ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА
Факультет прикладної математики та інформатики
Кафедра обчислювальної математики

Курсова робота

АНАЛІЗ АТАК НА ЛІНІЙНІ МОДЕЛІ МАШИННОГО НАВЧАННЯ

Виконав: студент III курсу групи ПМп-31
напрямку підготовки (спеціальності)
113 – ”Прикладна математика”
Середович В.В.
(прізвище та ініціали)
Керівник: Музичук Ю.А.
(прізвище та ініціали)

Львів — 2020

Зміст

Вступ

Алгоритми машинного навчання активно використовується у різних областях нашого життя та дуже добре демонструють себе у задачах класифікації. Однак, виявляється, що ці алгоритми є досить вразливими навіть до незначних, правильно підібраних, пертурбацій вхідної інформації. Такі модифіковані приклади, які навмисно змінюють для того, щоб вони були хибно передбаченні називають змагальними прикладами або adversarial samples. В деяких випадках ці зміни можуть бути настільки малі, що для людини вони зовсім не будуть помітні, однак класифікатор моделі буде робити хибне передбачення. Така вразливість може бути дуже небезпечною коли алгоритми машинного навчання застосовуються в критичних для здоров'я людини задачах, таких як передбачення небезпечного захворювання або розпізнавання знаків дорожнього руху автопілотом машини. Саме тому, тема вразливостей алгоритмів машинного навчання та відповідних методів захисту які могли б запобігти небажаним наслідкам, привертає до себе багато уваги і потребує досліджень.

В межах теми цієї роботи будемо розглядати декілька різних методів атак на лінійні моделі машинного навчання, які аналізуватимемо та порівнюватимемо між собою. Окрім того, також реалізуємо і перевіримо декілька методів захисту для того щоб визначити їх ефективність у захисті моделі.

Розділ 1

Постановка задачі

Метою даної роботи є дослідження ефективності різних методів атак на лінійні моделі машинного навчання, та аналіз можливих методів захисту від них.

Виходячи з мети, визначеними завданнями роботи є:

- Реалізувати лінійну модель машинного навчання
- Розглянути різні методи генерування змагальних прикладів
- Застосувати атаки на створену модель та проаналізувати їх ефективність
- Розглянути можливі методи захисту від атак

1.1 Основні поняття

В цій секції будуть розглянуті основні терміни, які будуть використовуватись в ході роботи.

Змагальний приклад (Adversarial sample)
Нехай існує класифікатор $f(x) : x \to y$, де $x \in X, y \in Y$, який передбачає значення y для вхідного x. Метою змагального прикладу є знайти такий x^*, який знаходиться в околі x, але хибно визначається класифікатором. Зазвичай максимальний рівень шуму в змагальному прикладі може бути не більше за певну L_p норму $\|x^* - x\|_p < \varepsilon$, де $p = 1, 2, \infty$. В межах даної роботи для визначення рівня пертурбації буде використовуватись саме L_∞ норма.

Цілеспрямовані атаки (Targeted Attacks)

Цілеспрямованими атаками є атаки, метою яких є внести в приклад x, який правильно передбачається як $f(x) = y$, незначний шум так, щоб класифікатор зробив хибне передбачення $f(x^*) \neq y$.

Нецілеспрямовані атаки (Untargeted Attacks)

Нецілеспрямованими атаками називають такі атаки, метою яких є внести у вхідний приклад x такі пертурбації, щоб класифікатор передбачив якийсь конкретний клас $f(x^*) = y^*$, де y^* є заданою ціллю $y^* \neq y$.

Атаки на відкриту модель (White Box attacks)

Атаки на закриту модель описують такий сценарій, коли в нападника є повний доступ до внутрішніх параметрів моделі таких як ваги і параметр зсуву.

Атаки на закриту модель (Black Box attacks)

До атак на відкриту модель відносять атаки, при яких в нападника нема доступу до внутрішніх параметрів моделі і для створення успішного нападу він може використовувати тільки передбачення моделі.

Розділ 2

Модель

2.1 Мультикласова логістична регресія

Лінійним методом машинного навчання, на котрий будемо здійснювати атаки, буде використовуватись модифікований алгоритм логістичної регресії для мультикласової класифікації.

