Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики

Ефективні контроверсійні атаки на нейронні мережі розпізнання мови

Текстова частина до курсової роботи за спеціальністю
«Інженерія програмного забезпечення» 121

Керівник курсової роботи
канд. фіз-мат наук Крюкова Г. В.
_____
(підпис)
«___» _____ 2021 р.

Виконав студент
Кучер А. О
«___» _____ 2021 р.

м. Київ 2021

Зміст

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Огляд технічної літератури для аналізу теми роботи	05.12.2020	
2.	Аналіз та ознайомлення з літературою	10.01.2021	
3.	Аналіз архітектури нейронної мережі, її розробка	10.02.2021	
4.	Розробка і виконання атак на мережу	10.03.2021	
5.	Аналіз отриманих результатів, написання текстової частини	10.04.2021	
6.	Корегування і остаточне оформлення роботи	10.05.2021	
7.	Створення і оформлення презентації	15.05.2021	

Вступ

Нейронні мережі стали дуже популярними за досить короткий проміжок часу для історії і стали використовуватися у багатьох сферах. Так як ця технологія ще не є до кінця вивченою людиною, знаходяться і виявляються нові вразливості, які дають змогу маніпулювати вже навченими нейронними мережами. Одною з таких вразливостей і є, наприклад, контроверсійні атаки. Зміни досить незначні для того, щоб бути виявленими людиною, але вагомі для мережі. В таких сферах використання цієї технології як камери для розпізнавання особистості, системах виявлення вадливого програмного забезпечення ці недоліки стали головним пріоритетом і загрозою безпеки.

Об’єктом дослідження є фреймворки і системи глибокого навчання відкритого і закритого типу

Предметом дослідження є контроверсійні атаки на такі системи

Метою дослідження є розкриття можливостей захисту від подібних атак і аналізу їх вагомості

Анотація

Роботу присвячено аналізу контроверсійних атак на нейронні мережі та можливому захисту від цих атак. У ході дослідження створено нейронну мережу автоматичного розпізнання голосу, створені та проведені контроверсійні атаки. Проаналізований захист від подібних атак. Зроблені висновки і поданий подальший можливий розвиток і дослідження у майбутніх роботах.

1. Огляд існуючих рішень створення контроверсійних предметів

Аналізуючи можливості нападу на мережу та порівнюючи між собою способи атаки - структуровані такі пункти: для того щоб напасти на нейронну мережу контроверсійною атакою штучно створюється набір даних. Цей набір є результатом або змін за допомогою функцій або виходом від нейронної мережі, яка створена для відтворення мережі на яку планується атака. При цьому вибір атаки цілком залежить від інформації, яка доступна про нейронну мережу.

Згідно з роботою Папернота [1], всього виділяється 20 випадків, які різняться відносно їх складності по осі X і доступній інформації по осі У. Можемо побачити це на рисунку 1.

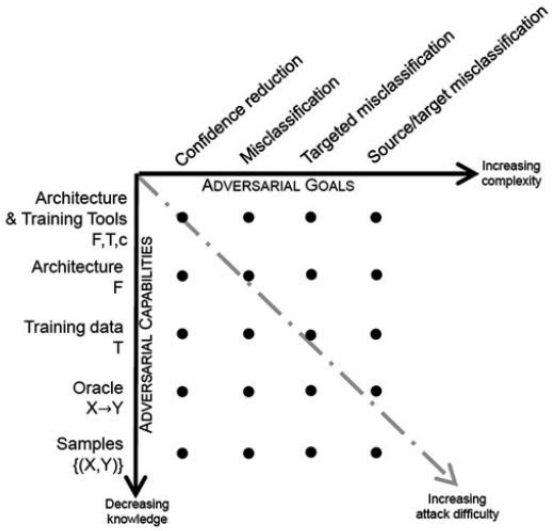

Рисунок 1 Різні типи контроверсійних атак [1]

Отож, можемо виділити такі види атак:

1. **Атака для зниження довіри**: суть полягає у зниженні довіри класифікації до потрібного нападнику значення.

