Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра _____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

«Дослідження методів змагальних атак на нейронні мережі»
(тема)

Виконала: студентка 2 курсу, групи ІПЗм-17-1___
спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)
Освітньо-наукової програми
Інженерія програмного забезпечення_____
(повна назва освітньої програми)

_____Сізон Ю.О._____
(прізвище, ініціали)
Керівник _____доц. каф. ПІ, к.т.н Каук В.І._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2019 р.

Харківський національний університет радіоелектроніки

Факультет_Комп'ютерних наук_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність _121-Інженерія програмного забезпечення_____
(код і повна назва)

освітньо-наукова програма _Інженерія програмного забезпечення_____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 2019 р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентці _Сізон Юлії Олегівні_____
(прізвище, ім'я, по батькові)

1. Тема роботи _«Дослідження методів змагальних атак на нейронні мережі» затверджена наказом по університету від "___" _____20 ___ р № 546СТ
заповнюється вручну після отримання наказу

2. Термін подання студентом роботи до екзаменаційної комісії
07 червня 2019 р.

3. Вихідні дані до роботи *алгоритми тренування нейронної мережі, алгоритми створення змагальних прикладів, пояснювальна записка. Використовувати середовище Jupyter Notebooks у Google Colaboratory.*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної основних проблем та стану їх розв'язання, огляд засобів захисту від атак, актуальність та поширення змагальних атак, оцінка наслідків вразливостей до змагальних атак для машинного навчання, огляд змагальних прикладів, огляд методів змагальних атак, визначення мети практичних досліджень, опис навчання нейронної мережі, опис атаки на нейронну мережу, опис розробки покращеного методу атаки на нейронну мережу, розгляд показників для покращення, вибір показника, опис реалізації покращеного методу змагальної атаки, аналіз результату атаки на мережу після покращення* _____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *приклади змагальних зображень, схеми архітектур нейронних мереж, графічне зображення функцій, код методів та алгоритмів, графи функції витрат, розроблені змагальні приклади, шум (пертурбації), результати аналізу успішності покращеної змагальної атаки, демонстраційні матеріали.*

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доц. к.т.н. Каук В.І.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Аналіз предметної галузі	10 березня 2019р.	
2.	Огляд існуючих методів	11 березня 2019р.	
3.	Навчання нейронної мережі	20 березня 2019р.	
4.	Проведення атаки на нейронну мережу	30 березня 2019р.	
5	Розробка покращеного методу атаки на нейронну мережу	15 квітня 2019р.	
6.	Підготовка пояснювальної записки	2 травня 2019р.	
7.	Спецчастина	15 травня 2019р.	
8.	Підготовка презентації та доповіді	18 травня 2019р.	
9.	Попередній захист	27 травня 2019р.	
10.	Нормоконтроль, рецензування	4 червня 2019р.	
11.	Занесення диплома в електронний архів	4 червня 2019р.	
12.	Допуск до захисту у зав. кафедри	5 червня 2019р.	
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання _____ _____2019 р.

Студент _____
(підпис)

Керівник роботи _____ __доц. каф. ПІ, к.т.н Каук В.І.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 80 ст., 45 рис., 33 джерела.

Об’єктом дослідження є методи змагальних атак на нейронні мережі.

Метою роботи є знаходження покращеного методу змагальної атаки на нейронну мережу.

Методи дослідження базуються на технологіях Python 3, Keras, Tensorflow, skimage, Flask.

У результаті роботи проведено дослідження методів змагальних атак на нейронні мережі, знайдено покращений методи атаки.

ЗМАГАЛЬНІ АТАКИ, ЗМАГАЛЬНІ ПРИКЛАДИ, НЕЙРОННІ МЕРЕЖІ, PYTHON 3, TENSORFLOW, KERAS.

The object of the research are methods of adversarial attacks on neural networks.

The aim is to introduce an improved version of an adversarial attack method.

Methods of the research are based on Python 3, Keras, Tensorflow, skimage, Flask technologies.

As a result of the work, research on the methods of competitive attacks on neural networks was carried out and an improved method of adversarial attack was found.

ADVERSARIAL ATTACKS, ADVERSARIAL EXAMPLES, NEURAL NETWORKS, PYTHON 3, TENSORFLOW, KERAS.

ПЕРЕЛІК СКОРОЧЕНЬ

Глибоке навчання	Сукупність методів машинного навчання, заснованих на навчанні уявленням, а не спеціалізованих алгоритмах під конкретні завдання.
Датасет	Набір окремих наборів інформації, що обробляються комп'ютером як єдиний блок.
Згорткова нейронна мережа	Алгоритм глибокого навчання, який може приймати вхідне зображення, присвоювати важливість (ваги, що отримуються навчанням) різним аспектам чи об'єктам зображення і може відрізняти одне зображення від іншого.
Змагальна атака	Техніка, спрямована на те, щоб обманути модель через подання зловмисних даних.
Змагальний приклад	Дані, що подаються до нейронної мережі, які призводять до помилки у роботі мережі.
Нейронна мережа	Обчислювальна система, натхненна біологічними нейронними мережами, що навчається виконувати завдання на прикладах.
ШІ	Штучний інтелект.
FGSM	Fast Gradient Sign Method.
ImageNet	Організована база даних зображень.

InceptionV3	Модель Keras, з вагами, попередньо навченими на ImageNet.
InceptionResNetV2	Модель Keras, з вагами, попередньо навченими на ImageNet.
InceptionV3	Модель Keras, з вагами, попередньо навченими на ImageNet.
Keras	Відкрита нейромережева бібліотека.
Pointwise convolution	Згортка 1x1, «точкова» чи «поточкова».
ReLU	«Випрямляч», є функцією активації.
ResNet50	Модель Keras, з вагами, попередньо навченими на ImageNet.
TensorFlow	Бібліотека з відкритим кодом для потоків даних і диференційованого програмування.
Xception	Модель Keras, з вагами, попередньо навченими на ImageNet.

ЗМІСТ

ВСТУП

Глибоке навчання (сукупність методів машинного навчання, заснованих на навчанні уявленням) стрімко розвивається і показує чудові результати в широкому спектрі завдань. Методи глибокого навчання використовується в багатьох додатках, де важлива безпека. Однак, незважаючи на чудові результати, що найчастіше перевершують людські можливості, методи глибокого навчання уразливі для спеціальних вхідних даних. Дані такого виду називаються змагальними (англ. adversarial) зразками. Розробники змагаються в проектуванні змагальних атак (технік, спрямованих на те, щоб обманути модель через подання зловмисних даних) з одного боку і створенні механізмів захисту з іншого [1].

Завдання змагальної атаки взаємопов'язані із завданнями комп’ютерного зору, такими як розпізнавання об'єктів, класифікація. Для обробки зображень за допомогою глибокого навчання невеликі відхилення у вхідних даних можуть вести до значних змін на виході. Такі зміни майже непомітні для людини і не змінюють семантику зображення, але вони здатні обманути методи глибокого навчання.

Змагальні атаки є важливим інструментом при роботі з вимогливими до безпеки додатками, такими як розпізнавання особистості, контроль доступу і тому подібними. Алгоритм приймає важливі рішення, тому необхідно бути впевненими, що його нельзя обманути.

Таким чином, метою роботи є дослідження методів змагальних атак та вразливостей, які вони використовують у нейронних мережах (обчислювальні системи, натхненні біологічними нейронними мережами, що навчаються виконувати завдання на прикладах), а також покращення одного з методів змагальних атак із метою виявлення критичних вразливостей нейронних мереж.

Мета роботи є актуальною, бо із кожним роком нейронні мережі впроваджуються у все більше сфер нашого життя, і при тому критично важливими з точки зору безпеки людини, такі як сфера охорони здоров’я,

військова справа, біометрія, автоматично керовані автомобілі та дрони, тощо [2]. Саме тому важливо досліджувати тему змагальних атак і вдосконалювати якість змагальних прикладів – це змушує нейронні мережі ставати стійкішими до похибок даних та зловмисників, запобігаючи зловживанню та знижуючи потенційну небезпеку від їх вживання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

У 2014 році публікація дослідження, проведеного дослідницькою групою штучного інтелекту (ШІ), проведеної під керівництвом Google, відкрило новий підхід до злому (англ. “hacking”), яке називається змагальною атакою.

Методи, продемонстровані в дослідженні, не тільки змінили наше розуміння того, як працює машинне навчання, але й на практиці показали, потенційно наскільки сильно може бути підірвано один з найбільш науково, інженерно та комерційно перспективних і довгоочікуваних аспектів науково-технічної революції [3].

Цей новий напрямок атаки виявився настільки притаманним базовій структурі глибоких нейронних мереж, що до сьогоднішнього дня найкращі уми із багатьох галузей, пов’язаних із нейронними мережами, таких як розпізнавання зображень і відео, розробка рекомендаційних систем, класифікація зображень, аналіз медичних зображень, обробка природної мови, дослідження щодо інтелектуальної власності, тощо відчувають труднощі у розробці ефективних засобів захисту від неї.

Це не було проблемою рівня Y2 (клас комп'ютерних помилок, пов'язаних з форматуванням і зберіганням даних календаря для дат, починаючи з 2000 року), тобто проблемою нагляду за програмою, а радше системною архітектурною вразливістю, яка загрожує перенести теми змагальних атак з поточного періоду академічного і теоретичного дослідження і розвитку на нейронні мережі в бізнес, військові, медичні, політичні, автоматизаційні, громадські та інші системи майбутнього.

Якщо машинне навчання має фундаментальну слабкість, еквівалентну сценарію «атаки 51%» [4] в схемах криптовалют, тоді його сприйнятливість до змагальних атак виглядатиме так само серйозно, якщо не більше, бо буде ставити під загрозу безпеку людини.

1.1 Аналіз основних проблем та стану їх розв'язання

Основною перевагою нейронних мереж нового покоління є здатність працювати з візуальним світом. Програмне забезпечення для аналізу зображень на основі ШІ може керувати транспортними засобами, аналізувати медичні зображення, розпізнавати обличчя, проводити перевірки безпеки, розширювати можливості роботів, класифікувати бази даних зображень, створювати простір у розширеній реальності, аналізувати та інтерпретувати відеоматеріали для подій та мови, і навіть допомагати в хірургічних процедурах [5].

Для цього система машинного навчання на основі зображень потребує двох ресурсів. Перший – це навчальний набір нерухомих зображень або відео, що дозволяє системі отримати належне точне розуміння об'єктів і подій, які вона може пізніше розпізнати. Другий – це зображення дії, наприклад, відеоматеріали CCTV.

З економічної точки зору, така система повинна бути розроблена для роботи з наявними матеріалами. Навіть якщо вхідні дані мають такі проблеми, як артефакти стиснення або низька роздільна здатність, ШІ повинен враховувати ці обмеження у своєму процесі.

Такий високий рівень відмовостійкості дозволяє нападникам отруїти вхідні дані і вплинути або навіть контролювати результат аналізу ШІ.

Термін «змагальна атака» вперше був введений у документі, опублікованому дослідником ШІ з Google, Крістіаном Сегеді, в 2014 році. Тонко змінюючи певні частини чи окремі пікселі зображення з бази даних ImageNet, дослідники змусили систему машинного навчання, розроблену на прикладі популярної нейронної мережі AlexNet, неправильно класифікувати зображення, як страусів замість інших випадкових об’єктів із набору даних [6]. Приклад зображень можна побачити на рисунку 1.1 У лівому стовпці показано вихідні зображення ImageNet. Середній стовпчик показує тонку матрицю відмінностей, застосовану дослідниками. У правій колонці показано кінцеві оброблені зображення, які всі були класифіковані як «страуси».

Процедура не просто легко відтворювалася, але, як не дивно, вона виявилася широко переносимою в різних моделях і конфігураціях нейронних мереж.

Атака – це неправильне використання фундаментальної функції концептуальної моделі систем машинного навчання на основі зображень, що використовує процес ліквідації, який система використовує при оцінці, яку позначку, тобто клас даних, слід застосувати до зображення.

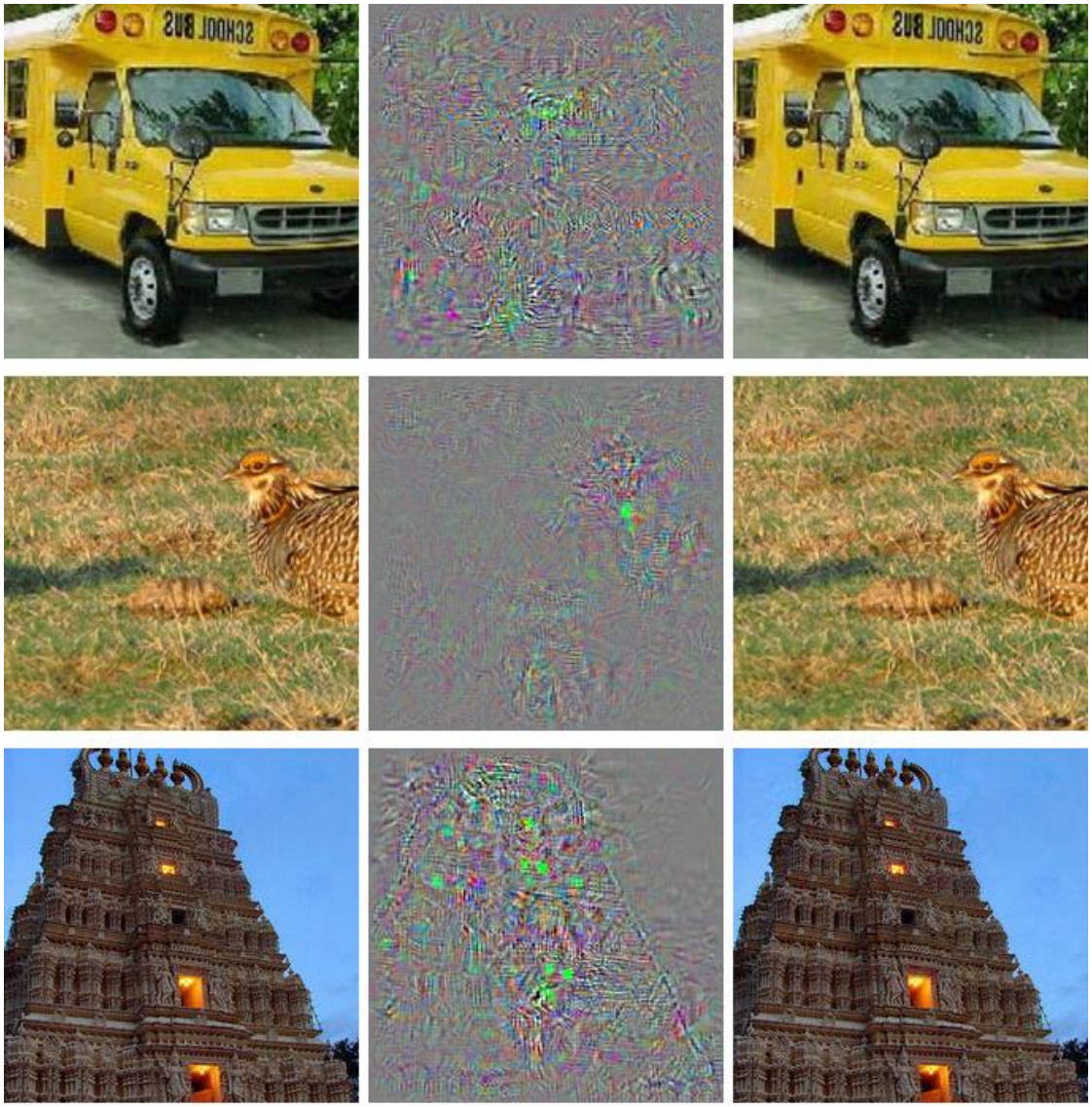

Рисунок 1.1 – Приклад атакованого зображення, матриці відмінностей та результуючого зображення

Характеристики можуть бути витягнуті з зображення, яке, швидше за все, класифікується як, наприклад, страус, а потім непомітно застосовується до зображень, що не мають страусів, так що вони стануть класифіковані як

страуси. Математика, що керує процедурою ліквідації, дозволяє зловмиснику систематично вносити трансформоване зображення в цільову класифікацію.

