються через *GAN* на кожному кроці оптимізатора. Обрано розмір парті 128.

– число ітерацій (*number of iterations*) вказує, скільки разів *GAN* викликає свій крок вперед і крок зворотного поширення. У кожній ітерації *GAN* виконує одну ітерацію генератора і одну або декілька ітерацій дискримінатора. Кількість операцій можна обрати довільною. За основу обрано 170000 ітерацій, оскільки подальше збільшення не призводило до покращення точності.

– число ітерацій дискримінатора на одну ітерацію генератора (*number of discriminator iterations per generator iteration*) вказує, скільки ітерацій дискримінатор виконує в кожній ітерації *GAN*. Кількість ітерацій дискримінатора на генеративну ітерацію було встановлено рівним 10, що є значенням за замовчуванням, яке використовує *IWGAN*.

– розмірність моделі (*model dimensionality*), яка представляє кількість вимірювань для кожного загорткового шару. Проведено експеримент з використанням 6 залишкових верств як для генератора, так і для дискримінатора, причому кожен з шарів в обох глибоких нейронних мережах мав 128 вимірювань.

– коефіцієнт штрафу градієнта (λ) (*gradient penalty coefficient*), визначає штраф, який застосовується до норми градієнта дискримінатора щодо його вхідних даних. Збільшення цього параметра призводить до більш стабільної тренуванні. Штраф градієнта встановлений на рівні 10.

– довжина вихідний послідовності (output sequence length), вказує максимальну довжину рядків, що генеруються генератором (*G*). Вказує максимальну довжину паролів які будуть згенеровані. Розмір можна обрати.

– розмір вектора вхідного шуму (*Size of the input noise vector (seed)*), визначає, скільки випадкових чисел з нормального розподілу подається в якості вхідних даних в *G* для генерації вибірок. Встановлено розмір на 128 чисел з плаваючою точкою.

– максимальна кількість прикладів (*Maximum number of examples*), представляє максимальну кількість навчальних елементів (паролів) для завантаження. Максимальна кількість прикладів, що завантажуються *GAN*, було встановлено рівним розміру всього набору навчальних даних.

Гіпер-параметри оптимізатора *Adam* (*Adam optimizer's hyper-parameters):*

– швидкість навчання, як швидко коригуються ваги моделі;

– коефіцієнт *β1*, який вказує швидкість убування змінного середнього градієнта;

– коефіцієнт *β2*, який вказує швидкість убування змінного середнього квадрата градієнта.

Коефіцієнти *β1* і *β2* оптимізатора *Adam* встановлені на 0.5 та 0.9 відповідно, а швидкість навчання становила 0.0005 для генератора та 0.0015 для дискримінатора.

3.5. Реалізація програмної частини

3.5.1 Модуль підготовки вхідних данних

Тренування моделі відбувається на основі вхідних даних. Підготовка вхідних даних є важливою процедурою. На вхід порібно надати текстовий файл з паролями. Модуль підготовки розподілить данний файл на два файли 20% паролів у першому та 80% у другому. Це необхідно для розподілу даних на дані для тренування та дані для тестування. Вхідні дані перемішуються рандомним чином. Є можливість відфільтрувати паролі до 10 символів. Фрагмент коду:

```
import sys
import random
random.seed(1337)
with open('3wifi.txt', 'r') as f:
```

```
    lines = f.readlines()
    # filter only passwords with 10 characters or more
    lines = filter(lambda x: len(x) >= 10, lines)

    # randomize order
    print('[info] shuffling data')
    random.shuffle(lines)
    split = int(len(lines) * 0.80)
    with open('../data/train.txt', 'w', encoding="ISO-8859-1") as f:
        print('[info] saving 80% ({}) of dataset for training in ../data/train.txt'.format(split))
        f.write(''.join(lines[0:split]))
    with open('../data/test.txt', 'w', encoding="ISO-8859-1") as f:
        print('[info] saving 20% ({}) of dataset for testing in ../data/test.txt'.format(len(lines) - split))
        f.write(''.join(lines[split:]))
```

Важливою частиною є підготовка вхідного словника. Бажано очистити від цифрових паролів. Найчастіше відсоток числових паролів досить великий. Що не добре впливає на тренування і вихідні дані.

3.5.2 Модуль тренування моделі

В даному модулі відбувається тренування моделі по заданим параметрам. Користувач має можливість вводити параметри для тренування моделі, цим самим зменшуючи або збільшуючи час та точність тренування.