2. **Шкідлива класифікація**: суть полягає у зміні класифікації до іншого (неправильного) класу. Клас на який відбувається зміна не є в цьому варіанті відомим до проведення атаки.

3. **Цільова шкідлива класифікація**: суть полягає у перенаправленні класифікації до деякого потрібного нападнику (неправильного) класу. В цьому варіанті вибір класу відбувається перед атакою та є відомим.

4. **Шкідлива класифікація джерела та цілі**: нападник намагається примусити систему класифікувати певний тип даних до певного потрібного нам класу, відмінного від початкового. Така класифікація є вдосконаленим попереднім варіантом для певного типу даних, а не індивідуального варіанту.

Можливі відомі знання про нейрону мережу F і тренувальні дані T відповідно класифікуються на:

1. Архітектура системи і її тренувальний набір даних: відомо про кількість і типи шарів системи F, а також нейронів, функцій ціни, ваги, функцію втрати. Також маємо доступ до набору даних на якому ця система була натренована (T). Це майже вся інформація що взагалі може бути доступна про мережу і це також означає що такий випадок є малоймовірним для нападника, якщо звісно не тестуються атаки на власній створеній системі.

2. Архітектура системи: відомо про архітектуру системи F і тільки про неї. Досить вагома інформація, яка дає змогу відтворити систему і знайти всю іншу інформацію. При цьому немає набору даних на якому мережа тренувалась, що є досить вагомим мінусом.

3. Тренувальний набір даних: відомо про дані T, на яких ця система тренувалася. Проте, відсутня інформація про архітектуру мережі. У такому

випадку найкращим варіантом є тренування подібної мережі для відтворення справжньої. Це дає змогу знайти іншу інформацію, але вона не завжди буде вірною

4. “Той, хто може повідомити майбутнє”: називається ситуація, коли відомо про вихідні дані мережі для заданих вхідних даних. При цьому знання про архітектуру та тренувальні дані відсутні. В таких випадках рекомендується знайти зв'язок між змінами у даних на вході і виході щоб мати фундамент для створення контроверсійних даних.

5. Деякі пари входу і виходу мережі: у цьому випадку відомо про деякі результати системи, але неможливо користуватися нею і дізнатися про різницю, як у попередньому варіанті.

Зроблені наступні висновки: чим менше відомо про мережу, тим складнішою стає спроба атакувати її. При цьому захист від атак, якщо відомо все про мережу, дозволить захистити її від атак, коли про мережу нічого не відомо. Це поділяє типи атак на відкриті, якщо відомо про архітектуру та параметри системи, та відповідно закриті, якщо такі знання відсутні. Так як такі атаки вже існують деякий час, вже наявно досить багато різних відомих типів атак, які класифікують за параметром відкритості або закритості, а також таким аспектом як спрямована чи неспрямована атака [1].

2. Огляд існуючих контроверсійних атак на нейронні мережі

2.1 Різні типи атак відносно їх типу та спрямованості

Відповідно до роботи Z Yin [2] відомими і найбільш дієвими і популярними є такі види атак:

Таблиця 1 Типи контроверсійних атак на нейронні мережі

Назва атаки	Відкритого / закритого типу	Спрямована / Неспрямована
Підхід L-BFGS, обмежений коробкою [5]	Відкритого типу	Спрямована
Швидкий метод градієнтного знаку [6]	Відкритого типу	Неспрямована
Стандартний ітеративний метод [7]	Відкритого типу	Неспрямована
Ітеративний метод найменш частішого класу [7]	Відкритого типу	Спрямована
Мапова атака відмінності Якоба [1]	Відкритого типу	Спрямована
DeepFool	Відкритого типу	Неспрямована
Атака Карліні і Вагнера	Відкритого типу	Спрямована
Атака на основі відтворення [1]	Закритого типу	Спрямована
Атака Гудінні [8]	Закритого типу	Спрямована