Рівень візуального «ускладнення» в таких змагальних зображеннях настільки низький, що і людське око, і нейронна мережа навчаються сприймати їх як шум. Але нейронна мережа не може цього зробити, оскільки цей шум формується як дистильований «хеш» для встановленої (хоча і неправильної) класифікації. Тому машина різко закриває рутину аналізу і повертає невірний результат, а редукціоністська математика нейронної мережі обертається проти себе у вигляді атаки.

Відкриття Сегеді виявилося вдвічі вражаючим, оскільки виявило лінійну природу вищого мірного простору глибоких мережних моделей навчання. До цього моменту вважалося, що нейронні мережі володіють малою лінійністю на цьому рівні експлуатації - ексцентричною особливістю, яка, за іронією долі, повинна захищати їх від таких систематизованих атак, які фактично сприяють суперечливому характеру «чорної скриньки» систем штучного інтелекту [7].

Натомість атака була не тільки тривіальною для відтворення, але й швидко виявилася, що вона має «реальний світ».

У 2016 році дослідження Carnegie Mellon побудували на основі висновків Szegedy, щоб створити метод, здатний обдурити сучасну систему розпізнавання обличчя за допомогою 2D-друкованих окулярів [8]. Приклад можна побачити на рисунку 1.2.

Перша колонка: два дослідники ухиляються від виявлення системи розпізнавання обличчя. Друга колонка: дослідник успішно визначив як ціль, актриса Мілла Йовович. Третя колонка: дослідник успішно ідентифікований як колега. Четверта колонка: дослідник успішно видає себе за телеведучого Карсона Дейлі.

На цьому етапі безліч нових дослідницьких робіт використовували теорію змагальних атак і почали розрізняти неправильну класифікацію (ухилення від правильної ідентифікації) і цільову класифікацію (прищеплення певного і невірного ідентифікатора цілі на зображення).

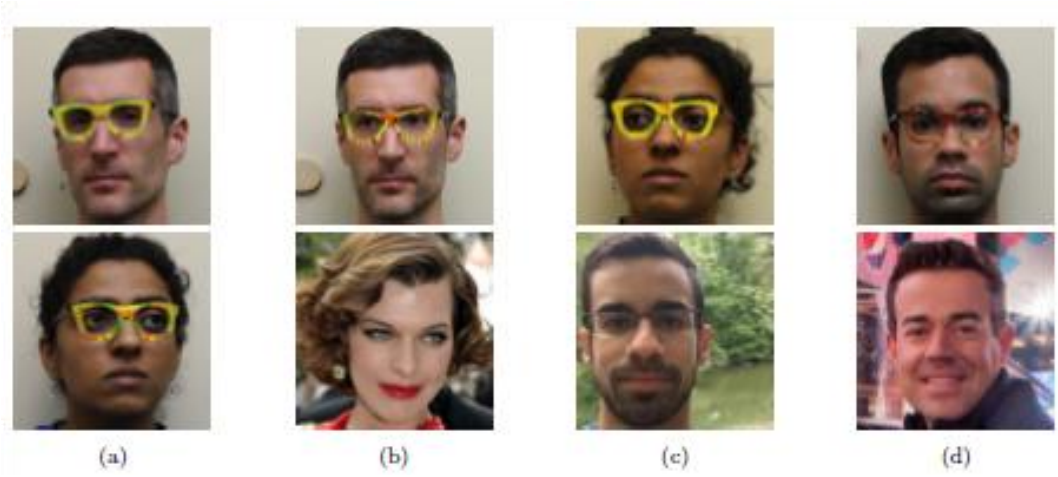

Рисунок 1.2 – Приклад надурення системи розпізнавання обличчя за допомогою окулярів

У цьому випадку, дослідники змогли переконати систему розпізнавання облич і об'єктів, що одним з дослідників-чоловіків була голлівудська актриса і модель Мілла Йовович, а іншим – телеведучий Карсон Дейлі. Крім того, вони змогли поміняти свої ідентичності між собою або іншим чином змусити систему розпізнавання обличчя не виявити їх.

У всіх випадках ключем до ухилення, обману, була кодована текстура, надрукована на їхніх великих окулярах, що було достатнім для «хешування» цілі для системи машинного навчання.

Будь-які сумніви щодо стійкості цільових текстур були усунені в 2017 році, коли дослідники MIT 3D-друкували спеціально створену модель черепах, здатну переконати систему класифікації зображень InceptionV3 від Google, що це гвинтівка.

1.2 Огляд засобів захисту від атак

З моменту існування змагальних атак було розглянуто широкий спектр контрзаходів. Дослідники з Гарвардського університету в 2017 році

запропонували регулювати вхідні дані для того, щоб вирівняти умови і полегшити ідентифікування створених зображень атаки [9].

Проте, ця методика вимагає подвоєння вже напружених ресурсів, необхідних для роботи життєздатної системи машинного навчання. Навіть там, де можуть бути здійснені варіації на цій техніці, «градієнтне маскування», можливі контратаки [10].

У дослідженні Стенфордської науково-дослідної роботи 2017 року запропоновано «біологічно натхненний» [11] глибокий захист мережі, який штучно створює нелінійність, яку дослідники мали, як вважалося, до 2014 року. Тим не менш, критики свідчать про те, що методика легко заперечується стабілізацією входу градієнта [12].

Нещодавній звіт [13] Institute of Electrical and Electronics Engineers (IEEE) оцінює чотирирічні дослідження, орієнтовані на захист, після розкриття техніки змагальної атаки і не знаходить запропонованого підходу, який може остаточно і економічно подолати системний характер уразливості.

У доповіді робиться висновок, що змагальні атаки створюють «реальну загрозу для глибокого навчання на практиці, особливо у важливих для безпеки та безпеки випадках».

Дослідники Gragnaniello, Marra, Poggi та Verdoliva у роботі “Analysis of adversarial attacks against CNN-based image forgery detectors” [14] погоджуються з тим, що сучасні контрзаходи не страхують від сценарію змагальної атаки.

1.3 Актуальність та поширення змагальних атак

Змагальні атаки набувають все ширшого поширення. Найбільш тривожним аспектом змагальної атаки є її здатність переноситись не тільки між різними системами машинного навчання і навчальними наборами, але й у інші області машинного навчання, крім комп'ютерного зору.

Розглянемо, яким чином змагальні атаки проникли у інші галузі обробки даних, окрім розпізнавання зображень, стан розвитку проблеми та який ризик змагальні атаки несуть для цих галузей.

1.3.1 Змагальні атаки в аудіо

В 2018 році дослідники з Університету Каліфорнії Carlini та Wagner у своїй роботі “Audio Adversarial Examples: Targeted Attacks on Speech-to-Text” [15] продемонстрували, що до звукової хвилі (наприклад, запису мови) можна додати змагальне ускладнення і повністю змінити транскрипцію мовлення в текст до цільової фрази – або навіть приховати мовну інформацію в інших типах аудіо, наприклад, музику.

Це сценарій нападу, який, серед інших можливостей, загрожує цілісності асистентів ШІ, тобто домашніх асистентів, які через голосові команди керують температурою в будинку, технікою, виконують покупки, тощо.

Комерційна експлуатація пристроїв постійного прослуховування в домашніх умовах вже зазнала жорстокої критики, а обман розмовного входу користувачів чітко впливає на інвазивні маркетингові кампанії.

1.3.2 Змагальні атаки в тексті

Науково-дослідницька робота компанії India Research Lab за 2017 рік [16] описує сценарій змагальної атаки, заснований на чистому текстовому введенні, і зауважує, що така техніка може бути використана для маніпулювання та обманювання систем аналізу настроїв, які використовують методи обробки природних мов (НЛП).

На широкому рівні така техніка може потенційно зробити можливою кампанію з аналітичної дезінформації в категорії «фальшивих новин».

1.4 Оцінка наслідків вразливостей до змагальних атак для машинного навчання

Іан Гудфелло, творець генеративних змагальних мереж [17] і ключовий учасник дебатів про змагальні атаки, прокоментував [18], що системи машинного навчання не повинні повторювати помилку ранніх операційних систем, де захист спочатку не був вбудований в дизайн.

Проте звіт IEEE, підкріплений іншими дослідженнями, свідчить про те, що глибокі навчальні мережі мали цю слабкість з початку існування. Всі спроби протистояти синдрому за його власних умов можуть бути відбиті, оскільки вони є ітераціями (і контр-ітераціями) тієї ж математики.

Залишається можливість «брандмауера» або дозорного підходу до очищення даних, де супутникові технології розвиваються навколо безперервно вразливої системи машинного навчання, ціною мережевих і машинних ресурсів, і зменшується цілісність всього процесу.

Таким чином, схоже, що комерційні проблеми мають пріоритет. До теперішнього часу єдиним реальним прикладом змагальної атаки було враження потенційної важкості проблеми для ЗМІ, коли група дослідників змусила систему машинного навчання неправильно витлумачити дорожні знаки, просто надрукувавши їх тонко змінені версії.

Тим не менш, у доповіді IEEE відмічається зневажливе ставлення до загрози в наукових колах: «Незважаючи на те, що кілька робіт свідчать про те, що змагальні атаки на глибоке навчання не можуть бути серйозною проблемою, у великій кількості відповідної літератури вказано інше.» Сучасна придатність або обсяг змагальних атак не має значення, якщо фундаментальна вразливість виживає в пізніших виробничих системах. У той момент комерціалізації та поширеного розповсюдження, такі системи, ймовірно, представляють набагато більш привабливу ціль, заслуговуючи більших ресурсів від тих, хто будуть використовувати одні з нових найбільш чутливих і критичних керованих штучним інтелектом додатків майбутнього.

2 ОПИС ПРОВЕДЕНИХ ТЕОРЕТИЧНИХ ДОСЛІДЖЕНЬ

2.1 Огляд змагальних прикладів

Змагальним прикладом є зразок з малими, навмисними ускладненням, зашумленням ознак, які змушують модель машинного навчання робити помилкове передбачення. Змагальні приклади є даними, що мають за мету обманути модель, а не інтерпретувати її. Розглянемо, чим цікаві приклади змагальності, щоб переконатися, що вони не просто цікаві продукти машинного навчання без практичної актуальності. Змагальні приклади роблять моделі машинного навчання вразливими до атак, як продемонстровано нижче:

– автомобіль, що керується без безпосередньої допомоги людини, стикається їх іншим автомобілем, тому що він ігнорує знак зупинки. Хтось розмістив картинку над знаком, що схожа на знак зупинки із невеликою кількістю бруду з точки зору людини, але він був розроблений таким чином, щоб виглядати як знак заборони паркування для програмного забезпечення автомобіля, що розпізнає знаки;

– детектор спаму не може класифікувати електронний лист як спам. Спам-лист було розроблено, таким що нагадує звичайний електронний лист, але з наміром обманути одержувача;

– сканер, що працює на основі машинного навчання, сканує валізи в аеропорту на предмет наявності зброї. Було розроблено такий ніж, що уникає виявлення, змусивши систему вважати, що це парасолька.

Розглянемо деякі способи створення прикладів змагальності.

2.2 Огляд методів змагальних атак

Існує багато прийомів для створення змагальних прикладів. Більшість підходів передбачає мінімізацію різниці між змагальним прикладом та зразком,

що підлягає модифікації через змінення передбачення на бажаний (змагальний) результат. Деякі методи вимагають доступу до градієнтів моделі, яка, звичайно, працює тільки з моделями на основі градієнта, такими як нейронні мережі, інші методи вимагають лише доступу до функції передбачення, що робить ці методи агностичними. Методи, що буде розглянуто зосереджуються на класифікаторах зображень з глибокими нейронними мережами, оскільки багато досліджень виконуються в цій галузі, а візуалізація змагальних зображень є дуже навчальною. Змагальні приклади для зображень – це зображення з навмисно зміненими пікселями з метою обманути модель під час застосування. Приклади наочно демонструють, наскільки легко глибокі нейронні мережі для розпізнавання об'єктів можуть бути обмануто зображеннями, які здаються адекватними для людини. Змагальні приклади подібні до оптичних ілюзій, але для комп’ютерів.

2.2.1 Оптимізаційний підхід на основі градієнта

C. Szegedy та інші автори науково-дослідної роботи “Intriguing Properties of Neural Networks” [19] використовують підхід оптимізації на основі градієнта в своїй роботі для пошуку змагальних прикладів для глибоких нейронних мереж.

Приклад із цієї роботи можно побачити на рисунку 2.1 нижче: всі зображення в лівому стовпці класифіковані правильно. У середньому стовпці відображається (збільшена) помилка, додана до зображень для створення зображень у правому стовпці, які всі класифіковані (неправильно) як «страус».

Ці змагальні приклади були сформовані шляхом мінімізації наступної функції щодо r:

$$loss(f(x + r), l) + c \cdot |r| \quad (2.1)$$

де x – зображення (представлене у вигляді вектора пікселів),

r – зміна пікселів для створення змагального зображення (x + r створює нове зображення),

l – бажаний клас результатів,

c – параметр, що використовується для балансування відстані між зображеннями і відстані між передбаченнями.

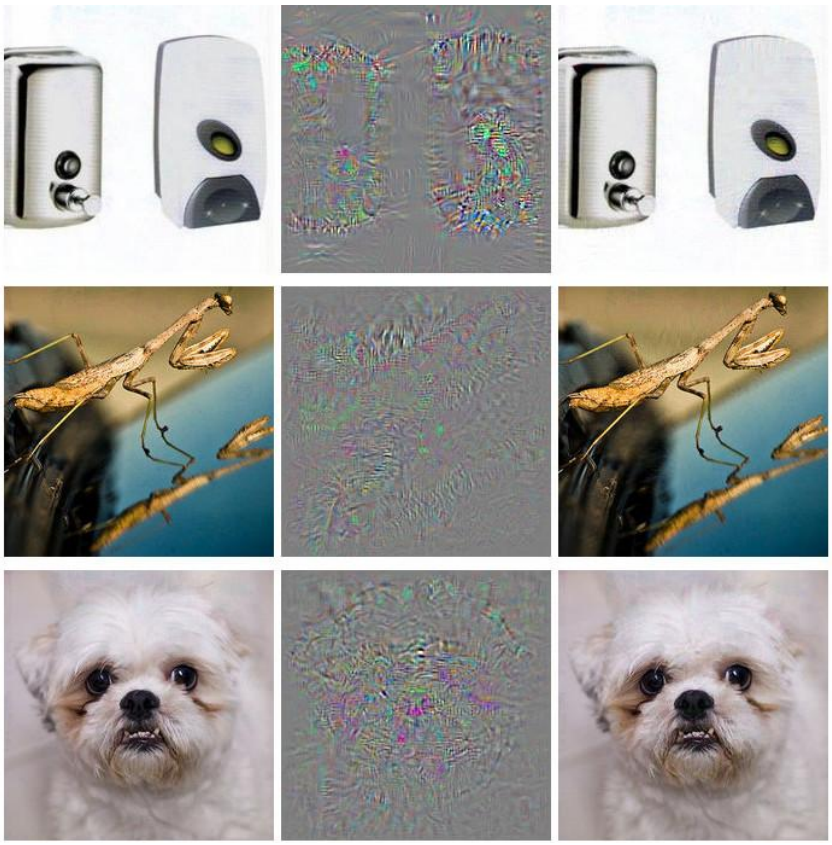

Рисунок 2.1 – Змагальні приклади для AlexNet із роботи науково-дослідної “Intriguing Properties of Neural Networks”

Перший член являє собою відстань між прогнозованим результатом прикладу змагальності і бажаним класом l, другий член вимірює відстань між змагальним прикладом і вихідним зображенням. Ця формула майже ідентична до функції втрат для формування контрфактичних пояснень. Існують додаткові обмеження для r такі, щоб значення пікселів залишалися між 0 і 1. Автори пропонують вирішити цю задачу оптимізації за допомогою блоку L-BFGS, тобто оптимізаційного блоку, який працює з градієнтами.