Фрагмент коду:

```
parser.add_argument('--training-data', '-i',
                    default='data/train.txt',
                    dest='training_data',
                    help='Path to training data file (one password per line) (default: data/train.txt)')
    parser.add_argument('--output-dir', '-o',
```

```
                required=True,
                dest='output_dir',
                help='Output directory. If directory doesn\'t exist it will be created.')
    parser.add_argument('--save-every', '-s',
                type=int,
                default=5000,
                dest='save_every',
                help='Save model checkpoints after this many iterations (default:
    5000)')
```

Для збільшення стабільності змінюємо параметр *learning_rate*, значення за умовчанням 0.001. (Стабільність збільшується, але для досягнення результату знадобиться більша кількість ітерацій навчання. Змінювати параметр варто в залежності від цілей. *Batch-size* – при тренувані великого словника, більше ніж 100МБ, та на графічному адаптері обсяг пам’яті не більше 4ГБ то варто зменишити розмір партії та відновити тренування.

Проведемо тестування змінюючи лише один параметр *learning_rate*.

На рисунку 3.4 зображено вихідні згенеровані паролі на 1000 кроці навчання. Зліва на право *learning_rate* = 0.001, 0.0001, 0.00001.

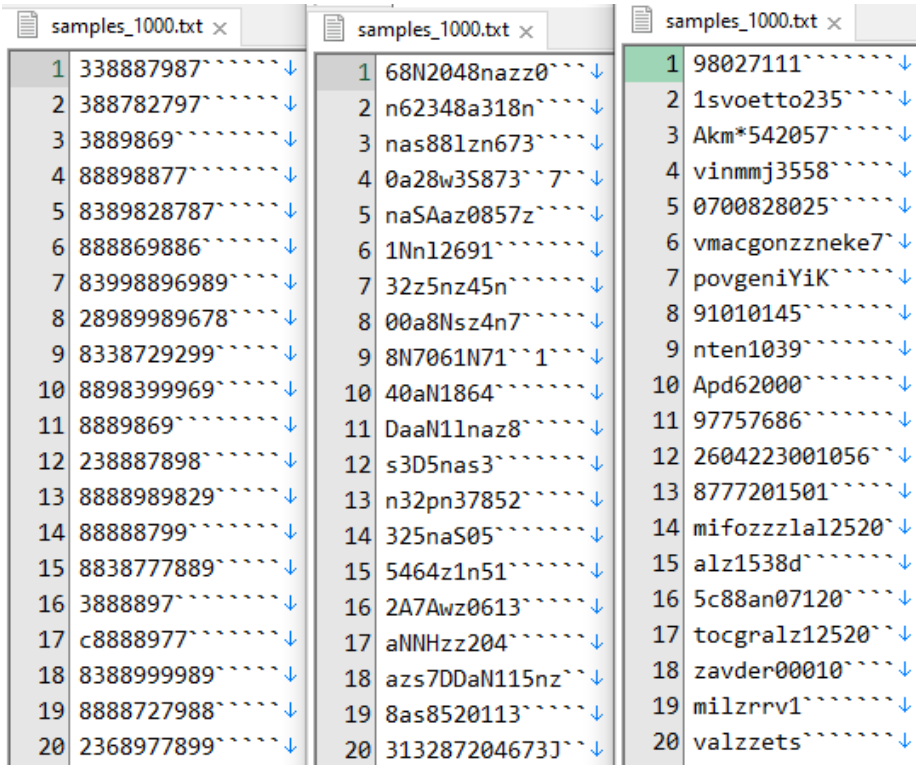

Рис. 3.4. Вплив параметра *learning_rate* на вихідні дані

Але потрібно пам’ятати, що у швидкого навчання є слабка сторона – стабільність.

Завантаження даних для тренування:

```
lines, charmap, inv_charmap = utils.load_dataset(
    path=args.training_data,
    max_length=args.seq_length,
    )
```

Вивід на екран кількості унікальни символів у вхідному наборі:

```
with open(os.path.join(args.output_dir, 'charmap.pickle'), 'wb') as f:
    pickle.dump(charmap, f)
with open(os.path.join(args.output_dir, 'charmap_inv.pickle'), 'wb') as f:
    pickle.dump(inv_charmap, f)
print("Number of unique characters in dataset: {}".format(len(charmap)))
print("characters: ", charmap)
```

Реалізація ітератору набора даних:

```
def inf_train_gen():
    while True:
        np.random.shuffle(lines)
        for i in range(0, len(lines)-args.batch_size+1, args.batch_size):
            yield np.array(
                [[charmap[c] for c in l] for l in lines[i:i+args.batch_size]],
                dtype='int32'
            )
```