Нехай маємо набір тренувальних даних:

$$(x^{(1)}, y^{(1)}), \quad (x^{(2)}, y^{(2)}), \quad ... \quad , (x^{(m)}, y^{(m)})$$

$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \quad x \in \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}$$

$$Y = \begin{bmatrix} \vdots & \vdots & & \vdots \\ y^{(1)} & y^{(2)} & \dots & y^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \quad y \in \begin{bmatrix} y_1 \\ \dots \\ y_C \end{bmatrix} \quad y_1, ..., y_C$$

X - матриця в якої кожен стовпець є набором характеристик $i-$ого прикладу, $i = 1, \dots, m$

Y - матриця класів в якої кожен стовпець це масив розмірності C, з одиницею на місті справжнього класу

m - кількість прикладів

n - кількість характеристик в кожному прикладі

C - кількість класів

Задача прикладу $x \in R^n$, знайти $\hat{y} = P(y = 1 \mid x), \quad 0 \leq \hat{y} \leq 1$

$$\hat{y} = softmax(\omega^T x + b), \quad \text{де } \omega \in R^n, b \in R \text{ - невідомі параметри}$$

Функція активації матиме вигляд:

$$softmax(z) = \frac{e^z}{\sum_{k=1}^{C} e_k^z} \tag{2.1}$$

Для кожного прикладу з тренувального датасету потрібно обчислити:

$$z^{(i)} = \omega^T x^{(i)} + b, \text{ де } \hat{y}^{(i)} = softmax(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{C} e_k^z}, \text{ так щоб } \hat{y}^{(i)} \approx y^{(i)}$$

В якості функції втрати буде використовуватись функція кросс-ентропії:

$$\xi(y^{(i)}, x^{(i)}) = -\sum_{j=1}^{C} y_j \log(\hat{y_j}) \tag{2.2}$$

$$\xi(Y, X) = \frac{1}{m} \sum_{i=1}^{m} \xi(y^{(i)}, x^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{C} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right) = J(\omega, b) \tag{2.3}$$

Задача полягає в тому щоб знайти параметри $w \in R_x^n, b \in R$ що мінімізують функцію $J(\omega, b)$. Для цього будемо використовувати алгоритм градієнтного спуску який буде виглядати наступним чином 1.

Algorithm 1 Градієнтний спуск

Input: Тренувальні дані X, Y, гіперпараметри: N, ε, α
Output: Оптимальні параметри моделі ω, b
i = 0
while i $< N$ або ($\|d\omega\| < \varepsilon$ та $\|db\| < \varepsilon$) **do**
    $d\omega = \alpha \frac{\partial J(\omega, b)}{\partial \omega}$
    $db = \alpha \frac{\partial J(\omega, b)}{\partial b}$
    $\omega = \omega - d\omega$
    $b = b - db$
    i += 1
end while
return ω, b.

2.2 Датасет

В якості тренувального датасету будемо використовувати MNIST базу даних яка складається з 60 тис. тренувальних та 10 тис. тестувальних зображень рукописних цифр 2.1. Розмір кожного із них складає 28×28, а значення їх пікселів знаходяться в проміжку $[0, 255]$. На основі неї будемо здійснюватись тренування моделі і аналіз методів атак та захисту.

Рис. 2.1: датасет рукописних цифр MNIST

2.3 Тренування моделі

У створені ефективної моделі машинного навчання велику роль грає не тільки обраний алгоритм, але й правильна обробка даних та використання методів оптимізації. Тому під час реалізації мультикласової логістичної регресії, також були застосовані декілька алгоритмів оптимізації.

2.3.1 Обробка даних

Значення зображень датасету MINIST знаходяться в проміжку $[0, 255]$, а для того щоб процес навчання відбувався швидше, необхідно щоб всі дані перебували в деякому малому проміжку, наприклад, такому як $[0, 1]$. Для цього можна скористатись методом нормалізації або стандартизації даних, але через те, що всі оригінальні значення знаходяться у фіксованому проміжку, найпростішим підходом буде просто розділити всі значення на максимальне число, тобто на 255.

2.3.2 Оптимізація моделі

Mini batch gradient descent

Для покращення роботи моделі будемо використовувати градієнтний спуск з mini batch оптимізацією. Ідея даної оптимізації полягає в тому, щоб розбити весь тренувальний датасет на деяку кількість частин розміру k ($k = 2^q, q \in \mathbb{N}$)

$$X = [X^1, X^2, ..., X^{|m/k|}]$$
$$Y = [Y^1, Y^2, ..., Y^{|m/k|}]$$

Таким чином, ми будемо оновлювати параметри моделі ω і b після обчислення функції втрати на кожній з цих частин, а не на всіх прикладах одразу. Перевага цього методу в тому, що він дає можливість за меншу кількість ітерацій досягти збіжності. Також його варто використовувати для того щоб зменшити обчислювальні ресурси необхідні для тренування моделі.