2.2.2 Fast Gradient Sign Method

Goodfellow та інші автори науково-дослідної роботи “Explaining and harnessing adversarial examples” [20] винайшли метод Fast Gradient Sign Method (FGSM) для генерації змагальних зображень. FGSM використовує градієнт базової моделі для пошуку змагальних прикладів. Оригінальне зображення x модифікується додаванням або відніманням невеликої помилки є до кожного пікселя. Чи буде додано або віднято є залежить від того, чи є знак градієнта для пікселя позитивним або негативним. Додавання помилок у напрямку градієнта означає, що зображення навмисно змінюється, так що класифікація моделі не вдається.

На рисунку 2.2 нижче можна побачити, як Goodfellow та інші автори науково-дослідної роботи роблять панду схожою на гібона для нейронної мережі. Додаючи невеликі деформації (середнє зображення) до оригінальних пікселів панди (ліве зображення), автори створюють змагальний приклад (дані, що подаються до нейронної мережі, які призводять до помилки у роботі мережі), який класифікується як гібон (праве зображення), але виглядає як панда для людей.

Наступна формула описує суть FGSM:

$$x' = x + \epsilon \cdot \ sign(\nabla_x J(\theta, x, y)) \quad (2.2)$$

де $\nabla_x J$ – градієнт функції втрати моделей щодо початкового вхідного піксельного вектора x,

y – вірний вектор міток для x,

θ – векторний параметр моделі.

З вектора градієнта (який є довжиною вектора вхідних пікселів) нам потрібен тільки знак: знак градієнта позитивний (+1), якщо збільшення

інтенсивності пікселів збільшує втрати (помилка моделі) і негативний (-1), якщо зменшення інтенсивності пікселів збільшує втрати.

Ця вразливість виникає, коли нейронна мережа обробляє зв'язок між інтенсивністю вхідних пікселів і лінійним показником класу.

Зокрема, архітектури нейронних мереж, які сприяють лінійності, такі як LSTM (архітектура штучної рекурентної нейронної мережі (RNN), що використовується в області глибокого навчання), мережі максимуму, мережі з блоками активації ReLU («випрямляч», функція активації) або інші лінійні алгоритми машинного навчання, такі як логістична регресія, уразливі до FGSM.

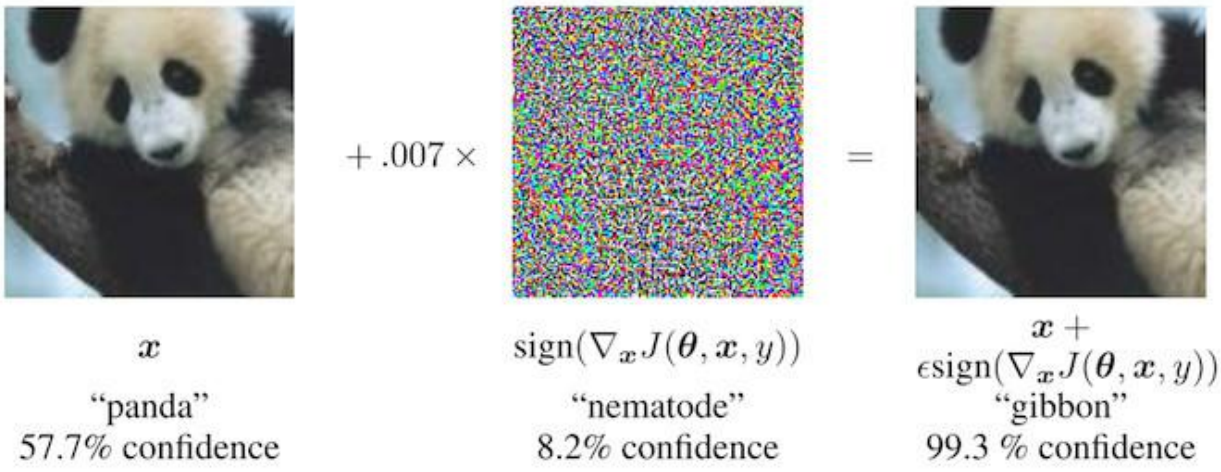

Рисунок 2.2 – Goodfellow та інші автори науково-дослідної роботи "Explaining and harnessing adversarial examples" роблять панду схожою на гібона для нейронної мережі

Атака здійснюється екстраполяцією. Лінійність між інтенсивністю вхідних пікселів і оцінками класів призводить до вразливості до викидів, тобто модель може бути обманута шляхом переміщення значень пікселів в області поза розподілом даних. Можна очікувати, що ці змагальні приклади будуть досить специфічними для даної архітектури нейронної мережі. Але насправді, можна повторно використовувати приклади змагань, щоб обманювати мережі з іншою архітектурою, підготовленою на одне і те ж завдання. Goodfellow та інші автори науково-дослідної роботи запропонували додавати змагальні приклади до навчальних даних для навчання надійних моделей.

2.2.3 Метод 1-піксельної атаки

Підхід, запропонований Goodfellow та його колегами, вимагає зміни багатьох пікселів, хоча б трохи. Підійдемо до завдання через змінення лише одного пікселю та подивимося, чи можливо таким чином обдурити модель машинного навчання. Su та інші автори науково-дослідної роботи “One pixel attack for fooling deep neural networks” показали [21], що насправді можна обманювати класифікатори зображень шляхом зміни одного пікселя.

На рисунку 2.3, що наведено нижче, можна побачити приклад із науково-дослідної роботи Su та інших авторів, де навмисно змінюючи один піксель (позначений колами), нейронну мережу, що навчається на ImageNet, обмануто, і вона передбачає неправильний клас замість оригінального.

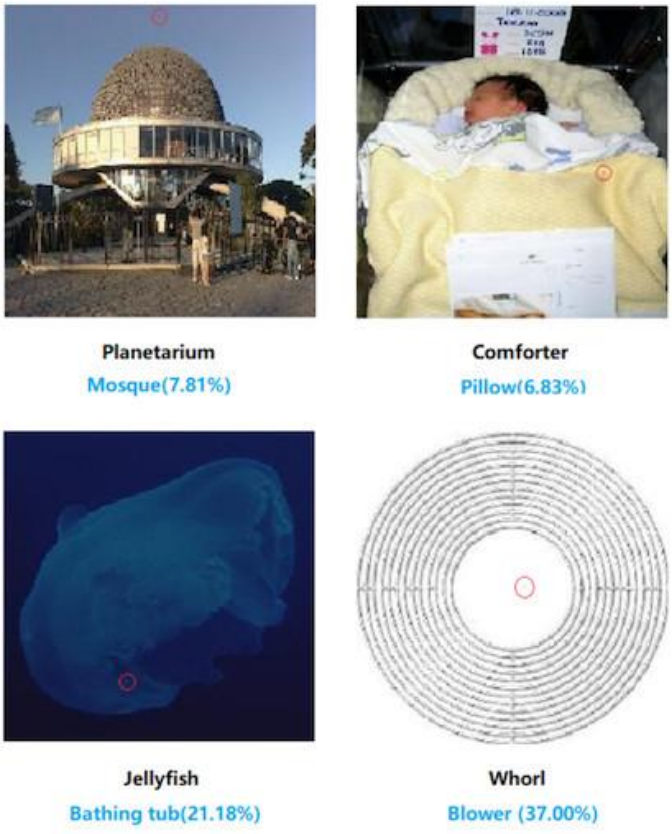

Рисунок 2.3 – Приклад із роботи “One pixel attack for fooling deep neural networks” від Su та інших авторів

Подібно до контрфактики (гіпотези), 1-піксельна атака шукає модифікований приклад *x*, який наближається до вихідного зображення *x*, але змінює прогноз на змагальний результат. Однак визначення близькості відрізняється: може змінюватися лише один піксель. 1-піксельна атака використовує диференціальну еволюцію (метод, який оптимізує проблему шляхом ітеративної спроби поліпшити рішення з урахуванням певного показника якості), щоб з'ясувати, який піксель потрібно змінити і як.

Ідея диференціальної еволюції взята із біологічної еволюції видів, далі наведено пояснення. Популяція індивідуумів, які називаються рішеннями кандидатів (англ. candidate solutions), рекомбінує покоління за поколінням до виявлення рішення. Кожне рішення-кандидат кодує піксельну модифікацію і представляється вектором з п'яти елементів: *x*- і *y*-координат та червоного, зеленого і синього (RGB) значень. Пошук починається з, наприклад, 400 рішень-кандидатів (тобто пропозицій щодо модифікації пікселів) і створює нове покоління рішень-кандидатів (дочірніх елементів) з початкового покоління з використанням наступної формули:

$$x_i(g + 1) = x_{r1}(g) + F \cdot (x_{r2}(g) + x_{r3}(g)) \quad (2.3)$$

де *xi* – елемент рішення кандидата (або *x*-координата, *y*-координата, червоний, зелений або синій),

g – поточне покоління,

F – параметр масштабування (встановлений в 0,5),

r1, *r2* і *r3* – різні випадкові числа.

Кожен новий дочірній варіант-кандидат, в свою чергу, є пікселем з п'ятьма атрибутами розташування і кольору, і кожен з цих атрибутів є сумішшю трьох випадкових батьківських пікселів.

Створення дочірніх варіантів зупиняється, якщо один з рішень-кандидатів є змагальним прикладом, тобто він класифікується як неправильний клас, або якщо досягнута кількість максимальних ітерацій, вказаних користувачем.

2.2.4 Метод змагального патча

Brown та інші автори науково-дослідницької роботи “Adversarial patch” [22] розробили етикетку для друку, яка може бути закріплена біля об'єктів, щоб вони виглядали як тостери для класифікатора зображень.

На рисунку 2.4, нижче, можна побачити наклейку, яка змушує класифікатор VGG16 (модель згорткової нейронної мережі, тобто алгоритму глибокого навчання, який може приймати вхідне зображення, присвоювати важливість (ваги (англ. weights), що отримуються навчанням) різним аспектам чи об'єктам зображення і може відрізняти одне зображення від іншого) навчений на ImageNet, класифікувати зображення банана як тостера.

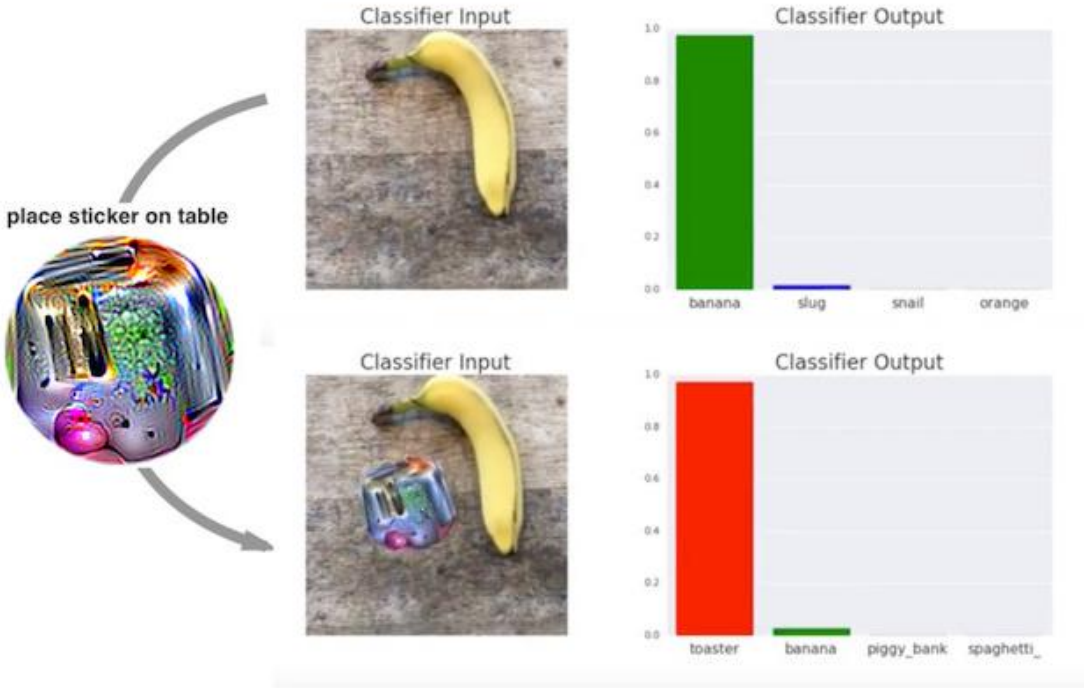

Рисунок 2.4 – Приклад із роботи “Adversarial patch” від Brown та інших авторів

Цей метод відрізняється від методів, представлених вище, оскільки у цьому випадку немає обмеження, зв’язаного з тим, що змагальне зображення має бути дуже близьким до вихідного зображення. Замість цього метод

повністю замінює частину зображення патчем, який може мати будь-яку форму.

Зображення патча оптимізовано під різні фонові зображення, з різними положеннями патча на зображеннях, іноді він переміщуються, іноді він є більшим або меншим чи є поверненим, таким чином патч працює в багатьох ситуаціях.

В результаті це оптимізоване зображення може бути надруковано і використано для обману класифікаторів зображень у житті.

2.2.5 Метод стійких (робастних) змагальних прикладів

Athalye та інші автори науково-дослідницької роботи “Synthesizing robust adversarial examples” [23] розробили метод стійких (робастних) змагальних прикладів. Надрукована на 3D-принтері черепаха, розроблена для того, щоб виглядати як гвинтівка до глибокої нейронної мережі з майже всіх можливих кутів, тобто фізичний об'єкт, який виглядає як черепаха для людини, виглядає як гвинтівка до комп'ютера.

На рисунку 2.5, нижче, можна побачити приклад із роботи “Synthesizing robust adversarial examples” від Athalye та інших авторів, де надрукована на 3D-принтері черепаха визнана гвинтівкою попередньо

Автори знайшли спосіб створити змагальний приклад в 3D для 2D-класифікатора, який є змагальним для перетворень, наприклад, всі види повороту черепахи, збільшення масштабу і так далі.

Інші підходи, такі як FGSM, не працюють, коли зображення повертається або змінюється кут огляду. Athalye із колегами пропонують алгоритм очікуваного перетворення (англ. EOT), який є методом генерування змагальних прикладів, які працюють навіть, коли зображення трансформується.