Зберігаємо приклади виводу кожні 100 кроків, також оновлюємо модель.

```
# Output to text file after every 100 samples
    if iteration % 100 == 0 and iteration > 0:
        samples = []
        for i in range(10):
            samples.extend(generate_samples())
        for i in range(4):
            lm = utils.NgramLanguageModel(i+1, samples, tokenize=False)
            lib.plot.plot('js{}'.format(i+1), lm.js_with(true_char_ngram_lms[i]))
```

```
        with open(os.path.join(args.output_dir, 'samples',
'samples_{}.txt').format(iteration), 'w') as f:
            for s in samples:
                s = "".join(s)
                f.write(s + "\n")
    if iteration % args.save_every == 0 and iteration > 0:
        model_saver = tf.train.Saver()
        model_saver.save(session, os.path.join(args.output_dir, 'checkpoints',
'checkpoint_{}.ckpt').format(iteration))
        print("{} / {} ({}%)".format(iteration, args.iters, iteration/args.iters*100.0 ))
    if iteration == args.iters:
        print("...Training done.")
    if iteration % 100 == 0:
        lib.plot.flush()
    lib.plot.tick()
```

3.5.3 Модуль генерації вихідних даних

Для генерації даних на основі створеної моделі необхідно вказати шлях до директорії де зберігається модель, назву вихідного файлу, файл точки відновлення, кількість прикладів для генерації, розмір пакету для генерації, максимальну довжину пароля для генерації та кількість прихованих слоїм мережі.

```
with tf.Session() as session:
  def generate_samples():
    samples = session.run(fake_inputs)
    samples = np.argmax(samples, axis=2)
    decoded_samples = []
    for i in range(len(samples)):
        decoded = []
        for j in range(len(samples[i])):
            decoded.append(inv_charmap[samples[i][j]])
```

```
            decoded_samples.append(tuple(decoded))
        return decoded_samples
    def save(samples):
        with open(args.output, 'a') as f:
            for s in samples:
                s = "".join(s).replace('`', '')
                f.write(s + "\n")
    saver = tf.train.Saver()
    saver.restore(session, args.checkpoint)
    samples = []
    then = time.time()
    start = time.time()
    for i in range(int(args.num_samples / args.batch_size))
        samples.extend(generate_samples())
```

Збереження вихідного файлу відбувається в залежності від розміру вказаного пакету:

```
        # every 1000 batches
        if i % 1000 == 0 and i > 0:
            save(samples)
            samples = [] # flush
            print('wrote {} samples to {} in {:.2f} seconds. {} total.'.format(1000 *
args.batch_size, args.output, time.time() - then, i * args.batch_size))
            then = time.time()
```

3.6. Висновки до розділу

В даному розділі описано постановку задачі. Створено модуль для підготовки вхідних даних, модуль для тренування моделі та модуль для генерації вихідних даних. Наведено опис архітектури системи з детальними поясненнями. Підібрано та описано параметри для тренування моделі. Налаштовано оптимізатор *Adam*.

РОЗДІЛ 4
ВИКОРИСТАННЯ ПРОГРАМНОГО ЗАСОБУ

4.1. Вимоги до апаратно-програмного забезпечення

4.1.1. Вимоги до програмного забезпечення

Одним з результатів дипломної роботи є налаштоване робоче середовище, яке можна переносити на інші обчислювальні пристрої. Потрібно мати лише встановлене програмне забезпечення *Anaconda*.

За умови встановлення кожного пакету окремо необхідно дотримуватись певних версій пакетів:

Версія *Tenorflow* має бути 1.4.1 для *CPU* та *GPU, matplotlib* не нижче 2.1.1, *numpy* не нижче 1.13.3, Python не нижче 3.6.

4.1.2. Вимоги до апаратного забезпечення

Процес тренування моделі потребує ресурсів графічного процесору. Рекомендовано використовувати *GPU* з не менш ніж 3 ГБ пам'яті. Для тренування моделі на великих наборах оптимальним буде вибір *GPU з* 11 ГБ пам'яті. В залежності від обраних параметрів для тренування потрібно більший або менший обсяг оперативної пам'яті. Рекомендований мінімум 12 ГБ ОЗУ.

Кількість потоків центрального процесору буде впливати на швидкість передачі даних. Рекомендований мінімум *Ryzen* 3100 або *Intel i3* 10100.

Для підсистеми збереження даних рекомендовано мати ssd диск.