Momentum

Ще одним ефективним алгоритмом оптимізації є momentum оптимізація яка базується на методі рухомого середнього значення. Тепер під час оновлення ваг і зсуву моделі ми будемо враховувати і всі попередні обчислені градієнти теж, з деяким коефіцієнтом згладжування μ. Оновлення параметрів моделі тепер буде виглядати наступним чином:

$$V_{d\omega} = \mu V_{d\omega} + (1 - \mu)d\omega; \quad \omega = \omega - \alpha V_{d\omega};$$
$$V_{db} = \mu V_{db} + (1 - \mu)db; \quad b = b - \alpha V_{db};$$

В результаті застосування цього методу, кроки градієнтного спуску будуть згладжені, що допоможе під час тренування уникати потрапляння моделі у точки локального оптимума.

2.3.3 Тренування

Для знаходження оптимальних гіперпараметрів моделі був використаний алгоритм перебору параметрів з заданої множини. За результатами перевірки на тестовому датасеті точність отриманої моделі складає $\sim 92\%$.

Розділ 3

Методи атак

В цьому розділі будуть розглянуті деякі методи генерування змагальних зразків, які були реалізовані і протестовані на лінійній моделі натренованій в розділі 2.3.

Далі, під час опису будуть використовуватись такі позначення:

- $\boldsymbol{X}$ - це зображення, зазвичай це 3-D тензор (ширина $\times$ висота $\times$ глибина). В нашому випадку має розміри $(28 \times 28 \times 1)$ пікселів, яке складається з додатних цілих чисел в межах $[0, 255]$.
- $\boldsymbol{X}^*$ - згенерований змагальний приклад для зображення $\boldsymbol{X}$
- y_{true} - правильний клас для зображення $\boldsymbol{X}$.
- $J(\boldsymbol{X}, y)$ - штрафна функція моделі. Ваги моделі ω навмисно не будемо вказувати, вважатимемо їх сталими, як значення які будуть отримані з моделі машинного навчання. В нашому випадку штрафною функцією є функція кросс-ентропії 2.2.
- $Clip_{\boldsymbol{X},\varepsilon}\{\boldsymbol{X}^*\}$ - функція яка виконує по-піксельне обрізання зображення X^* так, щоб результат був в L_∞ ε - околі вхідного зображення $\boldsymbol{X}$. Рівняння для цього виглядає наступним чином:

$$Clip_{X,\varepsilon}\{\boldsymbol{X}^*\}(x,y,z) = min\Big\{255, \boldsymbol{X}(x,y,z) + \varepsilon, max\{0, \boldsymbol{X}(x,y,z) - \varepsilon, \boldsymbol{X}^*(x,y,z)\}\Big\}, \tag{3.1}$$

де $\boldsymbol{X}(x,y,z)$ - це значення каналу z зображення $\boldsymbol{X}$ з координатами (x,y).

3.1 FGSM

Одним з найпоширеніших методів для генерування змагальних зразків машинного навчання є Fast Gradient Sign Method. Вперше він був представлений в роботі Goodfellow, Shlens та Szegedy [1]. Ідея цього методу полягає в тому, щоб знайти такий змагальний приклад x^* який максимізує функцію втрати $J(x^*, y)$ до певного L_∞ обмеження. Цього можна досягти один раз використавши зворотне поширення:

$$X^* = X + \varepsilon \cdot sign(\Delta_x J(x, y)), \tag{3.2}$$

де ε - деякий гіперпараметер.

Для того щоб застосувати цей метод для логістичної регресії необхідно обчислити градієнт від функції кросс-ентропії 2.2. Для цього спочатку знайдемо похідну від $softmax$ функції:

$$z_i = \omega^T x^{(i)} + b; \quad \hat{y}_i = a_i = softmax(z_i); \quad softmax(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{C} e_k^z};$$

Якщо $i = j$,

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{\partial \frac{e^{z_i}}{\sum_{k=1}^{C} e_k^z}}{\partial z_j} = \frac{e^{z_i} \sum_{k=1}^{C} e^{z_k} - e^{z_j} e^{z_i}}{(\sum_{k=1}^{C} e^{z_k})^2} = \tag{3.3}$$

$$= \frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}} \times \frac{(\sum_{k=1}^{C} e^{z_k} - e^{z_j})}{\sum_{k=1}^{C} e^{z_k}} = y_i(1 - y_j) \tag{3.4}$$

Якщо $i \neq j$,

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{\partial \frac{e^{z_i}}{\sum_{k=1}^{C} e_k^z}}{\partial z_j} = \frac{0 - e^{z_j} e^{z_i}}{\sum_{k=1}^{C} e^{z_k}} \times \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}} = -y_i y_j \tag{3.5}$$