Ці атаки відрізняються функціями створення і інформацією яка необхідна для проведення атаки. Атаки не дозволяють знайти потрібну інформацію про нейронну мережу. Відповідальність і способи знаходження такої інформації покладені на нападника. Для цього використовується такі способи як відтворення мережі, аналіз мережі та інші. Якщо атаки У роботі детальніше розглянуті такі атаки як підхід L-BFGS та ітеративні методи. Проаналізовані методи захисту від подібних атак.

2.2 Підхід L-BFGS, обмежений коробкою

Ця атака була одної з перших, яка була винайдена та проаналізована у роботі Сегеди [5]. У цій праці розглянуто підхід, суть якого полягає у незначних невідчутних змінах до оригінального зображення, метою і наслідками яких є некоректна класифікація мережою.

Формально він описує процес як вираховування наближеного значення c > 0, для якого мінімалізатор r буде задовільняти f(x + r) = 1. Цьому відповідає така оптимізаційна задача:

$$min_r \mathrm{c}|\mathrm{r}| + loss_f(\mathrm{x} + \mathrm{r}, \mathrm{l})\ \mathrm{s.t.}\ \mathrm{x} + \mathrm{r} \in [0,1]^m$$

Рішенням такої проблеми буде наближене значення, за допомогою якого формуються відповідні контроверсійні зображення. У роботі автора вони були протестовані на низці наборів даних, таких як: MNIST, AlexNet, QuocNet. Головною проблемою таких атак для людини є те, що їх важко відрізнити оком чи тоді, коли оригінальне зображення відсутнє. Приклад використання атаки можна побачити на Рисунку 2. Зліва розташоване оригінальне зображення, справа – результат змін, посередині – різниця між зображеннями. [2]

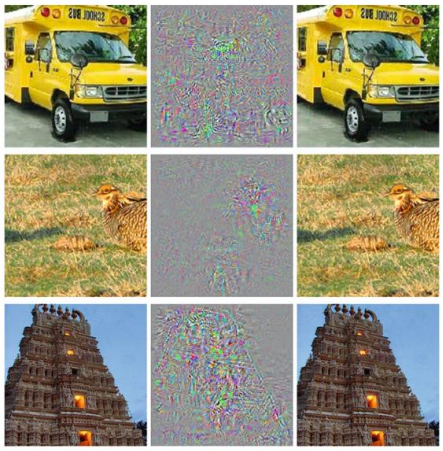

Рисунок 2 Приклад штучно створених екземплярів створених на AlexNet [5]

2.3 Швидкий метод градієнтного знаку

Більш популярним через свою простоту і швидкість виконання є швидкий метод градієнтного знаку. Він був представлений Гудфеллоу у його роботі [6]. Він припускає що наявність лінійної поведінки у багатовимірних просторах дозволяє створити контроверсійні екземпляри. Створення відбувається послідовно і є швидким, бо обрахунок поступових перетворень відбувається у один крок.

Тобто, знаючи ваги W нейронної мережі, розглянемо X як вхід до моделі, Y як класам класифікації, і J(W, x, y) яка є функцією оцінки. Ця функція випрямляється відносно знань W, видаючи оптимально максимальну норму обмежених змін. Цьому відповідає така оптимізаційна задача:

$$\eta = \mathrm{z} \ * \mathrm{sign}(\nabla_x \mathrm{J}(W, \mathrm{x}, \mathrm{y}))$$

У цьому рівнянні, дельта X це градієнт функції ціни з відповідним вхідним значенням x, яке обраховується шляхом зворотнього поширення помилки. Амплітуда змін контролюється параметром z. Чим більше його значення, тим більше шанс надурити мережу. Проте, з більшим значенням, контроверсійний екземляр починає сильно відрізнятися від оригіналу, так що його може помітити людина. [2]