4.2. Запуск програмного засобу

Для початку роботи з програмним засобом потрібно встановити необхідне програмне забезпечення.

Необхідно запустити *Anaconda Navigator* (Рис. 4.1).

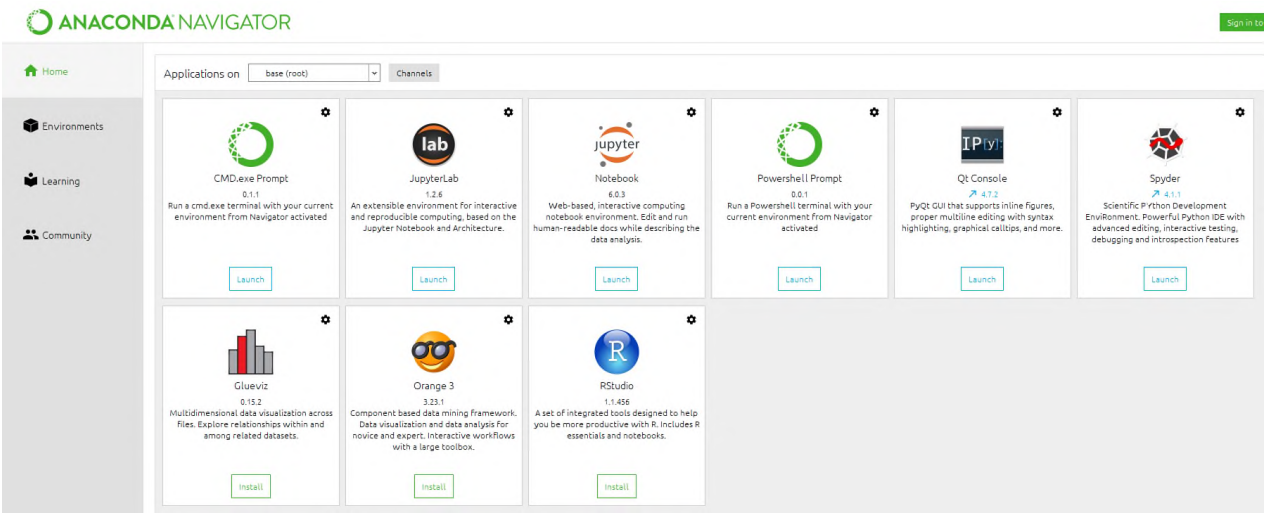

Рис. 4.1. Anaconda Navigator

В середовищі *Anaconda Navigator* необхідно обрати *Powershell/Prompt* (Рис 4.2). Для переходу в режим командного рядка.

Рис. 4.2. *Powershell*

Для активація підготовленого віртуального середовища зі встановленим програмним забезпеченням необхідно виконати наступну команду:

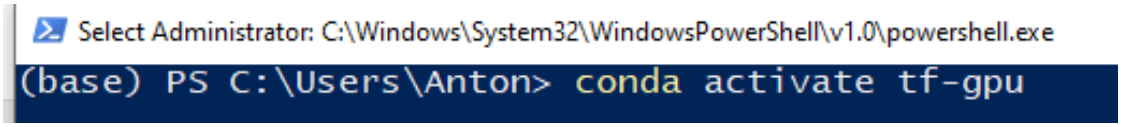

Рис. 4.3. Активація віртуального середовища

4.3. Робота користувача з додатком

Для початкового очищення та розбиття словника на необхідні файли потрібно використати скрипт *preparingData.* Вказавши необхідний файл з вхідними даними.

Після підготовки вхідних даних, необхідно приступити до тренування моделі на основі нейромережі *GAN.*

Етап введення параметрів для тренування моделі. В кожного параметру є значення за умовчанням, якщо параметр не буде вказаний користувачем.

Для користувача доступні наступні параметри:

1. «*--training-data*» або «*-i*» – шлях до файла для тренування. (за умовчанням «*data/train.txt*»).
2. «*--output-dir*» або «*-o*» – шлях до вихідних файлів.
3. «*--save-every*» або «*-s*» – створення точки відновлення для моделі. Параметр вказує кількість ітерацій після якої необхідно зберігати модель.
4. «*--iters*» або «*-n*» – кількість ітерацій для тренування моделі (за умовчанням 160000).
5. «*--batch-size*» або «*-b*» – розмір пакета (за умовчанням 128).
6. «*--seq-length*» або «*-l*» – максимальна довжина паролів які будуть згенеровані. (за умовчанням 16).
7. «*--layer-dim*» або «*-d*» – кількість прихованих шарів для генератора та дискримінатора (за умовчанням 128).
8. «*--critic-iters*» або «*-c*» – вказує кількість оновлень дискримінатора на одне оновлення генератора. (за умовчанням 10).
9. «*--lambda*» або «*-p*» – штраф який застосовується до норми градієнта дискримінатора відносно вхідних даних.