Далі обчислюємо похідну від функції кросс-ентропії:

$$\xi(y, x) = -\sum_{i=1}^{C} y_j \log(\hat{y}_j)$$

$$\frac{\partial \xi}{\partial z_i} = -\sum_{j=1}^{C} \frac{\partial y_j \log(\hat{y}_j)}{\partial z_i} = -\sum_{j=1}^{C} y_j \frac{\partial \log(\hat{y}_j)}{\partial z_i} = -\sum_{j=1}^{C} y_j \frac{1}{\hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_i} =$$

$$= -\frac{y_i}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i} - \sum_{j \neq i} \frac{y_j}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_i} = -\frac{y_i}{\hat{y}_i}\hat{y}_i(1-\hat{y}_i) - \sum_{j \neq i} \frac{y_j}{\hat{y}_j}(-\hat{y}_i\hat{y}_j) =$$

$$= -y_i + y_i\hat{y}_i + \sum_{j \neq i} y_j\hat{y}_i = -y_i + \sum_{j=1}^{C} y_j\hat{y}_i = \hat{y}_i - y_i,$$

де i = 1, ..., C

Тепер можна обчислити градієнт від штрафної функції по x:

$$\frac{\partial \xi}{\partial x_i} = \frac{\partial \xi}{\partial z_i}\frac{\partial z_i}{\partial x_i} = (\hat{y}_i - y_i)\frac{\partial(\omega^T x_i + b)}{\partial x_i} = (\hat{y}_i - y_i)\omega^T = dz \cdot \omega^T \tag{3.6}$$

Маючи всі наведені обчислення, можна записати алгоритм знаходження змагального прикладу 2.

Algorithm 2 $FGSM$

1: **Input:** Приклад x, клас y_{true}, класифікатор f, параметри: ω, b.
2: **Input:** значення пертурбації ε.
3: **Output:** Adversarial x^* з нормою $\|x^* - x\|_\infty \leq \varepsilon$;
4: $z = \omega^T x + b$;
5: $\hat{y} = softmax(z)$;
6: $\delta = (\hat{y} - y_{true}) \cdot \omega^T$;
7: $x^* = Clip_{X,\varepsilon}\{x + \alpha \cdot sign(\delta)\}$;
8: **return** $x^* = x^*_T$.

3.2 I-FGSM

Даний метод є модифікацією попереднього, особливість якого полягає в тому, що для знаходження змагального прикладу буде використовуватись ітераційний підхід. Ми застосовуємо звичайний метод декілька разів з певним невеликим кроком α і після кожного кроку використовуємо функцію $clip$ 3.1, для того щоб переконатись що значення знаходяться в ε-околі оригінального зображення.

Ітераційний алгоритм швидкого градієнтного спуску буде мати вигляд 3.

Результат роботи алгоритму для $\varepsilon = 0.05$ можна побачити на рисунку 3.1.

Algorithm 3 $I - FGSM$

Input: Приклад x, значення класу y_{true}, класифікатор f
Input: значення пертурбації ε, кількість ітерації T.
Output: Adversarial x^* з нормою $\|x^* - x\|_\infty \leq \varepsilon$;
$\alpha = \varepsilon / T$;
$x^* = x$;
for $t = 0$ *to* $T - 1$ **do**
    $x^* = Clip_{X,\varepsilon}\{x^* + \alpha \cdot sign(\Delta_x J(x, y))\}$;
end for
return $x^* = x^*_T$.