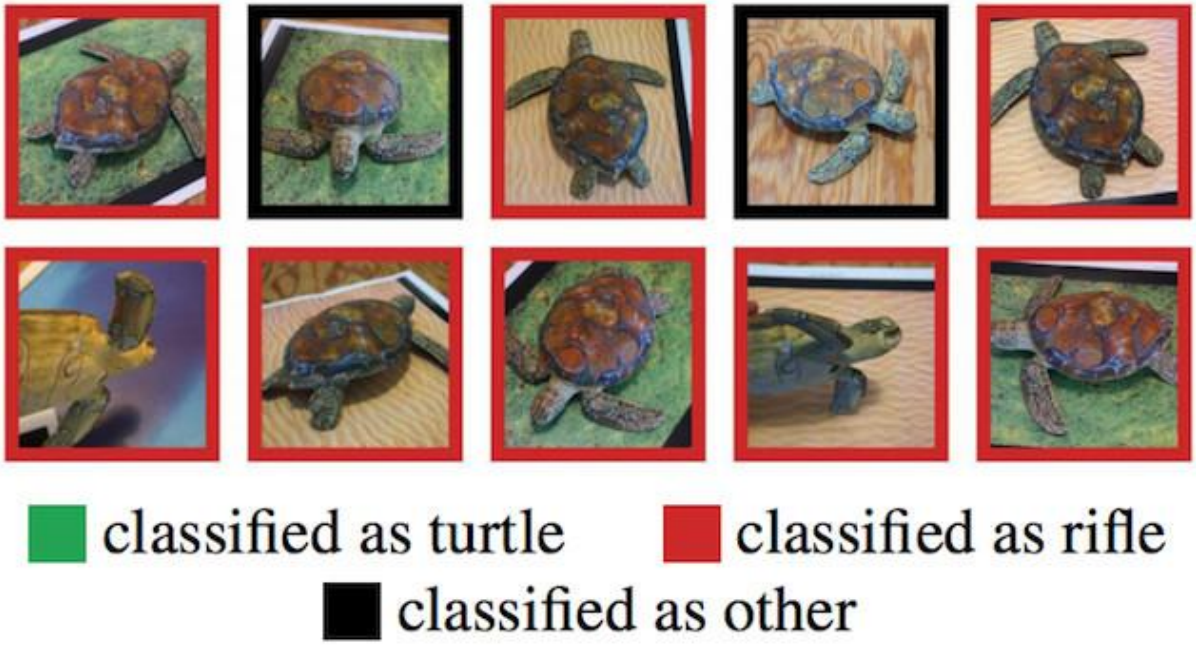

Рисунок 2.5 – Приклад із роботи “Synthesizing robust adversarial examples”

Основна ідея алгоритму очікуваного перетворення – оптимізувати змагальні приклади для багатьох можливих перетворень. Замість мінімізації відстані між змагальним прикладом і вихідним зображенням, алгоритм очікуваного перетворення зберігає очікувану відстань між двома нижче певного порогу, враховуючи вибраний розподіл можливих перетворень.

Очікувана відстань трансформації може бути записана як:

$$E_{t \sim T}[d(t(x'), t(x))] \quad (2.4)$$

де x – вихідне зображення,

$t(x)$ – перетворене зображення (наприклад, повернене),

x' – приклад змагання,

$t(x')$ – його перетворена версія.

Окрім роботи з розподілом перетворень, метод алгоритму очікуваного перетворення слідує знайомому шаблону обрамлення пошуку прикладів змагальності як завдання оптимізації. Ми намагаємося знайти змагальний

приклад x', який максимізує ймовірність для вибраного класу (наприклад, «гвинтівки») через розподіл можливих перетворень T:

$$arg\ max_{x'} E_{t\sim T}[logP(y_t|t(x'))] \quad (2.5)$$

З таким обмеженням, що очікувана відстань між усіма можливими перетвореннями між прикладом змагання x' і вихідним зображенням x залишається нижче певного порогу:

$$E_{t\sim T}[d(t(x'), t(x))] < \epsilon \quad \text{та} \quad \boldsymbol{x} \in [0, 1]^d \quad (2.6)$$

Можливості, які дає цей метод є дещо турбуючими. Інші методи засновані на маніпуляціях цифровими зображеннями, а ці стійкі змагальні приклади, надруковані на 3D-принтері можуть бути вставлені в будь-яку ситуацію у реальному житті та обманути комп'ютер, щоб він помилково класифікував об'єкт. Тобто, це дає можливість, скажімо, створити рушницю, що зчитується машиною як черепаха.

2.2.6 Метод атаки чорної скриньки

Papernot та його колеги у науково-дослідній роботі “Practical black-box attacks against machine learning” [24] показали, що можна створювати змагальні приклади без внутрішньої інформаційної моделі та без доступу до навчальних даних. Цей тип (майже) нульової атаки називається атакою чорної скриньки.

Як це працює:

– почати з декількох зображень, які надходять з того ж домену, що й дані тренувань, наприклад, якщо класифікатор, що підлягає атаці, є цифровим

класифікатором, використовувати зображення цифр. Необхідно знати домен, але не доступ до навчальних даних;

– отримати прогнози для поточного набору зображень із чорної скрині;

– навчати сурогатну модель на поточному наборі зображень (наприклад, нейронну мережу);

– створити новий набір синтетичних зображень, використовуючи евристику, яка досліджує поточний набір зображень, в якому напрямку маніпулювати пікселями, щоб зробити висновок моделі більш дисперсійним;

– повторити кроки від другого до четвертого для попередньо визначеного числа епох;

– створити змагальні приклади для сурогатної моделі з використанням FGSM (або подібного);

– атакувати оригінальну модель за допомогою змагальних прикладів.

Метою сурогатної моделі є апроксимація меж рішення моделі чорної скрині, але не обов'язково досягнення такої ж точності.

Автори випробували цей підхід, атакуючи класифікатори зображень, які навчалися на різних хмарних сервісах машинного навчання. Ці служби навчають класифікатори зображень на завантажених користувачами зображеннях і ярликах.

Програмне забезпечення тренує модель автоматично, іноді з алгоритмом невідомим користувачеві і розгортає його. Потім класифікатор робить передбачення для завантажених зображень, але сама модель не може бути перевірена або завантажена.

Papernot та інші дослідники, у зазначеній вище науково-дослідній роботі зазначають, що вони змогли знайти змагальні приклади для різних провайдерів, при цьому до 84% прикладів змагань були неправильно класифіковані.

Метод навіть працює, якщо модель чорної скрині, що підлягає обману, не є нейронною мережею. Це включає в себе моделі машинного навчання без градієнтів, таких як дерева рішень.

3 ОПИС ПРОВЕДЕНИХ ПРАКТИЧНИХ ДОСЛІДЖЕНЬ

3.1 Визначення мети практичних досліджень

Для формування мети практичного дослідження було проведено аналіз предметної галузі та проведено теоретичні дослідження щодо змагальних атак та змагальних прикладів, а саме: розглянуто існуючі на момент проведення дослідження види змагальних атак. Таким чином, потенційно корисним та новим із наукової точки зору дослідженням було визначено покращення одного з існуючих методів змагальних атак на нейронні мережі. Для реалізації поставленої задачі необхідно було визначити ряд теоретичних та практичних заходів, які необхідно вжити, а саме: обрання архітектури нейронної мережі, її розгортання та навчання; обрання одного із методів атаки, його реалізація, застосування на навченій нейронній мережі та фіксування показників атаки (ступінь впевненості мережі при розпізнаванні, метрики відмінності зміненого зображення від оригіналу, швидкість роботи алгоритму, тощо); розгляд показників для покращення, обрання того, над яким буде проведено роботу, розгляд методів покращення обраного показника та вибір одного з них, реалізація покращеного алгоритму, аналіз результату покращеного алгоритму атаки та порівняння із результатами застосування оригінального алгоритму.

3.2 Навчання нейронної мережі

Для того, щоб вжити змагальну атаку на нейронній мережі, потрібно спочатку обрати архітектуру нейронної мережі, налаштувати, підготувати дані, на яких її буде натреновано, та навчити вже саме мережу, на яку буде проведено спочатку оригінальну, а потім та покращену атаки.

3.2.1 Вибір та розгляд обраної архітектури нейронної мережі

Одним із найбільш поширених, цікавих для дослідження та тих, що розвиваються найбільш швидко, видів мереж є мережі, які розпізнають та класифікують зображення, тому такий вид мережі було обрано для атаки змагальними прикладами у цій роботі.

Видом архітектури нейронної мережі, на яку було проведені атаки, було обрано згорткову нейронну мережу Xception.

В останні кілька років нейронні мережі прийшли в усі галузі машинного навчання, але найбільше враження вони безперечно справили в галузі комп'ютерного зору. В рамках змагань ImageNet було представлено безліч різних архітектур згорткових мереж, які було оцінено інженерно-науковою спільнотою і впроваджено у багатьох фреймворках і бібліотеках.

Щоб поліпшити якість розпізнавання своїх мереж, дослідники намагалися додавати до мережі більше шарів, проте з часом прийшло розуміння, що іноді обмеження продуктивності просто не дозволяє навчати і використовувати настільки глибокі мережі. Це стало мотивацією для використання depthwise separable convolutions (термін можна перевести як «глибоко розділяєма згортка», але використовується у англійському варіанті) і створення архітектури Xception.

Кожного разу, коли додається ще один шар у згорточну мережу, потрібно приймати стратегічне рішення про те, якими будуть його характеристики: який розмір ядра згортки використовувати: 3x3, 5x5 чи поставити max pooling (дискретизацію на основі вибірки).

У 2015 році було запропоновано архітектуру Inception, ідея якої полягала в наступному: замість того, щоб вибирати розмір ядра, потрібно взяти кілька варіантів відразу, використати їх всі одночасно і об’єднати результати. Однак це суттєво збільшує кількість операцій, які необхідно виконати для обчислення активацій одного шару, тому автори оригінальної статті пропонують наступне: перед кожним згортковим блоком робити згортку з розміром ядра 1x1,

знижуючи розмірність сигналу, що подається на вхід згортками з великими розмірами ядер. На рисунку 3.1, нижче, можна побачити отриману конструкцію, що становить повний модуль Inception.

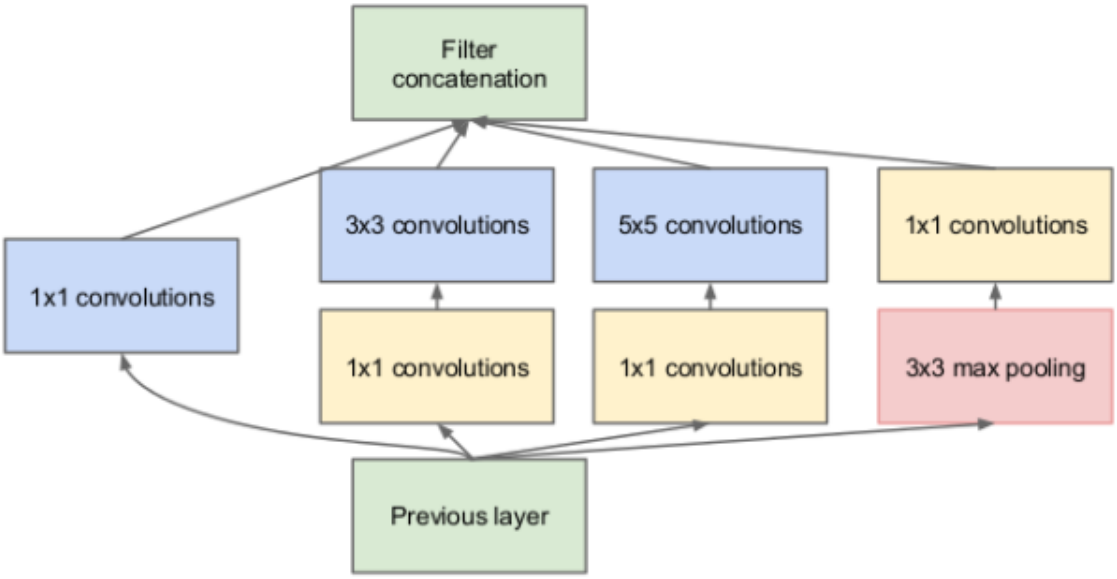

Рисунок 3.1 – Модуль Inception

У 2016 році Франсуа Шолль (François Chollet), автор і розробник Keras, відкритої нейромережевої бібліотеки, опублікував статтю, в якій запропонував піти далі і використовувати так званий екстремальний Inception-модуль, також відомий як depthwise separable convolution.

Візьмемо стандартний згортковий шар з фільтрами розміру 3x3, на вхід якого подається тензор розмірності

$$M * M * C_1 \quad (3.1)$$

де M – це ширина і висота тензора,

C_1 – кількість каналів.

Розглянемо, що робить такий шар. Він згортає одночасно всі канали вихідного сигналу різними згортками. На виході у такого шару виходить тензор розмірності $(M - 2) * (M - 2) * C_2$.

Замість цього можна зробити послідовно два кроки. Перший крок: згорнути вихідний тензор 1x1 згорткою, подібно до того як відбувається в блоці Inception, отримавши тензор $M * M * C_2$. Ця операція називається pointwise

convolution (можна перевести як «поточкова» чи «точечна» згортка, але цей термін використовується у англійському варіанті). Другий крок: згорнути кожен канал окремо 3x3 сверткой (при цьому розмірність не зміниться, так згортаються не всі канали разом, як в звичайному згорточному шарі). Ця операція називається depthwise spatial convolution (глибинна просторова згортка).

Схему pointwise convolution та depthwise spatial convolution можна побачити на рисунку 3.2, нижче.

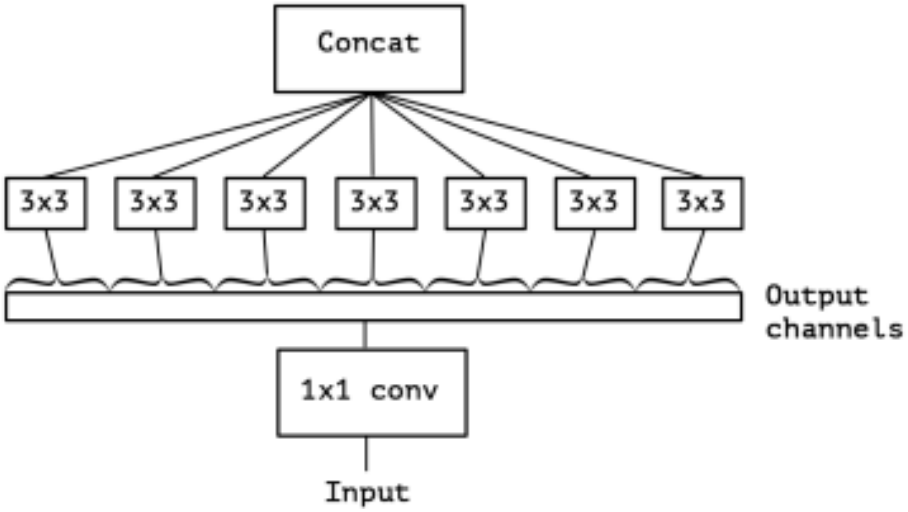

Рисунок 3.2 – Pointwise convolution та depthwise spatial convolution (глибинна просторова згортка)

Розглянемо конкретний приклад. Нехай згортається зображення з 16 каналами згортковим шаром з 32 фільтрами. Сумарно цей згортковий шар буде мати *16 * 32 * 3 * 3 = 4608* вагів, оскільки буде *16 * 32* згорток 3x3.

Визначимо, скільки вагів буде в аналогічному depthwise separable convolution блоці. По-перше, буде *16 * 32 * 1 * 1 = 512* вагів у pointwise convolution. По-друге, у нас буде *32 * 3 * 3 = 288* вагів у depthwise convolution. В сумі отримаємо 800 вагів, що набагато менше, ніж у звичайного згорткового шару.

Звичайний згортковий шар одночасно обробляє як просторову інформацію (кореляцію сусідніх точок всередині одного каналу), так і міжканальну інформацію, так як згортка застосовується до всіх каналів відразу. Архітектура Xception базується на припущенні про те, що ці два види

інформації можна обробляти послідовно без втрати якості роботи мережі, і розкладає звичайну згортку на pointwise convolution (яка обробляє тільки міжканального кореляцію) і spatial convolution (яка обробляє тільки просторову кореляцію в рамках окремого каналу).

Подивимося на реальний ефект. Для порівняння візьмемо дві по-справжньому глибоких архітектури згорточних мереж – ResNet50 і InceptionResNetV2. ResNet50 має 25 636 712 вагів, а переднавчена модель в Keras важить 99 Мб.

Точність, яка досягається цією моделлю на датасеті (наборі окремих наборів інформації, що обробляються комп'ютером як єдиний блок) ImageNet, становить 75,9%. InceptionResNetV2 має 55 873 736 навчаємих параметрів і важить 215 Мб, досягаючи точності 80.4%.

Подивимося, який результат матимемо з архітектурою Xception. Мережа має 22 910 480 вагів і важить 88 Мб. При цьому точність класифікації на ImageNet становить 79%.

Таким чином, ми отримуємо архітектуру мережі, яка перевершує за точністю ResNet50 і лише трохи поступається InceptionResNetV2, при цьому істотно виграючи за розмірами, а значить по необхідних ресурсів як для навчання, так і для використання цієї моделі.