Наступна команда запустить на виконання тренування моделі зі вказаними параметрами (Рис. 4.4):

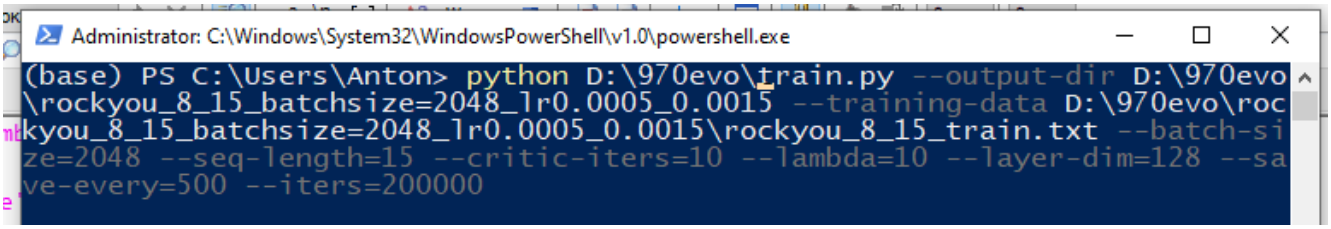

Рис. 4.4. Запуск тренування моделі

Після початку виконання вказаної команди користувач побачить кількість завантажених паролів для навчання. А також число унікальних символів в завантаженому наборі (Рис. 4.5):.

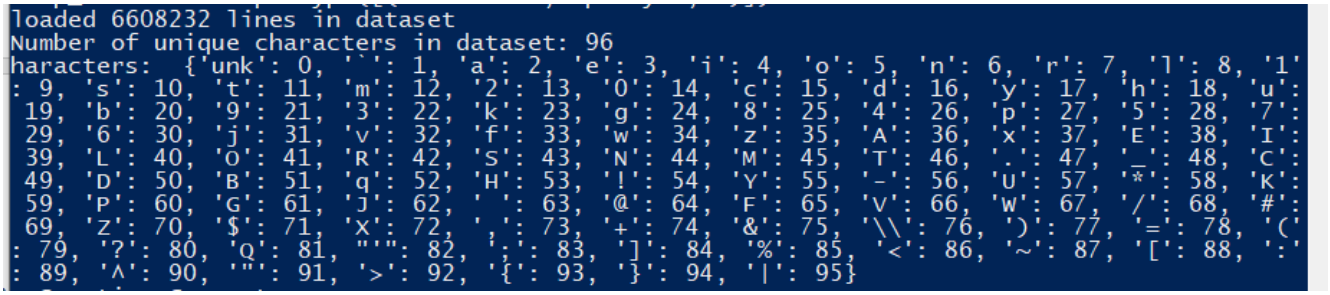

Рис. 4.5. Кількість паролів та число унікальних символів в наборі

Також на ерані відображено результат створення *Residual Block*, відповідно до вказаних параметрів користувача.