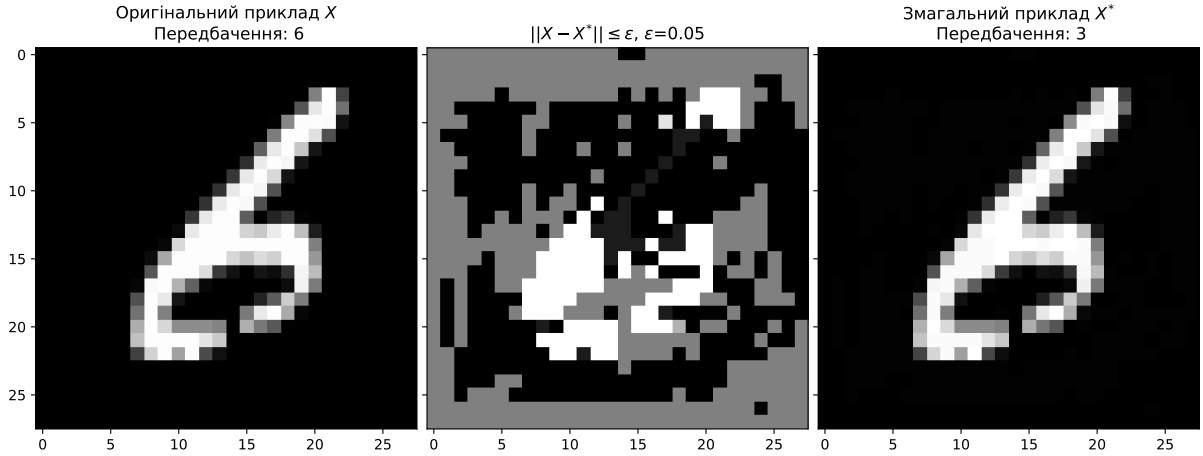

Рис. 3.1

3.3 TI-FGSM

TI-FGSM або як його ще називаюсь Projected Gradient Descent є цілеспрямованою версією попереднього алгоритму, а отже він дає можливість обрати клас який має передбачити модель. Два попередні методи описані вище намагалися максимізувати функцію втрати для правильного класу, не вказуючи яким буде неправильний клас. Цей метод - навпаки, мінімізує функцію втрати для деякого заданого неправильного класу. На сам алгоритм це вплине так, що тепер під час градієнтного спуску ми будемо рухатись в напрямку, протилежному градієнту штрафної функції, тобто на етапі додавання пертурбацій у вхідний приклад, знак буде змінений на протилежний.

3.4 MI-FGSM

Попередні розглянуті методи хоч і є ефективними, але мають певні недоліки. Для прикладу однокроковий (FGSM) обчислює градієнт тільки один раз, припускаючи, що межа рішень є лінійною. На практиці це зазвичай не так і тому для змагального прикладу, згенерованого таким чином, характерне *недотренування* що обмежує ефективність атаки. З іншого боку ітераційний метод (I-FGSM), з кожною ітерацією жадібно рухає змагальний приклад в напрямку градієнту, що може призвести до зупинки в локальній точці оптимума та *перетренуванню* моделі. Щоб розв'язувати цю проблему в роботі Dong та ін. [4] була представлена momentum оптимізація ітераційного методу. Вона допомагає стабілізувати напрямок зміни прикладу що допомагає уникнути потрапляння в локальному оптимумі.

Алгоритм даної модифікації має наступний вигляд 4

Algorithm 4 $MI-FGSM$

Input: Приклад x, значення цілі y, класифікатор f.;
Input: значення пертурбації ε, значення цілі y, кількість ітерації T.;
Output: Adversarial x^* з нормою $\|x^* - x\|_{inf} \leq \varepsilon$;
$a = \varepsilon / T$;
$g0 = 0$; $x_0^* = x$;
for $t = 0 \quad to \quad T - 1$ **do**
    $g_{t+1} = \mu \cdot g_t + \frac{\Delta_x J(x^*, y)}{\|\Delta_x J(x^*, y)\|_1}$;
    $x^* = Clip_{X,\varepsilon}\{x_t^* + \alpha \cdot sign(g_{t+1})\}$;
end for
return $x^* = x_T^*$.

3.5 DeepFool

DeepFool була вперше представлена в роботі Moosavi-Dezfooli, Fawzi та Frossard [2]. На відмінно від FGSM методу цей алгоритм не можна використовувати для цілеспрямовиних атак, де є можливість вибрати клас який має передбачити модель. Отже, його можна використовувати тільки для генерування прикладів які будуть класифікуватись хибно. Основною метою цього методу є знаходження мінімального необхідного значення пертурбації, для того, щоб модель хибно передбачила клас прикладу.