3.2.2 Опис процесу підготовки та тренування мережі

Задача, що вирішує мережа, яку було натревано, – це розпізнавання зображень котів та собак. Для цього було взято датасет з ресурсу Kaggle, за посиланням https://www.kaggle.com/c/dogs-vs-cats, що містить придатний набір зображень котів та собак.

Спершу датасет було розбито на три частини: train (4000 зображень), validation (2000 зображень) і test (10000 зображень).

Так як Франсуа Шолль, автор архітектури Xception, за сумісництвом є творцем Keras, він надав ваги цієї мережі, навченої на ImageNet, в своєму фреймворку, які було використано для налаштування мережі під поставлену задачу за допомогою transfer learning (проблема дослідження в машинному навчанні, яка фокусується на зберіганні знань, отриманих при вирішенні однієї проблеми і застосуванні її до іншої, але пов'язаної проблеми).

Було завантажено ваги з ImageNet у Xception, звідки було прибрано останні повнозв'язні шари. Потім датасет було пропущено через цю мережу для того, щоб отримати ознаки, які згорткова мережа може визначити з зображень – так звані bottleneck features (карти активації з останнього згорткового шару),

На рисунку 3.3, нижче приведено код, який необхідно виконати для реалізації цього кроку.

Наступним кроком було створення повнозв'язної мережі і навчання її на ознаках, отриманих з згорткових шарів, із використанням спеціально відкладеної частини вихідного набору даних для валідації.

Код для виконання цього кроку можна побачити на рисунку 3.4, нижче.

Після цієї стадії, яка займає пару хвилин за умови використання відеокарти GeForce GTX 1060, було отримано accuracy (можно перевести як точність», але використовується у англійському варіанті) близько 99.4% на валідаційнійному датасеті.

Потім було продовжено навчання мережі з аугментацією вхідних даних (збіркою технік «поліпшення» тренувальних даних), із завантаженням до згорткових шарів вагів з ImageNet, а до повнозв'язних шарів – ваги, які мережа вивчила тільки що.

На рисунку 3.5, нижче, можна побачити реалізацію цього етапу. Після навчання мережі за описаною схемою, ще за п'ять епох було досягнуто точності в 99.5% на даних із валідаційного датасета.

Із перевіренням моделі на даних, з якими вона не взаємодіяла, і які не використовувалися для налаштування гіперпараметрів (параметрів попереднього розподілу), було отримано точність близько 96.9%.

3.3 Атака на нейронну мережу

Тепер, після того, як було навчено нейронну мережу за архітектурою Xception, можна розробити змагальний приклад та виконати змагальну атаку на навченій мережі.

```
inpt_tnsr = Input(shape=(img_height,img_width,3))

basic_model = xception.Xception(weights='imagenet',
                                include_top=False,
                                input_shape=(img_width,
                                             img_height,
                                             3),
                                pooling='avg')

generator_of_data = image.ImageDataGenerator(rescale=1. / 255)

training_generator = generator_of_data
                          .flow_from_directory(train_data_dir,
                                               target_size=(img_height,
                                                            img_width),
                                               batch_size=batch_size,
                                               class_mode=None,
                                               shuffle=False)

train_bottleneck_features = basic_model
                          .predict_generator(training_generator,
                                             // batch_size
                                             nb_train_samples)

np.save(open('train_bottleneck_features.npy', 'wb'),
        train_bottleneck_features)
```

Рисунок 3.3 – Отримання bottleneck features

Змагальним прикладом, що планується отримати, є зображення кота, до якого були приміненні такі непомітні оку трансформації, які змусять нейронну мережу класифікувати його, як зображення собаки. Тож перейдемо до генерації вхідних даних, що зможуть обманути мережу. Синтезування змагальних прикладів для нейронних мереж можна коротко описати так: невеликі, ретельно підібрані та застосовані на вхідних даних деформації, що призводять до того, що нейронні мережі помилково класифікують вхідні дані. Беручи до уваги, що змагальні приклади переносяться у фізичний світ і можуть бути надзвичайно дієвими та стійкими, це є справжньою проблемою безпеки.

```
this_model = Sequential()

this_model.add(Dense(256,  activation='relu',
                     input_shape=basic_model.output_shape[1:]))
this_model.add(Dropout(0.5))

this_model.add(Dense(128, activation='relu'))
this_model.add(Dropout(0.5))

this_model.add(Dense(1, activation='sigmoid'))
this_model.compile(optimizer=SGD(lr=0.01),
                   loss='binary_crossentropy',
                   metrics=['accuracy'])

checkpointer = ModelCheckpoint(filepath='top-weights.hdf5',
                               verbose=1,
                               save_best_only=True)

model_history = this_model.fit(train_data,
                               train_labels,
                               epochs=epochs,
                               batch_size=batch_size,
                               callbacks=[checkpointer],
                               validation_data=(validation_data,
                                                validation_labels))
```

Рисунок 3.4 – Навчання мережі на ознаках, отриманих з згорткових шарів

```
inpt_tensor = Input(shape=(img_height,img_width,3))

basic_model = xception.Xception(weights='imagenet',
                                include_top=False,
                                input_shape=(img_width, img_height, 3),
                                pooling='avg')

model_top = Sequential()

model_top.add(Dense(256, activation='relu',
                    input_shape=basic_model.output_shape[1:]))
model_top.add(Dropout(0.5))

model_top.add(Dense(128, activation='relu'))
model_top.add(Dropout(0.5))

model_top.add(Dense(1, activation='sigmoid'))

model_top.load_weights('top-weights.hdf5')

this_model = Model(inputs=basic_model.input,
                   outputs=top_model(basic_model.output))

this_model.compile(optimizer=SGD(lr=0.005,
                                 momentum=0.1,
                                 nesterov=True),
                   loss='binary_crossentropy',
                   metrics=['accuracy'])
```

Рисунок 3.5 – Навчання мережі з аугментацією вхідних даних

Далі наведено короткий вступ до алгоритмів синтезу змагальних прикладів і описано процес реалізації атак, що призводять до синтезування змагального прикладу, що слідує цій методиці.

3.3.1 Огляд методу атаки на мережу

Для розробки та застосування первісної атаки на нейронну мережу було обрано метод зворотного поширення помилки (backpropagation) [25] та метод проектованого градієнтного спуску (gradient descent) [26]. Розглянемо ці методи детальніше, щоб мати повне уявлення про те, як синтезувати змагальні приклади, що зможуть обманути нашу мережу.

Зворотне поширення, скорочене від «зворотного поширення помилки», є алгоритмом для навчання штучних нейронних мереж з використанням градієнтного спуску. Враховуючи архітектуру штучної нейронної мережі та функцію помилок, метод обчислює градієнт функції помилки щодо ваг нейронної мережі. Це узагальнення дельта-правила (метод навчання одношарових нейронних мереж для оновлення ваг для для вхідних даних за принципом градієнтного спуску) для персептронів (алгоритмів для керованого навчання бінарних класифікаторів) для багатошарових нейронних мереж.

Частина назви «зворотне» випливає з того факту, що обчислення градієнта йде зворотно через мережу, причому градієнт останнього шару ваг розраховується спочатку, і градієнт першого шару ваг розраховується останнім. Часткові обчислення градієнта з одного шару використовуються повторно в обчисленні градієнта для попереднього шару. Цей зворотний потік інформації про помилку дозволяє ефективно обчислювати градієнт на кожному шарі порівняно з наївним, тобто простим, підходом обчислення градієнта кожного шару окремо.

Зворотне розповсюдження пережило недавній сплеск популярності, пов’язаний із широким застосуванням глибоких нейронних мереж для

розпізнавання мови та розпізнавання зображень. Цей алгоритм вважається ефективним, і сучасні реалізації використовують переваги GPU (спеціалізованих графічних процесорів) для подальшого підвищення продуктивності.

У 1970-х роках було винайдено зворотне розповсюдження як загальний метод оптимізації для здійснення автоматичної диференціації складних вкладених (англ. nested) функцій. Проте лише 1986 року, після публікації статті Rumelhart, Hinton та Williams, під назвою “Learning Representations by Back-Propagating Errors” [27], важливість алгоритму була високо оцінена спільнотою машинного навчання.

Щоб зрозуміти математичне виведення алгоритму зворотного розповсюдження, спочатку сформуємо певне розуміння про співвідношення між фактичним виходом нейрона і правильним виходом для конкретного прикладу навчання. Розглянемо просту нейронну мережу з двома вхідними блоками, одним вихідним блоком і без прихованих блоків, і в якому кожен нейрон використовує лінійний вихід. Спочатку, перед тренуванням, ваги будуть встановлюватися випадковим чином. Тоді нейрон дізнається з навчальних прикладів, які в даному випадку складаються з набору кортежів (x_1, x_2, t) де x_1 і x_2 є входами в мережу і t є правильним виходом (який вихідна мережа повинна виробляти з урахуванням цих входів, коли навчання пройдено). Початкова мережа, задана як x_1 і x_2, обчислить висновок y, який, ймовірно, відрізняється від t (з урахуванням випадкових ваг). Функція втрати $L(t, y)$ використовується для вимірювання розбіжності між очікуваним виходом t і фактичним виходом y. Для задач регресійного аналізу квадратична помилка може бути використана як функція втрати, для класифікації може бути використана категорична перехресність. Схему описаної мережі можна побачити на рисунку 3.6, нижче. Як приклад розглянемо проблему регресії, використовуючи квадратичну помилку як втрату (loss):

$$L(t,y) = (t-y)^2 = E \quad (3.2)$$

де E — невідповідність або помилка.

Розглянемо мережу на одному тренувальному випадку: (1, 1, 0), таким чином, вхід x_1 і x_2 є 1 і 1 відповідно, а правильний вихід, t, дорівнює 0. Тепер, якщо фактичний вихід y нанесений на горизонтальну вісь проти помилки E на вертикальній осі, результатом буде парабола. Ілюстрацію можна побачити на рисунку 3.7, нижче.

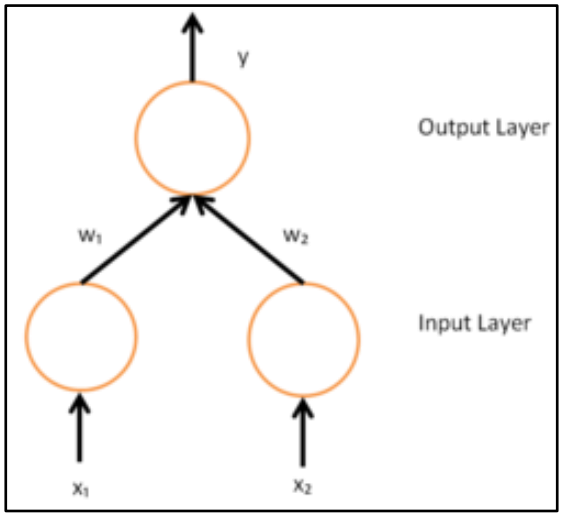

Рисунок 3.6 — Проста нейронна мережа з двома вхідними блоками і одним вихідним блоком

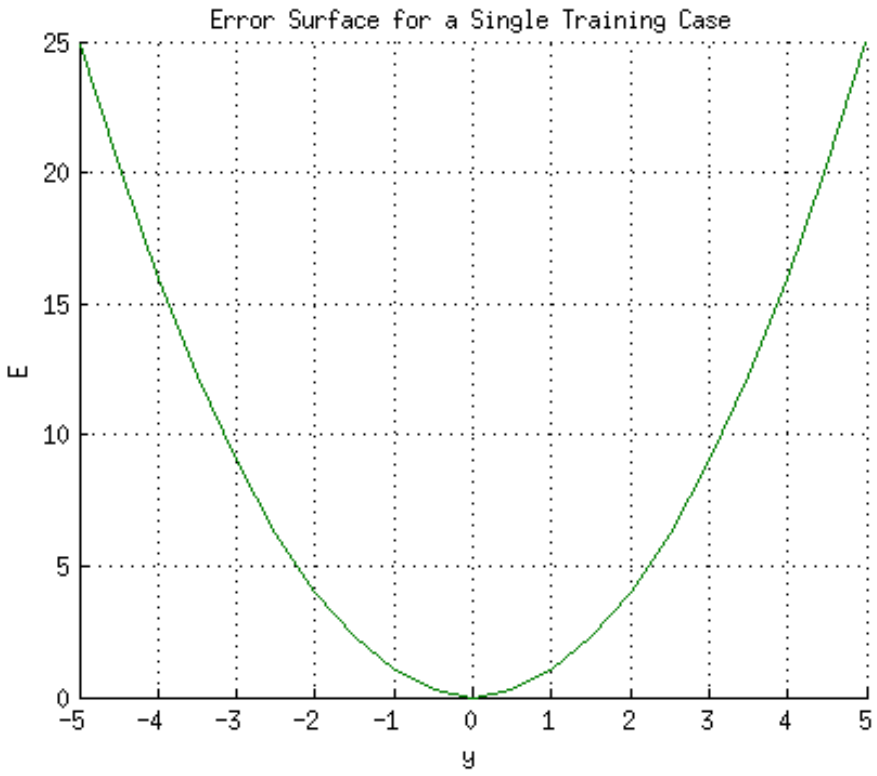

Рисунок 3.7 — Поверхнева помилка лінійного нейрона для одного навчального випадку

Мінімум параболи відповідає виводу *y*, який мінімізує помилку *E*. Для одного навчального випадку мінімум також торкається горизонтальної осі, що означає, що помилка буде нульовою, і мережа може вивести вихід y, який точно відповідає очікуваному виходу *t*. Тому проблему зіставлення входів до виходів можна звести до задачі оптимізації пошуку функції, яка дасть мінімальну помилку. Однак вихід нейрона залежить від зваженої суми всіх його входів:

$$y = x_1 w_1 + x_2 w_2 \quad (3.3)$$

де w_1 і w_2 — ваги на пересіченні від вхідних одиниць до вихідного блоку.

Тому помилка також залежить від вхідних ваг до нейрона, що в кінцевому рахунку є тим, що необхідно змінити в мережі, щоб зробити можливим навчання. Якщо кожен із вагів наноситься на окрему горизонтальну вісь, і помилка на вертикальній осі, то результатом є параболічна чаша. Для нейрона з *k* вагами, та ж сама ділянка вимагала би еліптичного параболоїду розміру *k + 1*. Ілюстрацію поверхневої помилки лінійного нейрона з двома вхідними вагами можна побачити на рисунку 3.8 нижче.

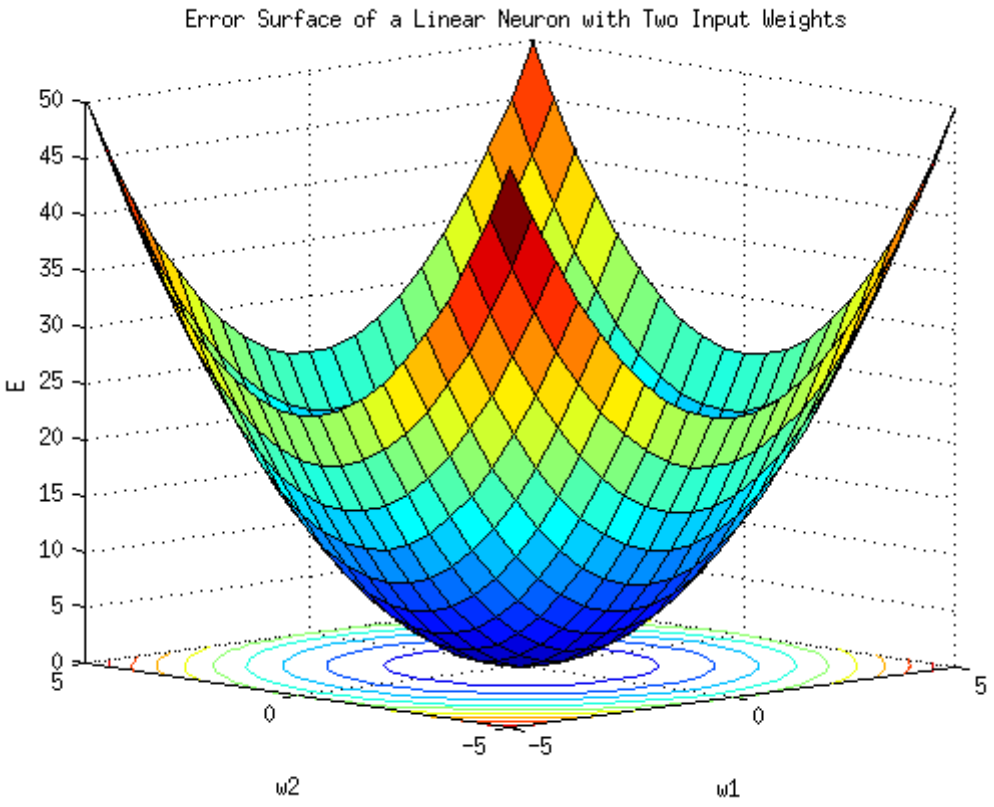

Рисунок 3.8 — Поверхнева помилка лінійного нейрона з двома вхідними вагами

Поверхнева помилка лінійного нейрона з двома вхідними вагами.