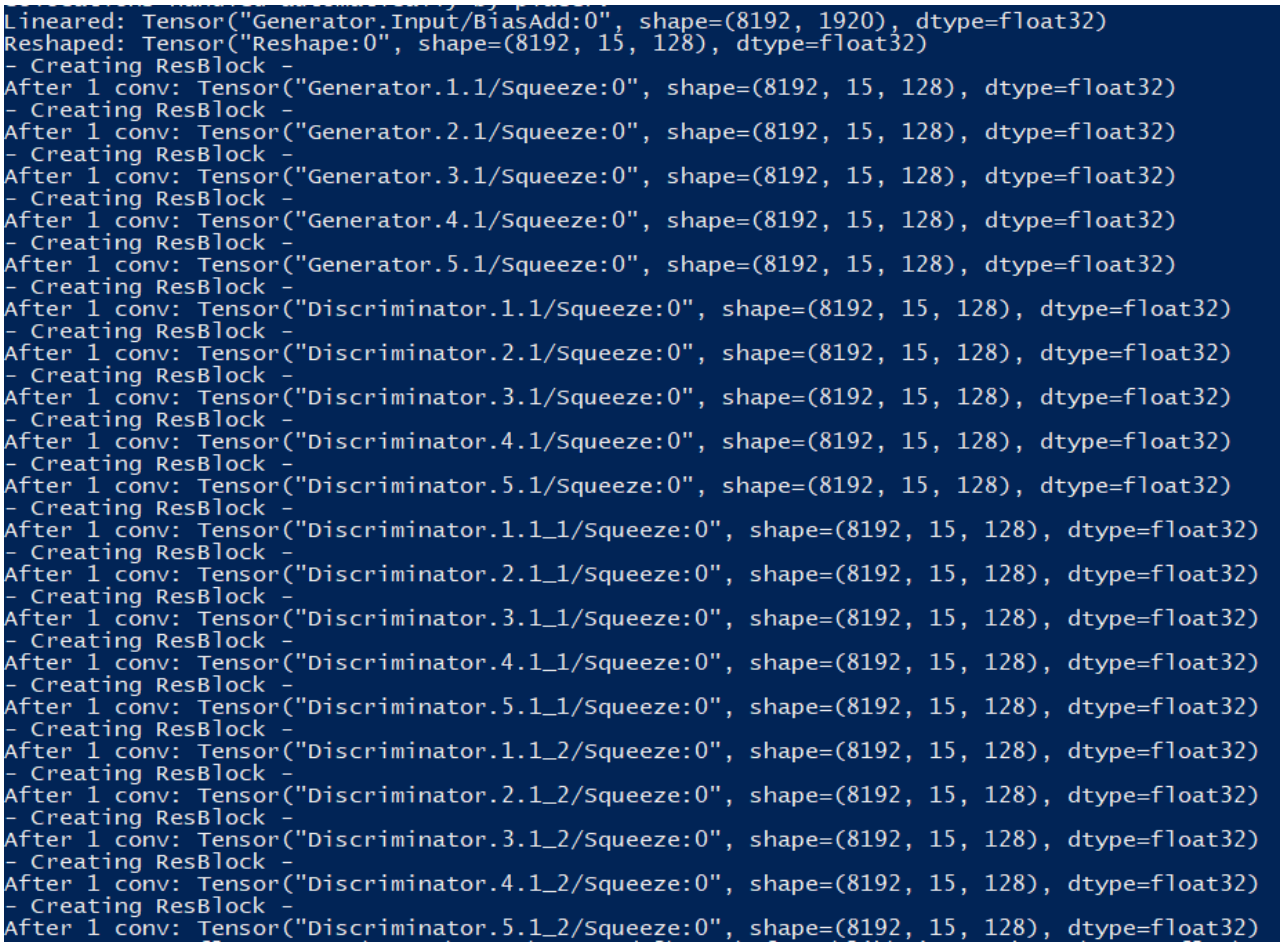

Рис. 4.6. Створення *Residual Block*

Обраний графічний процесор буде відображено в консолі, а також частота та кількість пам’яті яку виділено для процесу навчання (Рис. 4.6).