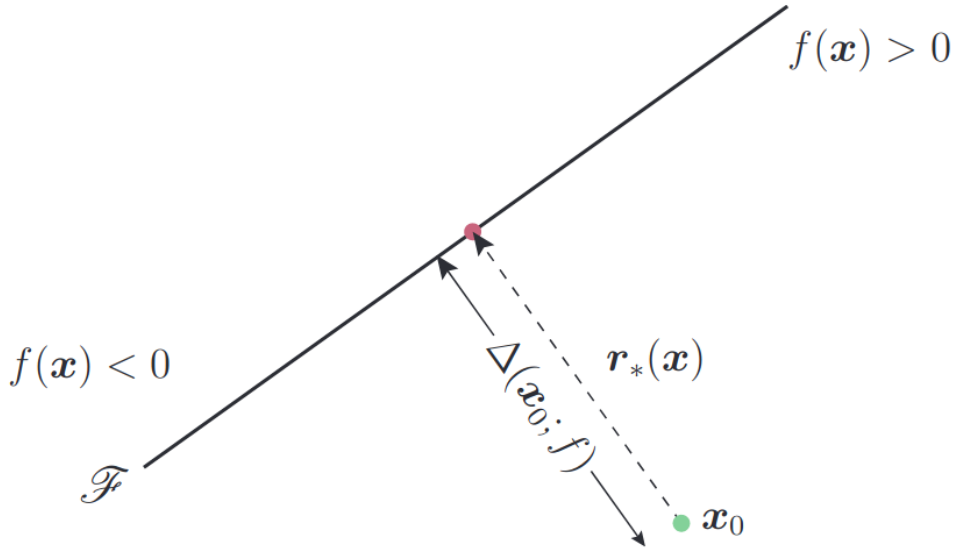

Рис. 3.2: Приклад для лінійного бінарного класифікатора [2]

Для випадку бінарної класифікації легко бачити, що надійність моделі f в точці x_0, $\Delta(x_0; f)$, дорівнює відстані від x_0 до площини гіперпараметра $\mathscr{F} = \{x : \omega^T x + b = 0\}$, яка розділяє два класи (Рис. 3.2). Таким чином, мінімальний зсув, необхідний для зміни рішення класифікатора відповідає ортогональній проєкції x_0 на $\mathscr{F}$. Це можна записати у вигляді формули:

$$r_*(x_0) := -\frac{f(x_0)}{\|\omega\|_2^2}\omega. \tag{3.7}$$

Для того що реалізувати цей алгоритм необхідно спочатку знайти градієнт від функції кросс-ентропії 2.2 по x.

$$\frac{\partial \hat{y}_i}{\partial x_k} = \begin{cases} y_i(1-y_j) & \text{, якщо } i = j \\ -y_i y_j & \text{, якщо } i \neq j \end{cases} \text{, де} \quad i = 1,..,C;\ j = 1,..,C; \tag{3.8}$$

При знаходженні похідної нас цікавить випадок $i = j$, тому отримуємо:

$$\frac{\partial \hat{y}_i}{\partial x_k} = \frac{\partial \hat{y}_i}{\partial z_j}\frac{\partial z_j}{\partial x_k} = y_i(1-y_i)\frac{\partial(\omega^T x_k + b)}{\partial x_k} = y_i(1-y_i)\omega^T = dz \cdot \omega^T$$

Для випадку l_2 норми DeepFool алгоритм для мультикласового випадку буде мати наступний вигляд 5.

Algorithm 5 DeepFool: мультикласовий випадок

Input: Приклад x, значення цілі y, класифікатор f.;
Input: значення цілі y, кількість ітерації T.;
Output: Adversarial x^* з нормою $\|x^* - x\|_2 \leq \varepsilon$;
$x_0 = x$; $i = 0$;
while $\hat{f}(x_i) = \hat{f}(x_0)$ **do**
    for $k \neq \hat{f}(x_0)$ **do**
        $w'_k = \Delta f_k(x_i) - \Delta f_{\hat{k}(x_0)}(x_i)$;
        $f'_k = \Delta f_k(x_i) - \Delta f_{\hat{k}(x_0)}(x_i)$;
    end for
    $\hat{l} = \arg\min_{k \neq \hat{f}(x_0)} \frac{|f'_k|}{\|w'_k\|_2}$;
    $r_i = \frac{\left|f'_{\hat{l}}\right|}{\|w'_{\hat{l}}\|_2^2} w'_{\hat{l}}$;
    $x_{i+1} = x_i + r_i$;
    $i = i + 1$;
end while
return $\hat{r} = \sum_i r_i$.

Результат роботи алгоритму для $\varepsilon = 0.05$ можна побачити на рисунку 3.1.