3. Реалізація практичної частини

3.1 Вибір програмного каркасу для створення нейронної мережі

На разі доступна велика кількість програмних каркасів для створення нейронних мереж. Кожен з них має свої переваги і недоліки, які потрібно ретельно дізнатися і завчасно брати до уваги. Одним з таких пунктів є те, що моя робота планувалась виконуватися на власній локальній машині, а не у хмарі. Також важливим пунктом є те, що вибір середовища розробки став Windows, попри наявний і рекомендований для мови Python середовища Linux.

Tensorflow, і вбудований в нього, починаючи з другої версії каркасу, Keras, і PyTorch є двома популярними рішеннями. Основною відмінністю є абстрактний рівень прикладного програмного інтерфейсу між ними. На одному боці Keras, який використовує Tensorflow як двигун, має високий рівень абстракції, що дозволяє описувати і створювати складні нейронні мережі з високою швидкістю розробки і широким функціоналом. На іншому - PyTorch, який має досить низький рівень абстракції, що дозволяє збільшити швидкість виконання, розширює можливість налаштування мережі і додає можливість більш низького зневадження для виявлення помилок на всіх етапах розробки. Проте, низький рівень абстракції збільшує час розробки і забирає можливість швидкої зміни архітектури мережі (доводиться переписувати багато коду).

Таким чином, беручи до уваги те, що набір даних для тренувань не є настільки великим, щоб задумуватися про швидкість виконання, і те, що в дослідницькому середовищі хочеться мати можливість швидко змінити прототип архітектури вибором стало використання Keras на Tensorflow.

3.2 Налаштування середовища для розробки мережі

Для збільшення швидкості тренування мережі краще замість процесора використовувати відеокарту на яку також потрібно встановити додаткові драйвери. Це дозволить оптимізувати різні операції, наприклад множення матриць.

Мовою розробки є Python, для якого створюється віртуальне середовище для ізоляції проектів, в кожному з який може бути різна версія встановлених пакетів. Версія мови, яка була стабільною для програмного каркасу є 3.8, версія tensorflow - 2.4.1, keras – 1.0.6 . У зазначених пакетів є свої залежності, які встановлюються автоматично. Протестувавши просту мережу, надану каркасом keras, видно що драйвери використовуються. Отже, середовище готове до подальших кроків.

3.3 Створення нейронної мережі для проведення атак на неї

Для того, щоб зробити атаку у штучному середовищі знадобиться модель, яка і буде тестовим макетом для експериментів. Вибором сфери для моделі стало розпізнання звуків, а саме: автоматичне розпізнавання мови. Вибір можна аргументувати тим, що моделі для розпізнання мови несуть у собі більше наукової новизни, а також відрізняються архітектурою від моделей для, наприклад, розпізнання і класифікації зображень.

В такому випадку проблему розпізнавання представляють як проблему порівняння послідовностей, де звук представляється у вигляді послідовності векторів функцій, а текст – у вигляді послідовності символів (слів).

Як тренувальний набір даних було вибрано використати LJSpeech з проекту LibriVox [3]. Він складається з 2.6 гігабайтів аудіо у вигляді уривків з 7 популярних книг написаних між 1884 і 1964 роками, які читає один і той

же оратор. Уривки різняться довжиною між 1 до 10 секунд. Загальна довжина даних складає приблизно 24 години. Також до кожного уривку наявна транскрипція у форматі UTF-8. Цих даних вистачить щоб натренувати модель, а також вони будуть використані для створення штучних підробок.

Дані були попередньо оброблені: тренувальний набір складає 99 відсотків, а перевірочний набір відповідно 1 відсоток. Дані для створення штучних підробок будуть взяті з перевірочного набору і будуть представляти 10 відсотків його розміру.