Один із часто використовуваних алгоритмів для пошуку набору ваг, що мінімізує помилку, є градієнт спуску.

Як вже було зазначено, із ним використовують зворотне поширення, щоб ефективно обчислити найшвидший напрямок спуску.

Виведення (derivation) для одношарової мережі.

На рисунку 3.9, нижче, наведено схему нейронної мережі, що допоможе у розумінні позначень, наведених у формулах, що будуть приведені нижче.

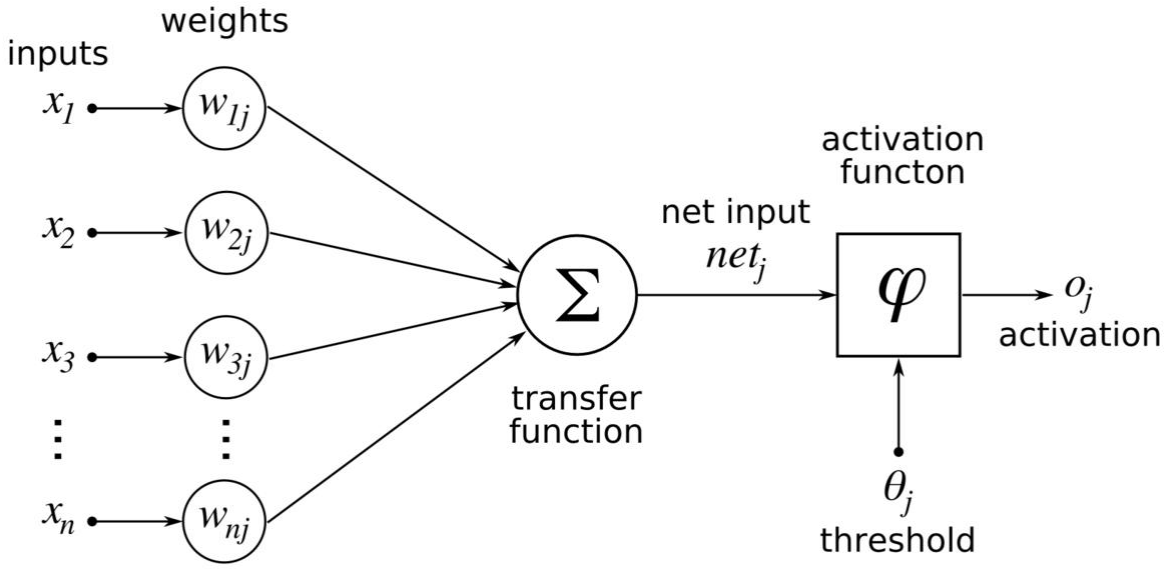

Рисунок 3.9 – Схема штучної нейронної мережі для ілюстрації позначень, що використані у формулах.

Метод градієнтного спуску передбачає обчислення похідної функції витрат по відношенню до вагів мережі. Зазвичай це робиться за допомогою зворотного поширення. Припускаючи, що один вихідний нейрон квадратичної функції помилки є:

$$E = L(t,y) \quad (3.4)$$

де E — втрати для виходу y і цільового значення t,

t — цільовий вихід для навчальної вибірки,

y — фактичний вихід вихідного нейрона.

$$o_j = \varphi(\mathrm{net}_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right).$$

Для кожного нейрона *j*, його вихід o_j визначається як:

У зазначеному рівнянні функція активації φ — нелінійна і диференційована (навіть якщо ReLU не в одній точці). Історично використовується функція активації, що є логістичною функцією:

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

Яка має зручну похідну:

$$\frac{d\varphi(z)}{dz} = \varphi(z)(1 - \varphi(z))$$

Вхідне net_j до нейрона — це зважена сума виходів o_k попередніх нейронів. Якщо нейрон знаходиться в першому шарі після вхідного шару, то o_k вхідного шару — це просто входи x_k в мережу.

Кількість вхідних одиниць до нейрона становить *n*. Змінна w_{kj} позначає вагу між нейроном *k* попереднього шару і нейроном *j* поточного шару.

Пошук похідної помилки.

Обчислення часткової похідної помилки відносно ваги w_{kj} здійснюється за допомогою використання правила ланцюга двічі:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net}_j} \frac{\partial \mathrm{net}_j}{\partial w_{ij}}$$

В останньому факторі правої частини формули наданої вище, тільки один член *y* суми net_j залежить від w_{ij}, так що:

$$\frac{\partial \mathrm{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=1}^{n} w_{kj} o_k \right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i .$$

Якщо нейрон знаходиться в першому шарі після вхідного шару, o_i буде дорівнювати просто x_i.

Похідна виходу нейрона j щодо його входу є просто частковою похідною функції активації (припускаючи, що використовується логістична функція):

$$\frac{\partial o_j}{\partial \mathrm{net}_j} = \frac{\partial}{\partial \mathrm{net}_j} \varphi(\mathrm{net}_j) = \varphi(\mathrm{net}_j)(1 - \varphi(\mathrm{net}_j))$$

Що для випадку функції логістичної активації буде дорівнювати:

$$\frac{\partial o_j}{\partial \mathrm{net}_j} = \frac{\partial \varphi(\mathrm{net}_j)}{\partial \mathrm{net}_j}$$

Це є причиною того, що зворотне поширення вимагає диференційованої функції активації. (Тим не менш, функція активації ReLU, яка не є диференційованою при 0, стала досить популярною, наприклад, в AlexNet)

Перший чинник можна сразу оцінити, якщо нейрон знаходиться у вихідному шарі, тому що $o_j = y$ та

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial L(t, y)}{\partial \varphi(y)} \frac{d\varphi(y)}{dy}$$

Якщо логістична функція використовується як активаціонна, а квадратична помилка як функція витрат, ми можемо її переписати як:

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2}(t - y)^2 = y - t$$

Однак, якщо j знаходиться у довільному внутрішньому шарі мережі,

знаходження похідної E щодо o_j менш очевидне. Розглядаючи E як функцію з входами, всі нейрони $L=\{u,v,\ldots,w\}$ отримують вхід з нейрона j,

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(\mathrm{net}_u, \mathrm{net}_v, \ldots, \mathrm{net}_w)}{\partial o_j}$$

Беручи повну похідну по o_j, отримуємо рекурсивне вираження для похідної:

$$\frac{\partial E}{\partial o_j} = \sum_{\ell \in L} \left(\frac{\partial E}{\partial \mathrm{net}_\ell} \frac{\partial \mathrm{net}_\ell}{\partial o_j} \right) = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \frac{\partial o_\ell}{\partial \mathrm{net}_\ell} \frac{\partial \mathrm{net}_\ell}{\partial o_j} \right) = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \frac{\partial o_\ell}{\partial \mathrm{net}_\ell} w_{j\ell} \right)$$

Тому похідну відносно, o_j можна обчислити, якщо відомі всі похідні відносно виходів o_ℓ наступного шару — ближче до вихідного нейрона.

Таким чином отримуємо:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} o_i$$
$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_j$$

Та отримуємо із неї:

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} \frac{\partial L(o_j,t)}{\partial \varphi(o_j)} \frac{d\varphi(o_j)}{do_j} \\ \left(\sum_{\ell \in L} w_{j\ell} \delta_\ell\right) \frac{d\varphi(o_j)}{do_j} \end{cases}$$

де φ — логістична функція,

помилка — квадратична помилка:

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} (o_j - t_j) o_j (1 - o_j) \\ \left(\sum_{\ell \in L} w_{j\ell} \delta_\ell\right) o_j (1 - o_j) \end{cases}$$

Щоб оновити вагу w_{ij}, використовуючи градієнтний спуск, необхідно вибрати швидкість навчання, $\eta > 0$. Зміна ваги повинна відображати вплив на E

збільшення або зменшення w_{ij}. Якщо:

$$-\eta \frac{\partial E}{\partial w_{ij}}$$

Тоді збільшення w_{ij} збільшує E; навпаки, якщо:

$$\frac{\partial E}{\partial w_{ij}} > 0,$$

Тоді збільшення w_{ij} зменшує E. Новий Δw_{ij} додається до старої ваги, а продукт швидкості навчання і градієнта, помножений на -1, гарантує, що w_{ij} змінюється таким чином, що E завжди зменшується. Іншими словами, в рівнянні нижче, завжди змінює w_{ij} таким чином, що E зменшується:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j$$

Розглянемо функцію втрат. Функція втрат є функцією, яка відображає значення одного або декількох змінних на дійсне число інтуїтивно, що представляють деяку «вартість», пов'язану з цими значеннями. Для зворотного поширення, функція втрат обчислює різницю між вихідними сигналом мережі та його очікуваним виходом, після того, як навчальний приклад поширився через мережу.

Припущення. Математичне вираження функції втрат повинно виконувати дві умови для того, щоб воно могло бути використане у зворотному поширенні [28]. Перший полягає в тому, що воно може бути записане у вигляді середнього:

$$E = \frac{1}{n} \sum_x E_x$$

Це середнє над функціями помилок E_x, для n індивідуальних прикладів

навчання, *x*.

Причина цього припущення полягає в тому, що алгоритм зворотного поширення обчислює градієнт функції помилки для одного прикладу навчання, який необхідно узагальнити на загальну функцію помилки. Друге припущення полягає в тому, що його можна записати як функцію виходів з нейронної мережі.

Приклад функції втрат. Нехай y, y' — вектори в R^n. Виберемо функцію помилки $E(y, y')$, яка вимірює різницю між двома виходами. Стандартним вибором є квадрат евклідової відстані між векторами y і y':

$$E(y, y') = \tfrac{1}{2}\|y - y'\|^2$$

Функція помилки над n навчальними прикладами може бути записана як середня втрат над окремими прикладами:

$$E = \frac{1}{2n}\sum_x \|(y(x) - y'(x))\|^2$$

Обмеження методу градієнтного спуску. Градієнтний спуск може знайти місцевий мінімум замість глобального мінімуму.

Градієнтний спуск із зворотним поширенням не гарантує знайдення глобального мінімуму функції помилки, а лише локальний мінімум; крім того, у нього є проблеми з переходом плато в помилці функції ландшафту. Це питання, викликане невипуклою функцією помилок в нейронних мережах, довгий час вважалося головним недоліком, але Yann LeCun та інші дослідники стверджують, що в багатьох практичних завданнях це не так [29].

Навчання зворотного поширення не вимагає нормалізації вхідних векторів, однак нормалізація може підвищити продуктивність [30].

Тепер розглянемо власне градієнтний спуск.

Градієнтний спуск – це алгоритм оптимізації, який використовується для мінімізації деякої функції шляхом ітеративного переміщення в напрямку

найшвидшого спуску, визначеного негативом градієнта.

У машинному навчанні градієнтний спуск використовується для оновлення параметрів моделі. Параметри відносяться до коефіцієнтів лінійної регресії і вагів в нейронних мережах.

Розглянемо 3-вимірний граф, наведений на рисунку 3.10 нижче, в контексті функції витрат.

Метою є перейти від гори у верхньому правому куті (висока вартість) до темно-синього моря в нижньому лівому куті (низька вартість). Стрілки являють собою напрямок крутого спуску (негативний градієнт) з будь-якої заданої точки, тобто напрямок, що зменшує функцію витрат якомога швидше.

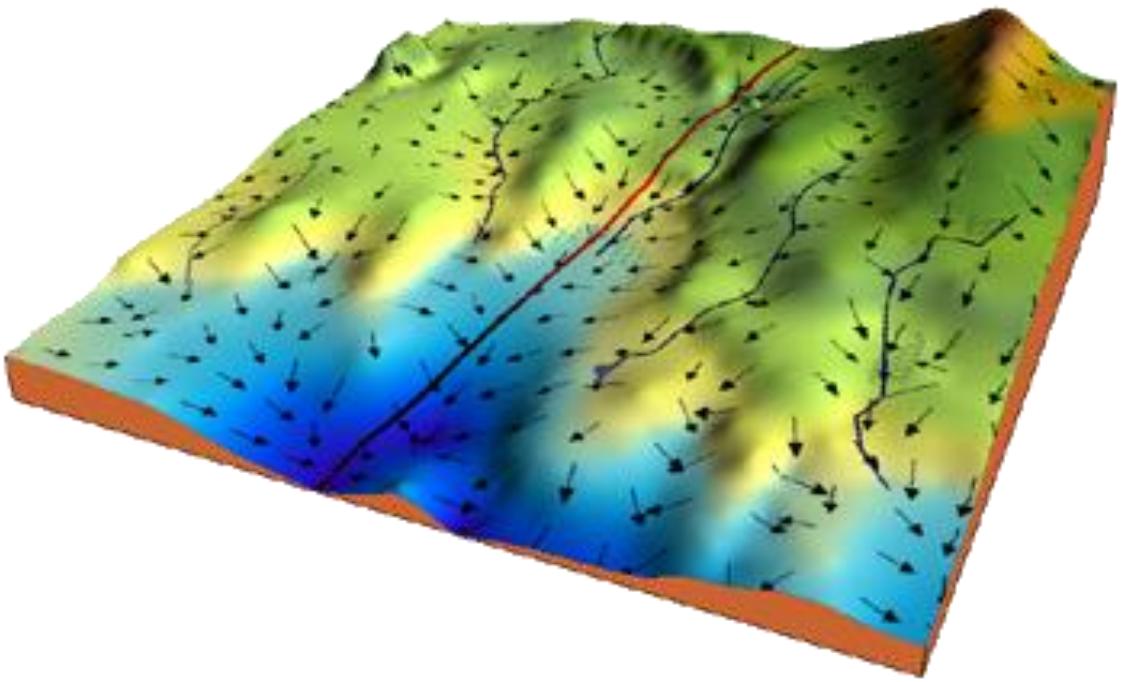

Рисунок 3.10 — 3-вимірний граф в контексті функції витрат [31]

Починаючи з вершини гори, потрібно зробити перший крок у напрямку, визначеному негативним градієнтом.

Далі потрібно перерахувати негативний градієнт (в координатах нової точки) і зробити ще один крок у напрямку, який він вказує. Потрібно продовжувати цей процес ітеративно, поки не буде досягнуто нижньої частини графа, або точки, де більше неможливо рухатися вниз — місцевий мінімум. Ілюстрацію можна побачити на рисунку 3.11 нижче.