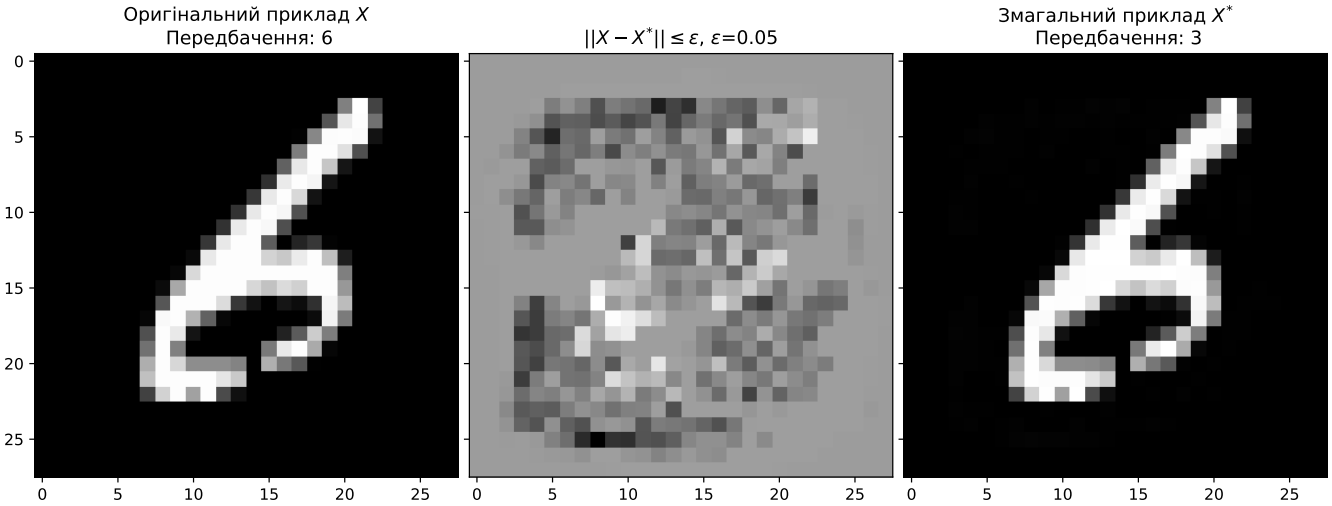

Рис. 3.3

Розділ 4

Методи захисту

Маючи класифікатор моделі F і деяке зображення $\hat{X}$, яке може бути або оригінальним зображенням X або змагальним X^*, метою захисту моделі є або посилити F як F' таким чином, щоб $F(\hat{X}) = F(X)$, або трансформувати $\hat{X}$ через деякий метод T так, щоб $F(T(\hat{X})) = F(X)$.

4.1 Randomization

Одним методів захисту, який добре продемонстрував себе в ході змагань за найкращих метод атак та захисту 2018 року [6] є рандомізація вхідних даних.

Зазвичай пертурбації, які генеруються в наслідок ітераційних атак, можуть легко опинитись перетренованими для деяких параметрів моделі та є унікальними для них. Ця властивість дає можливість зробити висновок, що трансформації додані до вхідних даних, такі як зміна розміру, стискання, або додавання відступів до зображення може зруйнувати структуру певної змагальної пертурбації і таким чином стати непоганим захистом. Якщо трансформації відбуваються випадково і нападник не знає їх характер, тоді цей метод може стати ефективним як для атак на закриті моделі, так і на відкриті теж. Алгоритм для даного методу буде мати наступний вигляд:

1. Спочатку оригінальне зображення x розмірами $W \times H \times 1$ замінюють на нове x' з випадково вибраними розмірами $W' \times H' \times 1$. Ватро зауважити, що $|W' - W|$ та $|H' - H|$ мають бути у відносно малому проміжку, інакше точність моделі на чистих зображеннях суттєво впаде.

2. Другим кроком буде випадкове наповнення деякого простору навколо зображення x' нулями, після чого буде утворене нове зображення x'' з розмірами $W'' \times H'' \times 1$. Таким чином ми можемо наповнити зображення w нульовими пікселями зліва, $W'' - W' - w$ нульовими пікселями справа, h зверху та $H'' - H' - h$ зліва. Це в результаті дає нам $(W'' - W' + 1) \times (H'' - H' + 1)$ різних варіацій.

3. Після двох попередніх етапів, зображення x'' передається оригінальному класифікатору натренованої моделі.

4.2 Pixel Reflection

Ще одним методом, який за допомогою внесення деякого шуму в зображення намагається зруйнувати його змагальну структуру є Pixel Reflection. Він був представлений в роботі Prakash та ін. [8] і добре продемонстрував себе в захисті нейронних мереж. Ідея методу полягає в тому, щоб випадково вибрати піксель із зображення і потім, так само випадково замінити його на інший піксель в його малому околі.

Для того щоб покращити цей алгоритм можна визначити найбільш важливі для класифікації класів місця зображення і уникати їх деформації, знижуючи ймовірність внесення туди змін. Для цього можна скористатись активаційною картою класів моделі. Для випадку нашої моделі вони будуть мати вигляд 4.1.