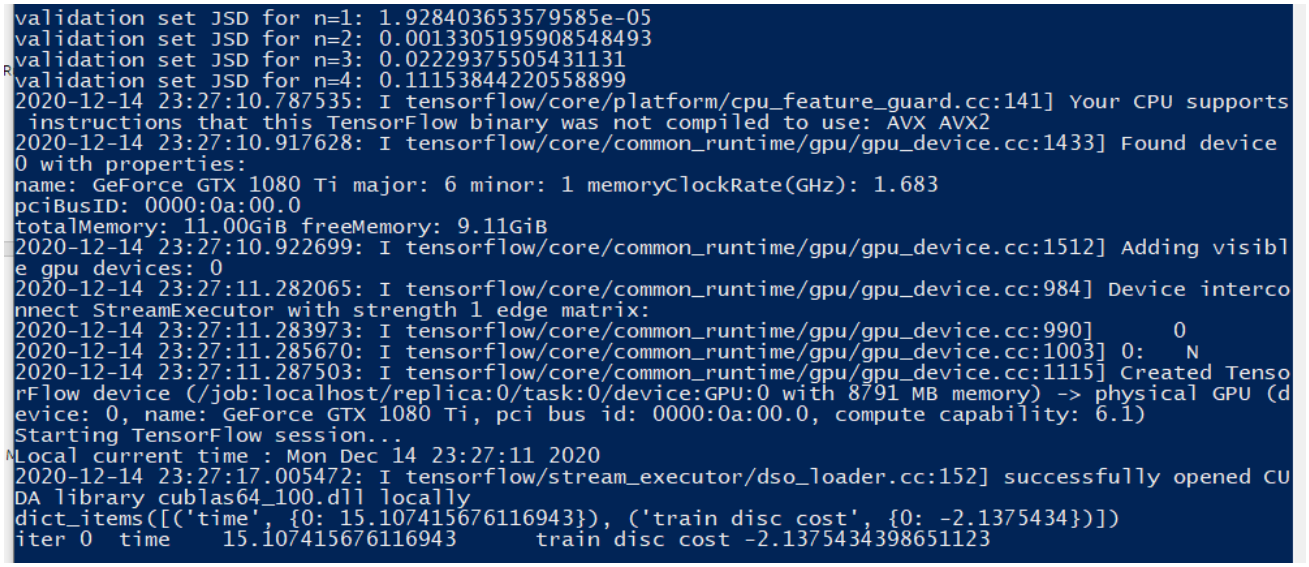

Рис. 4.6. Валідація даних та обраний графічний процесор

Після надходження вхідних даних та графічного адаптера відбувається процес тренування моделі. Користувач має можливість спостерігати за результатом тренування (Рис. 4.7).

Рис. 4.7. Процес тренування моделі

Файли моделі зберігаються на основі вказаних параметрів, наприклад, якщо користувач вказав зберігати кожні 1000 кроків, можна побачити наступний результат (Рис. 4.8).

Name	Date modified	Type	Size
checkpoint	29.07.2020 00:09	File	1 KB
checkpoint_1000.ckpt.data-00000-of-00001	27.07.2020 20:42	DATA-00000-OF-0...	22 641 KB
checkpoint_1000.ckpt.index	27.07.2020 20:42	INDEX File	6 KB
checkpoint_1000.ckpt.meta	27.07.2020 20:42	META File	8 580 KB
checkpoint_2000.ckpt.data-00000-of-00001	27.07.2020 23:44	DATA-00000-OF-0...	22 641 KB
checkpoint_2000.ckpt.index	27.07.2020 23:44	INDEX File	6 KB
checkpoint_2000.ckpt.meta	27.07.2020 23:44	META File	8 639 KB
checkpoint_3000.ckpt.data-00000-of-00001	28.07.2020 02:48	DATA-00000-OF-0...	22 641 KB
checkpoint_3000.ckpt.index	28.07.2020 02:48	INDEX File	6 KB
checkpoint_3000.ckpt.meta	28.07.2020 02:48	META File	8 698 KB
checkpoint_4000.ckpt.data-00000-of-00001	28.07.2020 05:50	DATA-00000-OF-0...	22 641 KB
checkpoint_4000.ckpt.index	28.07.2020 05:50	INDEX File	6 KB
checkpoint_4000.ckpt.meta	28.07.2020 05:51	META File	8 756 KB
checkpoint_5000.ckpt.data-00000-of-00001	28.07.2020 08:53	DATA-00000-OF-0...	22 641 KB
checkpoint_5000.ckpt.index	28.07.2020 08:53	INDEX File	6 KB
checkpoint_5000.ckpt.meta	28.07.2020 08:53	META File	8 815 KB
checkpoint_6000.ckpt.data-00000-of-00001	28.07.2020 11:57	DATA-00000-OF-0...	22 641 KB
checkpoint_6000.ckpt.index	28.07.2020 11:57	INDEX File	6 KB
checkpoint_6000.ckpt.meta	28.07.2020 11:57	META File	8 874 KB
checkpoint_7000.ckpt.data-00000-of-00001	28.07.2020 15:03	DATA-00000-OF-0...	22 641 KB
checkpoint_7000.ckpt.index	28.07.2020 15:03	INDEX File	6 KB
checkpoint_7000.ckpt.meta	28.07.2020 15:03	META File	8 933 KB
checkpoint_8000.ckpt.data-00000-of-00001	28.07.2020 18:05	DATA-00000-OF-0...	22 641 KB
checkpoint_8000.ckpt.index	28.07.2020 18:05	INDEX File	6 KB
checkpoint_8000.ckpt.meta	28.07.2020 18:05	META File	8 992 KB

Рис. 4.8. Результат збереженої моделі

За текстовим відображенням тренування не зручно спостерігати, тому реалізовано генерація тестових даних в процесі навчання (Рис. 4.9). Тобіж можна наглядно спостерігати процес покращення моделі, так як, згенеровані паролі будуть більш схожі ті, які створені людьми.