Архітектурою для моделі була вибрана запропонована авторами статті “Attention is all you need” [4]: оригінальний трансформер, який складається з структур кодувальника і декодувальника. Перевагою цієї системи є те, що вона є автоматично регресивною: вона використовує вже оброблені символи як додаткову інформацію при генерації наступних. Це є важливим для сфери розпізнання мови, адже це допомагає мережі зрозуміти, наприклад, що чотири дієслова у ряд є менш випадковим варіантом. Можемо побачити архітектуру на рисунку 3

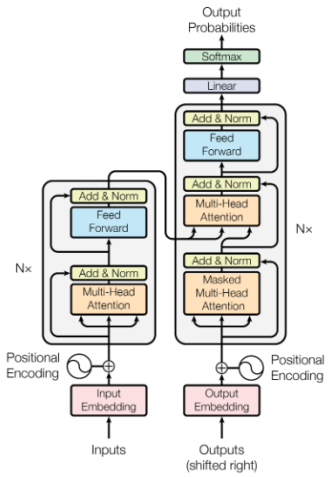

Рисунок 3 - Архітектура моделі трансформера [4]

Також головною відмінністю цієї архітектури є наявність функції уваги, яку можна описати як зв'язок між запитом до моделі і виходом у вигляді мапи ключ-значення, в якому кожне значення є результатом функції сумісності важелів мережі до ключа. У мережі вона застосовується паралельно до матриці послідовності. Цей процес називається багатоголова увага.

$$MultiHead(Q, K, V) = Concat(h_1, \dots, h_n)W^0,$$

$$\text{де } h_i = Attention(QW_i^Q, KW_i^K, VW_i^W)$$

Він є частиною рівнів кодувальника, декодувальника і пари кодувальник-декодувальник. Представленням кодувальника є клас TransformerEncoder представлений на рисунку 7. Представленням декодувальника є клас TransformerDecoder. Ці два класи є, як зазначалось раніше, частиною моделі-трансформера, представленою класом Transformer. Створена модель з такими параметрами: двомастами прихованими нейронами, двома головами, чотирьомастами нейронами прямого поширення, чотирьома рівнями кодувальника і одним рівнем декодувальника. Саме створення можна побачити на рисунку 8.

Також мережа налаштована зберігати ваги епохи, якщо вони краще показали себе на перевірочному наборі даних. Цей функціонал також дозволяє додати такі можливості як зупинка тренування для тестування, відтворення тестування з останньої найкращої епохи.

3.4 Опис процесу тренування нейронної мережі

Модель початково була запущена на сто епох з послідовною зміною швидкості навчання. Запуск можемо побачити на рисунку 4. Після кожної епохи будуть викликатися дві функції: відображення результатів роботи

мережі на випадково взятому наборі у розмірі однієї партії та порівняння результатів з попередньою епохою для збереження у разі меншої розбіжності з результатом.

```
history = model.fit(ds, validation_data=val_ds, callbacks=[display_cb, model_checkpoint_callback], epochs=100)
```

Рисунок 4 - Запуск тренувального процесу

Під час першої епохи, як і очікувалось, видно ще початкові здібності мережі до розпізнання мови. Відслідковується результат у вигляді 1.4014 на тренувальному наборі даних і 1.4323 на перевірочному наборі. Аналіз 203 партій по 64 аудіо кожен зайняв всього 1379 секунд, в середньому по 7 секунд на партію. Вивід можна побачити на рисунку 5.

```
Epoch 00001: val_loss improved from inf to 1.66838, saving model to best_model.hdf5
Epoch 1/100
203/203 [==============================] - 1379s 7s/step - loss: 1.4014 - val_loss: 1.4323
target:     <the increased information supplied by other agencies will be wasted.>
prediction: <the the the the the the te the the the the te the the the the the the the the the the the the te

target:     <prs must develop the capacity to classify its subjects on a more sophisticated basis than the present
prediction: <the the the the the the the the t the the the the the are te ale the the the the te the the the the t

target:     <its present manual filing system is obsolete#>
prediction: <the the the the the the te the the the the te the the the the the the the the the the the the the te
```