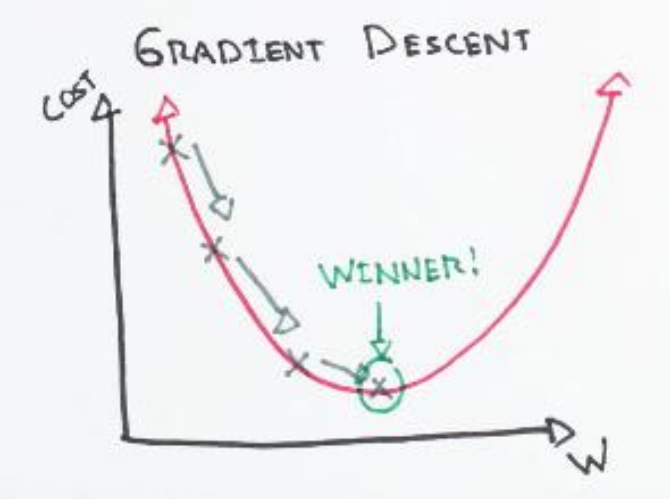

Рисунок 3.11 — процес проходження графу на прикладі однієї ітерації.

Розмір цих етапів, кроків називається швидкістю навчання. При високій швидкості навчання можна охопити більше місця на кожному кроці, але це визиває ризик перевищення найнижчої точки, оскільки нахил пагорба постійно змінюється. При дуже низькій швидкості навчання можна впевнено рухатися в напрямку негативного градієнта, оскільки він перераховується часто. Низький рівень навчання є більш точним, але обчислення градієнта займає багато часу, тому знадобиться дуже багато часу, щоб дістатися до дна. Функція втрати повідомляє, наскільки добре модель робить прогнози для даного набору параметрів. Функція витрат має власну криву та власні градієнти. Нахил цієї кривої говорить про те, як оновити параметри, щоб зробити модель більш точною. Запустимо градієнтний спуск за допомогою нашої нової функції витрат. У нашій функції витрат ми можемо керувати двома параметрами: m (вага) та b (зміщення).

Оскільки повинно враховувати вплив, який кожен має на кінцевий прогноз, ми повинні використовувати часткові похідні. Розраховуємо часткові

похідні функції вартості по відношенню до кожного параметра і зберігаємо результати в градієнті. Скажімо, маємо наступну функцію витрат:

$$f(m, b) = \frac{1}{N} \sum_{i=1}^{n} (y_i - (mx_i + b))^2$$

Градієнт можна обчислити як:

$$f'(m, b) = \begin{bmatrix} \frac{df}{dm} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum -2x_i(y_i - (mx_i + b)) \\ \frac{1}{N} \sum -2(y_i - (mx_i + b)) \end{bmatrix}$$

Тепер, маючи уявлення про методи, якими ми будемо керуватися у розробці змагального прикладу, та реалізації атаки, перейдемо до їх застосування.

3.3.2 Опис реалізації атаки

Спочатку візьмемо зображення кота та проведемо його через нейронну мережу, щоб переконатися, що воно класифікується, як кіт. На рисунку 3.12 , нижче, наведено початкове зображення кота

Рисунок 3.12 – Початкове зображення кота

Загрузимо зображення та віддамо його моделі на передбачення. Реалізацію цього кроку можна побачити на рисунку 3.13.

```
img = io.imread(
'gdrive/MyDrive/cats-dogs-network/data/examples/real-cat.jpg'
)
image = np.expand_dims(img, axis=0)

res = base_model.predict(image)

print(res, class)
```

Рисунок 3.13 — Загрузка зображення та виклик методу передбачення класу

За результатами передбачення маємо вивід, наведений на рисунку 3.14 нижче.

```
0.9615, cat
```

Рисунок 3.14 — Результати передбачення

Цей вивід означає, що на зображенні кіт із вірогідністю 96%.

Перейдемо до створення змагального прикладу. Маючи зображення x, наша нейронна мережа виводить розподіл ймовірностей над мітками, $P(y/x)$. Коли розробляється конкурсний вхід, потрібно знайти $\hat{x}$, де $logP(\hat{y}\square\hat{x})$ максимізовано для цільової мітки $\hat{y}$: таким чином, вхідні дані будуть неправильно класифіковані як цільовий клас, тобто собака. Можна переконатися, що $\hat{x}$ не виглядає занадто відмінним від початкового x, із обмеженням ℓ_∞, радіусом ϵ, щоб:

$$\|x - \hat{x}\|_\infty \leq \epsilon.$$

Спочатку проводимо ініціалізацію змагального прикладу як $\hat{x} \leftarrow x$. Потім повторюємо приведені нижче кроки до збіжності:

$$\hat{x} \leftarrow \hat{x} + \alpha \, \square \, \square logP\,(\hat{y} \square \hat{x})$$

$$\hat{x} \leftarrow clip(\hat{x},\, x - \epsilon,\, x + \epsilon)$$

Clip у другій формулі означає обрізання під поріг, тобто якщо градієнти більше певного порогу, то вважаємо, що вони рівні цьому порогу.

Крок ініціалізації.

Спочатку потрібно написати операції TensorFlow для ініціалізації – їх наведено нижче, на рисунку 3.15.

```
base_x = tf.placeholder(tf.float32, (299, 299, 3))

base_x_hat = img # here we download adversarial example input
assign_op = tf.assign(base_x_hat, base_x)
```

Рисунок 3.15 — Код операцій

Етап градієнтного спуску. Далі потрібно описати крок градієнтного спуску, щоб максимізувати логарифмічну ймовірність цільового класу (або еквівалентно мінімізувати перехресну ентропію – перехресна ентропія між двома розподілами ймовірностей p і q над одним і тим же базовим набором подій вимірює середню кількість бітів, необхідних для ідентифікації події, взятої з набору, якщо схема кодування, що використовується для набору, оптимізована для розрахункового розподілу ймовірностей q, а не істинного розподілу p). Перейдемо до етапу проекції. Необхідно описати крок проекції, щоб змагальний приклад був візуально близьким до вихідного зображення. Додатково, треба відсікти (clip) його до [0,1], щоб зберегти зображення валідним. Нижче, на рисунку 3.17, наведений код реалізації цього етапу.

```
eps = tf.placeholder(tf.float32, ())

under = base_x - eps
over = base_x + eps

proj = tf.clip_by_value(tf.clip_by_value(base_x_hat,
                                         under, over), 0, 1)

with tf.control_dependencies([proj]):
  proj_step = tf.assign(base_x_hat, proj)
```

Рисунок 3.17 — Крок проекції

Тепер перейдемо до виконання.

Синтезуємо змагальний приклад. Було обрано другий клас, тобто собаку, як цільовий клас.

Нижче, на рисунку 3.18, приведена реалізація.

Після виконання у виводі було отримано показники, що наведені нижче, на рисунку 3.19.

Шум, що накладається на оригінальне зображення виглядає таким чином, як показано на рисунку 3.20.

```
dem_eps = 2.0/255.0 # a little perturbation
dem_lr = 1e-1
dem_step = 100
dem_tar = 2 # dog class

# step of initialization
sess.run(assign_op, feed_dict={x: img})

# PGD - projected gradient descent
for i in range(dem_step):
# step of PGD - projected gradient descent
  _, loss_val = sess.run([optimal_step, f_loss],
                         f_dict={learning_rate: demo_lr,
                                 base_y_hat: dem_tar})

# step of projection
sess.run(proj_step, f_dict={base_x: img, eps: dem_eps})
if (i+1) % 10 == 0:
  print('the step is %d, the loss is %g' % (i+1, loss_val))

# get the resulting adversarial example
adversarial_res = base_x_hat.eval()
```

Рисунок 3.18 — Синтезування змагального прикладу

```
the step is 10, the loss is 3.95831
the step is 20, the loss is 0.571342
the step is 30, the loss is 0.0315487
the step is 40, the loss is 0.0214902
the step is 50, the loss is 0.0173401
the step is 60, the loss is 0.0143289
the step is 70, the loss is 0.0121266
the step is 80, the loss is 0.0111390
the step is 90, the loss is 0.00954381
the step is 100, the loss is 0.0089127021
```

Рисунок 3.19 — Loss

Рисунок 3.20 – Шум

3.3.3 Результат атаки на мережу, аналіз отриманих показників

У результаті було отримано змагальний приклад, наведений на рисунку 3.21, нижче. Як можна побачити, зображення візуально не відрізняється від оригіналу та зчитується людським оком як зображення кота.

Рисунок 3.21 – Змагальне зображення кота

Виконуємо перевірку отриманого змагального зображення на нейронній мережі. На рисунку 3.22 приведений код, де ми загружаємо трансформоване зображення кота та віддаємо моделі для аналізу. За результатами передбачення маємо вивід, наведений на рисунку 3.23. Тобто зображення класифіковане нейронною мережею, як собака із вірогідністю 90%.

```
img = io.imread(
'gdrive/MyDrive/cats-dogs-network/data/examples/fake-cat.jpg'
)
image = np.expand_dims(img, axis=0)

res = base_model.predict(image)

print(res, class)
```

Рисунок 3.22 – Загрузка зображення та ініціація передбачення його класу

```
0.9008, dog
```

Рисунок 3.23 – Результат класифікації

3.4 Розробка покращеного методу атаки на нейронну мережу

За мету практичних досліджень атестаційної роботи було поставлено розробку покращеного методу змагальної атаки на нейронні мережі. Таким чином, у попередніх підрозділах розділу практичних досліджень було обрано та розглянуто архітектуру нейронної мережі та навчено мережу за цією архітектурою, після чого було обрано та розглянуто методи атаки на натреновану мережу, розроблено змагальний приклад за цими методами та проведено змагальну атаку. У цьому підрозділі буде нарешті розроблено, розглянуто та примінено покращений метод атаки.

3.4.1 Розгляд показників для покращення, вибір показника

До показників результативності змагальної атаки відносять такі показники як вірогідність належності до певного класу, тобто ступінь впевненості мережі, із якою вона неправильно класифікує зображення; швидкість виконання алгоритму створення змагального зображення; ступінь

відмінності зображення від оригіналу; якість, тобто стійкість змагальних зображень до шуму, деформації та перетворень – питання, що розглядається у роботі “Minimizing Perceived Image Quality Loss Through Adversarial Attack Scoping”, написаній Костянтином Хабарлаком та Ларисою Коряшкіною [32]; кількість запитів до моделі, які потрібно зробити у випадку атаки чорної скриньки, тощо.

Метрикою, що було покращено у цій роботі, є стійкість змагального зображення, яку ще називають «робастність» (від англ. “robust”, тобто міцний, витривалий), бо саме ця метрика є критично важливою із точки зору безпеки.

3.4.2 Опис реалізації покращеного методу змагальної атаки

Переходимо до реалізації більш просунутого, стійкого змагального прикладу, який ще називається робастним.

Будемо слідувати підходу до синтезу надійних змагальних прикладів, щоб знайти таку трансформацію зображення кішки, що буде успішним змагальним прикладом навіть при перетворенні, що складатиметься з повороту зображення на певний кут.

Було вироблено таке змагальне зображення, яке є надійним щодо обертання $\theta \in [-\pi/4, \pi/4]$.

Перш ніж продовжити, варто перевірити, чи попередній приклад залишається змагальним, якщо повернути його, скажімо, на кут $\theta = \pi/8$. На рисунку 3.24 нижче наведено повернутий змагальний приклад.

Перевіримо, як повернуте зображення класифікується нейронною мережею. На рисунку 3.25 наведено код, де ми загружаємо повернутий змагальний приклад та віддаємо мережі на класифікацію. За результатами передбачення маємо вивід, наведений на рисунку 3.26.

Рисунок 3.24 – Змагальне зображення кота із приміненням ротації

```
img = io.imread(
'gdrive/MyDrive/cats-dogs-network/data/examples/rotated-fake-cat.jpg')

image = np.expand_dims(img, axis=0)

res = base_model.predict(image)

print(res, class)
```

Рисунок 3.25 – Загрузка зображення та виклик методу класифікації

```
0.8502, cat
```

Рисунок 3.26 – Результат класифікації

Таким чином, можна побачити, що змагальний приклад повернутий на *π/8,* не проходить класифікацію як собака, а проходить як кіт із зниженим показником вірогідності.

Отже, розглянемо, як можна зробити стійкий до трансформації повороту змагальний приклад. З урахуванням деякого ротаційного перетворення T можна максимізувати $E_{t\sim T}\, logP\ (\hat{y}\ \square\ t\ (\hat{x}))$, враховуючи $\mathbb{l}x - \hat{x}\mathbb{l}_\infty \leq \epsilon$. Цю задачу

оптимізації можна вирішити за допомогою проектованого градієнтного спуску, зазначивши, що $\nabla \square_{t\sim T} logP\ (\hat{y} \square\ (t\ (\hat{x}))$ наближається до зразків на кожному кроці градієнтного спуску.

Щоб застосувати вибірку градієнта, використаємо функціональність TensorFlow: змоделюємо вибіркове сходження на основі градієнта, застосовуючи градієнтний спуск на ансамблі (методи ансамблю використовують кілька алгоритмів навчання, щоб отримати кращу прогностичну продуктивність, ніж можна було отримати з одного з алгоритмів навчання) стохастичних класифікаторів (техніка, за якою класифікатори обирають значення прогнозувань з найбільшою вірогідністю), які випадково вибирають вхідні дані з розподілу та перетворюють їх перш ніж класифікувати.

Нижче, на рисунку 3.27, наведено код, що реалізує цей крок.

```
img = io.imread(
'gdrive/MyDrive/cats-dogs-network/data/examples/rotated-fake-cat.jpg')

n_sampl = 10
avrg_loss = 0

for i in range(n_sampl):
    rottd = tf.contrib.image.rotate(img,
                                    tf.random_uniform(
                                        (),
                                        minval=-np.pi/4,
                                        maxval=np.pi/4))

    rottd_logits, _ = inception(rottd, reuse=True)

avrg_loss += tf.nn.softmax_cross_entropy_with_logits(logits=rottd_logits,
                                                     labels=g_labels)
```

Рисунок 3.27 – Вибіркове сходження з градієнтним спуском

Можна повторно використовувати assign_op і proj_step, хоча доведеться написати новий optimal_step для цієї нової мети. На рисунку 3.28 наведено код реалізації нового оптимального кроку.

```
optimal_step = tf.train.GradientDescentOptimizer(learning_rate)
                        .minimize(avrg_loss, var_list=[base_x_hat])
```

Рисунок 3.28 – Реалізація нового оптимального кроку

Нарешті, можна запустити проектований градієнтний спуск для створення змагального прикладу.

Як і в попередньому прикладі, виберемо другий клас, тобто «собака» як цільовий клас.

На рисунку 3.29 можна побачити реалізацію. Після виконання, маємо вивід, зазначений на рисунку 3.30.

За результатами обробки, можна побачити у виводі, що змагальне зображення класифікується як кішка із високою вірогідністю.

```
dem_eps = 8.0/255.0 # a little distortion still
dem_lr = 2e-1
dem_stps = 300
dem_trgt = 924 # "dog" class

# the step of initialization
sess.run(assign_op, f_dict={x: img})

# PGD - projected gradient descent
for i in range(dem_stps):
    # the step of the gradient descent
    _, loss_val = sess.run(
        [optimal_step, avrg_loss],
        f_dict={learn_rate: demo_lr, base_y_hat: dem_trgt})
    # the step of the project
    sess.run(proj_step, f_dict={base_x: img, eps: dem_eps})
    if (i+1) % 50 == 0:
        print('the step is %d, the loss is %g' % (i+1, loss_v))

        # get the produced adversarial example itself
adv_robust = base_x_hat.eval()
```

Рисунок 3.29 – Реалізація проектованого градієнтного спуску

```
the step is 50, the loss is 0.0803217
the step is 100, the loss is 0.0221388
the step is 150, the loss is 0.00732416
the step is 200, the loss is 0.00391829
the step is 250, the loss is 0.00626519
the step is 300, the loss is 0.00234199
```

Рисунок 3.30 – Loss

3.4.3 Аналіз результату атаки на мережу після покращення

Перевіримо, як класифікується отриманий стійкий змагальний приклад, що наведено на рисунку 3.31, нижче, нейронною мережею. На рисунку 3.32, нижче, наведений код, що загружає зображення та проводить його через нейронну мережу для класифікації. За результатами передбачення маємо вивід, що наведений на рисунку 3.33.