Name	Date modified	Type	Size
samples_100.txt	27.07.2020 17:57	Text Document	1 440 KB
samples_200.txt	27.07.2020 18:16	Text Document	1 440 KB
samples_300.txt	27.07.2020 18:34	Text Document	1 440 KB
samples_400.txt	27.07.2020 18:52	Text Document	1 440 KB
samples_500.txt	27.07.2020 19:10	Text Document	1 440 KB
samples_600.txt	27.07.2020 19:29	Text Document	1 440 KB
samples_700.txt	27.07.2020 19:47	Text Document	1 440 KB
samples_800.txt	27.07.2020 20:05	Text Document	1 440 KB
samples_900.txt	27.07.2020 20:23	Text Document	1 440 KB
samples_1000.txt	27.07.2020 20:42	Text Document	1 440 KB

Рис. 4.9. Приклади вихідних даних в процесі тренування

4.4. Висновки до розділу

В даному розділі описано використання програмного засобу перевірки безпеки інформаційних систем засобами нейромережі *GAN*. Надано вимоги до апаратного та програмного забезпечення для коректного функціонування. Описано процес першого запуску програмного засобу. Надано пояснення для користувача по роботі з програмним засобом, приклади виводу програмного засобу.

ВИСНОВКИ

Проаналізовано сучасні методи перевірки інформаційної безпеки, архітектури нейромереж для перевірки безпеки інформаційних систем, вхідні та вихідні дані для тренування. Розв’язано наступні задачі: розроблено модуль підготовки вхідних даних, розроблено модуль тренування моделі, розроблено модуль генерації вихідних данних, розроблено структурну схему моделі, розроблено інтерфейс користувача, розроблено модель на основі нейромережі *GAN*, вибір оптимальних параметрів для тренування моделі;

Виконано перевірку вихідних даних з правилами, порівняння з існуючими програмними засобами.

Для розв’язання задача використано набір інструментів та команд мови програмування *Python*, фреймворк *TensorFlow*. Використано віртуальне середовище на основі дистрибутиву *Anaconda.* Для візуалізація процесу тренування використано пакет *Matplotlib.* Для методу виміру двух схожих ймовірностей використано метод дивергенції Йенсена-Шеннона. Для оптимізації навчання обрано метод *Adam.* Для структури програмного засобу використано покращений *GAN*, а саме – *Improved Training of Wasserstein GAN*.

Існуючі методи підбору паролів на основі правил ефективні, але обмежені. Основний недолік використання паролів на основі правил в тому, що правила можуть генерувати лише кінцевий, відносно невеликий набір паролів. У результаті цього розроблений програмний засіб зміг перейти до кількості паролів, отриманих за допомогою правил генерації паролів.

Найкраща стратегія для тестування паролів на стійкість включає в себе використання декількох інструментів. Результати підтверджують, що об'єднання кількох методів приводить до найкращої загальної продуктивності.

Програмний засіб наразі вимагає виведення більшої кількості паролів у порівнянням з іншими інструментами. Ця вартість незначна при розгляді переваг запропонованих методик. Крім того, навчання більш широкому набору даних

дозволяє використовувати більш складні структури нейронних систем та проводити більш складне навчання.

Наукова новизна отриманих результатів. У роботі представлено техніку вгадування пароля, засновану на генеративних змагальних мережах (*GAN*). Програмний засіб призначений для вивчення інформації про розповсюдження паролів через витоки паролів. Як результат, на відміну від поточних інструментів вгадування паролів, програмний засіб не покладається на будь-яку додаткову інформацію, таку як явні правила, або припущення щодо марковійської структури вибраних користувачем паролів. Даний підхід до вгадування паролів є революційним, оскільки програмний засіб генерує паролі без втручання користувача – таким чином, не вимагаючи знання предметної області щодо паролів, а також ручного аналізу витоків бази даних паролів. Результати показують, що програмний засіб конкурує з найсучаснішими інструментами генерації паролів. Програмний засіб завжди міг генерувати таку ж кількість збігів, як і інші інструменти вгадування паролів. Однак, в даний час вимагає виведення більшої кількості паролів порівняно з іншими інструментами. Ця вартість є незначною, враховуючи переваги запропонованої техніки. Крім того, навчання на більшому наборі даних дозволяє використовувати більш складні нейромережеві структури та більш всебічне навчання. Як результат, базовий *GAN* може виконати більш точну оцінку щільності, тим самим зменшуючи кількість паролів, необхідних для досягнення певної кількості збігів.

Розроблений програмний засіб можливо використовувати в системах перевірки інформаційної безпеки, або як окремий програмний засіб, наприклад, для перевірки безпеки політики паролів в організації.