Рисунок 5 - Результати першої епохи

Пропустимо декілька епох, поки не буде видно поліпшень у результатах. Таку різницю можна побачити, наприклад на 40 епосі. Результат метрики змінився до 0.4495 на тренувальному наборі і до 0.58219 на перевірочному. Обробка партії зайняла менше часу: 703 секунди. На наступній епосі бачимо зменшення метрики до 0.4438, проте вона гірше показала себе на перевірочному наборі, тому не була збережена. Деякі слова

розпізнаються системою, проте вона ще потребує тренування. Вивід можна побачити на рисунку 5.

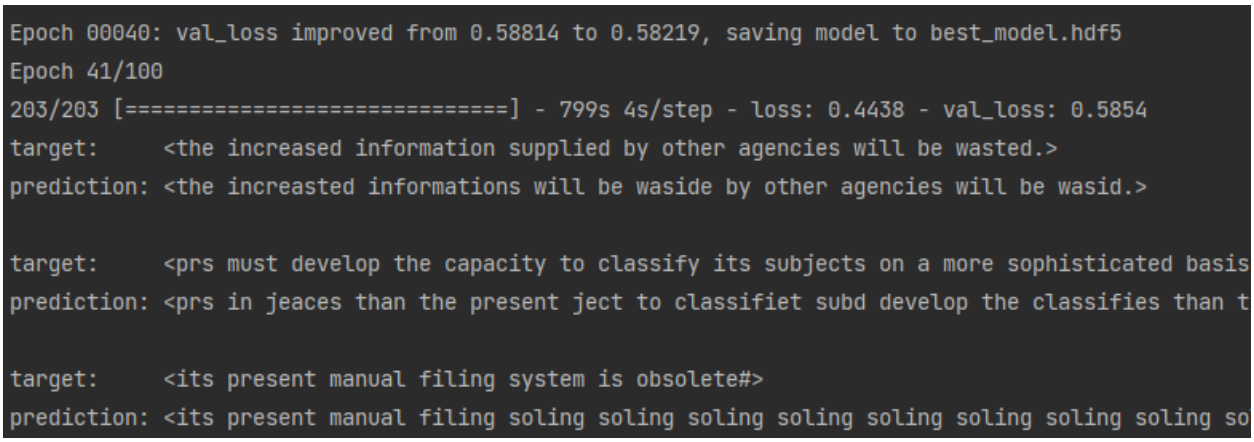

Рисунок 6 - Результати сорок першої епохи

Після шістдесятої епохи, результати перестали поліпшуватися, зупинившись на відмітці у **0.5284**. Причиною цього може бути те, що у більш складних системах автоматичного розпізнання голосу для кожної з характеристик мови присутня окрема нейронна мережа. Наприклад, система може складатися з нейронної мережі для розпізнавання вимови та акценту, мережі для прибирання шумів та мережі для розпізнання самої мови. Також такі системи мають більші за довжиною тренувальні набори даних. Їх загальна довжина може бути більше дванадцяти тисяч годин, в той час коли система представлена у цій роботі була тренована тільки на наборі даних довжиною у двадцять чотири години.

Проте, метою роботи не є створення ідеальної системи розпізнання мови, а атаки на такі системи. Тому якщо атаки спрацюють на системі яка має такі результати, то вона також буде працювати на більш натренованих системах.

3.5 Проведення атаки на створену нейронну мережу

Для проведення атаки були вибрані способи описані у теоретичній частині роботи: підхід L-BFGS, обмежений коробкою та швидкий метод градієнтного знаку. Такі атаки є відкритого типу, тобто відома інформація про архітектуру нейронної мережі. Перший тип атаки є спрямованим, а другий – неспрямованим.