Рисунок 3.31 – Стійкий змагальний приклад

```
img = io.imread(
'gdrive/MyDrive/cats-dogs-network/data/examples/
                robust-rotated-fake-cat.jpg')

image = np.expand_dims(img, axis=0)

res = base_model.predict(image)

print(res, class)
```

Рисунок 3.32 – Загрузка та класифікація змагального прикладу

```
0.9514, dog
```

Рисунок 3.33 – Результат класифікації

Таким чином, можна переконатися в успішності проведеної атаки та побачити, що нейронна мережа класифікує змагальний приклад кота як собаку із високою вірогідністю (0.9514).

Оцінимо результат.

Розглянемо обертальну інваріантність, тобто стійкість до перетворення через поворот, робастного (стійкого) змагального прикладу, який було створено з огляду на весь діапазон кутів, розглядаючи $P(\hat{y} \mid \hat{x})$ над $\theta \in [-\pi/4, \pi/4]$.

На рисунку 3.34 наведений код, що реалізує побудова графіку, який ілюструє результати.

Після виконання зазначеного коду маємо графік, що наведений на рисунку 3.35, нижче, на якому видно залежність успішності атаки від куту наклону змагального приклада.

Таким чином, на графику можна побачити, що наш змагальний приклад є інваріантним до зміни кута ротації, тобто не залежить від нього і змагальна атака із використанням цього змагального прикладу буде успішною, незалежно від кута оберту зображення.

```
rttd_image = io.imread(
    'gdrive/MyDrive/cats-dogs-network/data/examples/robust-rotated-fake-cat.jpg

thts = np.linspace(-np.pi/4, np.pi/4, 301) # thetas

p_simple = [] # naive
p_rbst = [] # robust

# tht stands for theta
for tht in thts:
        # rotated
    rttd = rttd_image.eval(f_dict={image: adversarial_rbst, angle: tht})
    p_rbst.append(probs.eval(f_dict={image: rttd})[0][dem_trgt])

    rttd = rttd_image.eval(f_dict={image: adversarial, angle: tht})
    p_simple.append(probs.eval(f_dict={image: rttd})[0][dem_trgt])

# robust line
rbst_l, = plt.plot(thts, p_rbst, color='b', linewidth=2, label='robust')
simple_line, = plt.plot(thts, p_simple, color='r', linewidth=2, label='naive')

plt.ylim([0, 1.05])
plt.xlabel('rotation angle')
plt.ylabel('target class probability')

plt.legend(handles=[robust_line, naive_line], loc='lower right')
plt.show()
```

Рисунок 3.34 – Код побудови графіку для демонстрації результатів

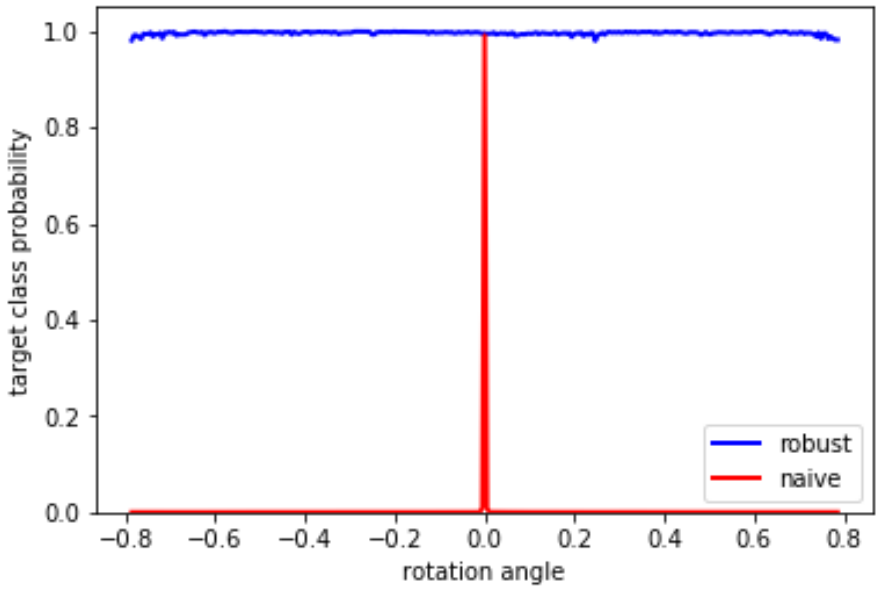

Рисунок 3.35 – Графік, що ілюструє успішність атаки, залежно від куту ротаці

4 ОПИС МОЖЛИВОСТЕЙ ВИКОРИСТАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ У НАУКОВІЙ І ПРАКТИЧНІЙ ДІЯЛЬНОСТІ

Машинне навчання має справу з так званим відомим невідомим: передбачення невідомих точок даних з відомого розподілу. Захист від нападів стосується відповідно невідомих невідомих: надійне прогнозування невідомих точок даних з невідомого розподілу змагальних вхідних даних. Оскільки машинне навчання інтегрується у все більше систем, таких як автономні транспортні засоби або медичні пристрої, вони також стають пунктами входу для атак. Навіть якщо передбачення моделей машинного навчання на тестовому наборі даних є 100% правильним, можна знайти чи створити змагальні приклади, що зможуть обманути модель. Захист моделей машинного навчання від атак є новою частиною сфери кібербезпеки.

Змагальні атаки є важливою темою дослідження і розгляду, оскільки було показано, що змагальні приклади переходять з однієї моделі в іншу, тобто змагальні приклади, створені для обману однієї моделі, можуть також обманути інші моделі, використовуючи іншу архітектуру або навчені з використанням різних наборів даних для одного і того ж завдання. Зараз це стає величезним ризиком для безпеки, оскільки зловмисники можуть розробляти локальні моделі для того самого завдання, що і цільова модель, генерувати змагальні приклади для локальної моделі (випадок «білої скриньки») і використовувати їх для нападу на ціль (це надає можливість простіше переносити атаки на цільові моделі на «чорні скриньки»). Ця проблема ставить під ризик цілий ряд широко використовуваних додатків та систем, таких як системи розпізнавання обличчя, автономно керовані автомобілі, біометричне розпізнавання, тощо.

Стаття, “Adversarial Examples In The Physical World” написана Alexey Kurakin, Ian J. Goodfellow та Samy Bengio [33] досліджує, наскільки реальною є ця загроза, враховуючи, що в багатьох реальних, а не імітованих дослідниками, випадках атак дані надходять з камер та інших пристроїв. Це демонструє, що навіть подача змагальних прикладів через камеру, а не лише програмне подання

даних, призводить до неправильної класифікації. “Synthesizing robust adversarial examples” від Athalye та інших дослідників [21], на яку вже було звернено увагу у цій роботі, є іншою роботою, що демонструє реальну загрозу, через генерування прикладів, які залишаються надійними (робастними), при перетвореннях поворотом, і в якій використано 3D-друк для створення надійних 3D змагальних об'єктів у реальному світі.

Очевидно, що змагальні атаки становлять серйозну загрозу для сучасносного високотехнологічного світу, на даний час існує дві категорії захисту: змагальна підготовка і оборонна дистиляція, розглянемо їх.

Один з найпростіших і найбільш грубих способів захисту від змагальних атак – це симулювання зловмисника, тобто генерація ряду змагальних прикладів проти власної мережі, а потім навчання моделі не реагувати на атаки такого роду, тобто класифікувати змагальні зображення правильно, не зважаючи на перетворення, застосовані із метою обману нейронної мережі. Це покращує генералізацію моделі, але не дає змоги забезпечити необхідний рівень надійності – насправді, це просто перетворюється на нескінченну гонитву атакуючих і захисників.

У другому способі захисту – оборонній дистиляції (англ. defensive distillation) – ми навчаємо вторинну модель, що є «згладженою» в напрямках, які зловмисник, як правило, намагається використати, тобто характеристики моделі є не очевидними, що ускладнює для атакуючих вироблення змагальних вхідних даних, які призводять до неправильної категоризації. Причина цього полягає в тому, що на відміну від першої моделі, друга модель навчається на «гнучких», виходах первинної моделі, а не на «жорстких» істинних мітках з реальних навчальних даних. Показано, що цей метод має певний успіх у захисті початкових варіантів змагальних атак, але він був «переможений» більш пізніми атаками, наприклад, атакою Carlini-Wagner [9], на яку вже було звернено увагу у цій роботі раніше, яка є поточним еталоном для оцінки надійності нейронної мережі проти змагальних атак.

Таким чином, стає зрозуміло, що у сучасному та скоріш за все майбутньому світі, через розповсюдження нейронних мереж у нашому житті, та

критичності аспекту безпеки у галузях, у яких мережі запроваджуються, вивчення змагальних атак та вдосконалення методів створення змагальних прикладів стає чим далі, тим більше затребуваною темою теоретичних та практичних досліджень, що вимагає рішень вже зараз.

ВИСНОВКИ

В результаті виконання атестаційної роботи було розглянуто наукову і патентну літературу на тему змагальних атак, проаналізовано стан вирішення проблеми, проведено теоретичні дослідження методів змагальних атак, таких як оптимізаційний підхід на основі градієнта, метод 1-піксельної атаки, метод змагального патча, метод стійких або робастних змагальних прикладів, метод атаки чорної скриньки, метод зворотного поширення помилки та метод проектованого градієнтного спуску. Методом атаки, який було обрано для практичного дослідження та вдосконалення була комбінація методів зворотного поширення помилки та метод проектованого градієнтного спуску та проектованого градієнтного спуску. Критеріями оцінки успішності атаки із використанням цих методів була робастність (англ. robust), тобто стійкість змагальних прикладів до перетворень. Саме ці критерії було обрано через їх критичність з боку безпеки, бо те, наскільки витривалим є змагальний приклад, означає чи зможе він змусити нейронну мережу класифікувати його певним образом, що вигідний атакуючому, чи ні, у той час як такі критерії, як час виготовлення змагального прикладу чи швидкість виконання алгоритму створення змагального зображення, тощо не є критичними для цілі – успішної змагальної атаки. В ході практичного дослідження було натреновано нейронну мережу за архітектурою Xception та застосовано на ній обрані методи змагальних атак. За результатами застосування отриманого змагального прикладу, виявилося, що оригінальна атака є успішною, але змагальний приклад не є стійким до перетворень. Таким чином, для того, щоб виробити робастний змагальний приклад, потрібно поліпшити метод атаки. Тому у наступній частині практичних досліджень було проведено роботу над покращенням методу змагальної атаки таким чином, щоб в результаті вироблення змагального прикладу, отримати стійкий до перетворень повороту приклад. Із урахуванням ротаційного перетворення було отримано задачу оптимізації, яку було вирішено за допомогою проектованого градієнтного

спуску через моделювання вибіркового сходження на основі градієнта, застосовуючи градієнтний спуск на ансамблі стохастичних класифікаторів. В результаті було отримано змагальні приклади, які класифікуються нейронною мережею, як таргетований клас при будь-якому куті ротації, тобто є робастними.

Отримане вдосконалення методу змагальної атаки вказує на поточні вразливості нейронних мереж та потенціал для зловмисного використання, тому результати роботи можуть послужити у проектуванні більш надійних нейронних мереж, що є безпечнішими для впровадження у системи, що працюють із розпізнаванням зображень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. С. Литвинов. Як обманути алгоритм розпізнавання облич: швидка генерація змагальних даних – 15.10.2018. URL:https://neurohive.io/ru/papers/kak-obmanut-algorim-raspoznavanija-lic-bystraja-generacija-sostjazatelnyh-dannyh/
2. Стаття розміщена компанією Inflexion. Adversarial attacks: How to trick computer vision – 14.12.2018. URL:https://hackernoon.com/adversarial-attacks-how-to-trick-computer-vision-7484c4e85dc0
3. N. Akhtar, A. Mian. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey – Journal Of Latex Class Files, Vol. Pp, August 2017
4. J. Frankenfield. 51% Attack – 6.05.2019. URL:https://www.investopedia.com/terms/1/51-attack.asp
5. H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, Jürgen Schmidhuber. A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks – Journal Advanced Robotics, 02.03.2012.
6. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus. Intriguing properties of neural networks – 19.04.2014.
7. P. Hocquet. Trust, and don't verify: the AI black box problem – 01.10.2017.
8. M. Sharif, S. Bhagavatula, L. Bauer. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition – 2016.
9. A. S. Ross, F. Doshi-Velez. Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing their Input Gradients – Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, USA.
10. N. Carlini, D. Wagner. Towards Evaluating the Robustness of Neural Networks – Cornell University, 16.08.2016.
11. A. Nayebi, S. Ganguli. Biologically inspired protection of deep networks from adversarial attacks – 27.03.2017

12. W. Brendel, M. Bethge. Comment on “Biologically inspired protection of deep networks from adversarial attacks” – Cornell University, 05.04.2017.
13. N. Akhtar A. Mian. Threat of Adversarial Attacks on Deep Learning Computer Vision: A Survey – Journal Of Latex Class Files, Vol. Pp, August 2017.
14. D. Gragnaniello, F. Marra, G. Poggi, L. Verdoliva. Analysis of adversarial attacks against CNN-based image forgery detectors – Department of Electrical Engineering and Information Technology, University Federico II of Naples, Naples, Italy.
15. N. Carlini, D. Wagner. Audio Adversarial Examples: Targeted Attacks on Speech-to-Text – University of California, Berkeley, 2018.
16. S. Samanta, S. Mehta. Towards Crafting Text Adversarial Samples – IBM India Research Lab, 2017.
17. Розміщено кафедрою комп’ютерник наук Стенфордського университету. Generative Adversarial Networks.
URL: https://cs.stanford.edu/people/karpathy/gan/
18. J. Snow. To protect artificial intelligence from attacks, show it fake data – 26.03.2018. URL:https://www.technologyreview.com/s/610656/to-protect-artificial-intelligence-from-attacks-show-it-fake-data/
19. I. J. Goodfellow, J. Shlens, C. Szegedy. Explaining and Harnessing Adversarial Examples – Cornell University, 20.12.2014.
20. J. Su, D. V. Vargas, K. Sakurai. One pixel attack for fooling deep neural networks – IEEE Transactions on Evolutionary Computation, 2019.
21. T. B. Brown, D. Mané, A. Roy, M. Abadi, J. Gilmer. Adversarial patch – Cornell University, 2017.
22. A. Athalye, L. Engstrom, A. Ilyas, K. Kwok. Synthesizing robust adversarial examples – Cornell University, 2017.
23. N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Berkay Celik, A. Swami. Practical black-box attacks against machine learning – Asia Conference on Computer and Communications Security, ACM, 2017.

24. Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition – Neural networks: the statistical mechanics perspective, 261:276, 1995.
25. Colah's Blog. Calculus on Computational Graphs: Backpropagation – 31.08.2015. URL: http://colah.github.io/posts/2015-08-Backprop/
26. ML Cheatsheet. URL:https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html
27. Adalta. A Powerful Contouring, Gridding, and Surface Mapping Package for Scientists and Engineers. URL:https://www.adalta.it/Pages/-GoldenSoftware-Surfer-010.asp
28. D.E. Rumelhart, G.E. Hinton, R.J. Williams Learning Representations by Back-Propagating Errors – University of California, 1986.
29. Nielsen, Michael A Neural Networks and Deep Learning – 2015.
30. LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey. Deep learning – Nature. 521 (7553): 436–444, 2015.
31. M. Buckland; M. Collins. AI Techniques for Game Programming – ISBN 978-1-9318Minimizing Perceived Image Quality Loss Through Adversarial Attack Scoping
32. K. Khabarlak, L. Koriashkina
33.. Minimizing Perceived Image Quality Loss Through Adversarial Attack Scoping – 2019.
34. A. Kurakin, I. J. Goodfellow, S. Bengio. Adversarial Examples In The Physical World – Workshop track - ICLR, 2017