Створення екземплярів аудіо відрізняється від створення екземплярів зображень. Метою є створення такого екземпляру аудіо, що нейронна мережа зробить помилку у її розпізнанні. Метод маніпуляції і зміни аудіо базується на статті [9].Для проведення атак був створений окремий клас, в якому були описані методи для створення екземплярів. Модель при тренуванні зберігалась у окремий файл, звідки була отримана вся потрібна інформація для атак, така як: функція втрати, ваги мережі.

Для екземплярів були взяті дані з перевірочного набору, при створені проводилося самостійне тестування для налаштування параметрів, які впливають на амплітуду змін. Доведено до моменту, коли зміни для вуха людини не є сильно відчутними.

При тестуванні на мережі таких екземплярів виявлено різку зміну результатів в сторону погіршення. Потрібно зазначити, що якщо мережа була б більш натренована, можливо що атака була б менш дієвою.

Висновки

Представлено порівняння різних каскадів для створення нейронних мереж. Налаштовано середовище для створення нейронних мереж. Проаналізовано і створено нейронну мережу автоматичного розпізнання голосу. Натреновано мережу до максимального результату впродовж ста епох. На створену мережу виконано контроверсійні атаки екземплярами, які були створені на основі перевірочного набору даних. Зроблені висновки, що такі атаки є справді вразливим місцем у нейронних мережах. Проведений аналіз можливих захистів від атак: потрібно повторно тренувати нейронну мережу на наборі даних з наявними контроверсійними екземплярами. Доступні також інші способи захисту від подібних атак, проте вони не були розглянуті в цій роботі. Такими є, наприклад, спосіб теорії ігор або захисна генеративна нейронна мережа. Це може бути темою дослідження нових робіт.

Список використаних джерел

1. Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, Swami A (2016) The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security andPrivacy (EuroSP), pp 372–387 https://arxiv.org/abs/1511.07528

2. Mingjian Tang, Mamoun Alazab (2019) Deep Learning Applications for Cyber Security URL https://www.researchgate.net/publication/331247533_Deep_Learning_Applications_for_Cyber_Security

3. LJ Speech Dataset 2016-2017 URL: https://keithito.com/LJ-Speech-Dataset/

4. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin (2017) Attention is all you need https://arxiv.org/abs/1706.03762

5. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow IJ, Fergus R (2013) Intriguing properties of neural networks. CoRR, https://arxiv.org/abs/1312.6199

6. Goodfellow I, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples. In:International Conference on Learning Representations https://arxiv.org/abs/1412.6572

7. Kurakin A, Goodfellow IJ, Bengio S (2016) Adversarial examples in the physical world. CoRR,abs/1607.02533 https://arxiv.org/abs/1607.02533

8. Cisse M, Adi Y, Neverova N, Keshet J (2017) Houdini: Fooling Deep Structured Prediction Models https://arxiv.org/abs/1707.05373

9. Paarth Neekhara, Shehzeen Hussain, Prakhar Pandey, Shlomo Dubnov, Julian McAuley, Farinaz Koushanfar 2019 Universal Adversarial Perturbations for Speech Recognition Systems https://arxiv.org/abs/1905.03828

Додатки

```python
class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, num_heads, feed_forward_dim, rate=0.1):
        super().__init__()
        self.att = layers.MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.ffn = keras.Sequential(
            [
                layers.Dense(feed_forward_dim, activation="relu"),
                layers.Dense(embed_dim),
            ]
        )
        self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = layers.Dropout(rate)
        self.dropout2 = layers.Dropout(rate)

    def call(self, inputs, training):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
```

Рисунок 7 - клас кодувальника моделі трансформера

```python
model = Transformer(
    num_hid=200,
    num_head=2,
    num_feed_forward=400,
    target_maxlen=max_target_len,
    num_layers_enc=4,
    num_layers_dec=1,
    num_classes=34,
)
```

Рисунок 8 - створення моделі нейронної мереж