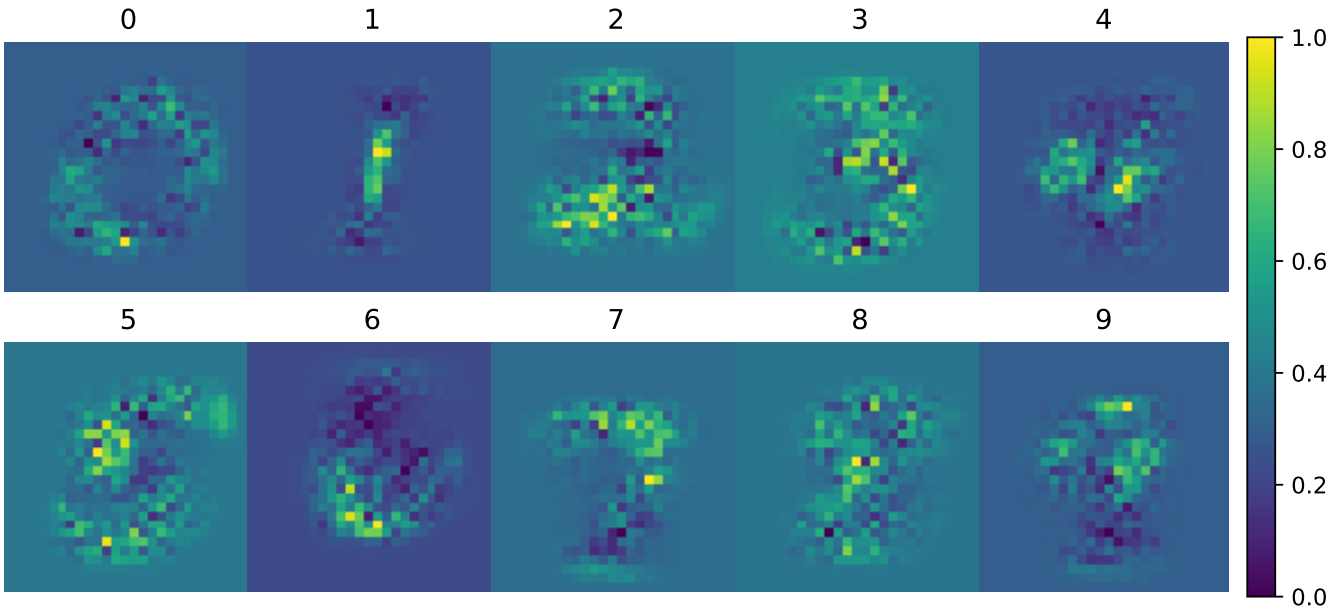

Рис. 4.1

Загалом, для захисту конкретного прикладу нас цікавить карта класу, якому модель надає найвищу ймовірність. Однак, враховуючи те, що серед вхідної інформації будуть потрапляти змагальні приклади, клас який буде визначатись як найбільш ймовірний, скоріше за все буде неправильним. На щастя, змагальний приклад який змінює найімовірніший клас, зазвичай майже ніяк не впливає на всі інші. За результатами експериментів, можна сказати, що в переважній більшості випадків модель визначає змагальний приклад другим за ймовірністю класом для оригінального зображення. Тому для того щоб уникати пошкоджень зображення в найважливіших для класифікації місцях, будемо використовувати середнє значення активаційних карт для двох найімовірніших класів.

Розділ 5

Аналіз алгоритмів

Для побудови графіків з високою точністю, але оптимально з точки зору використання обчислювальних ресурсів, для отримання результатів по роботі алгоритмів атак та захисту використовувались 1 тис. зображень з тестового датасету, які були нормалізовані і тому складались зі значень в проміжку $[0, 1]$. Критерієм деформації прикладу буде використовуватись норма L_∞ яка має бути меншою за максимальне значення пертурбації ε. Експерименти проводились для деякої кількості рівновіддалених значень ε:

$$\varepsilon_i = a + i \cdot h, \quad h = \frac{b-a}{n}; \quad i = 0, .., n; \quad n = 30$$

5.1 Аналіз атак

Під час побудування графіку залежності успішності атак від значення пертурбації 5.1, використовувались тільки такі тестові приклади, які коректно передбачались класифікатором f. У випадку графіка точності моделі 5.2 брались випадково обрані приклади, серед яких були і ті, що визначались класифікатором хибно.

Як можна побачити з графіків 5.1 та 5.2 класичний FGSM алгоритм відстає за ефективністю від його ітераційних модифікацій. В той час як I-FGSM та MI-FGSM методи для даної моделі та датасету дають майже ідентичні результати. Попри те, що DeepFool алгоритм націлений на утворення змагальних прикладів з мінімальним рівнем деформації оригінального зображення, він опинився найслабкішим з точки зору L_∞ норми. Однак зазвичай, приклади утворені за допомогою цього алгоритму мають меншу евклідову норму ніж в інших розглянутих алгоритмах і, відповідно, візуально вони є теж менш пошкодженими.

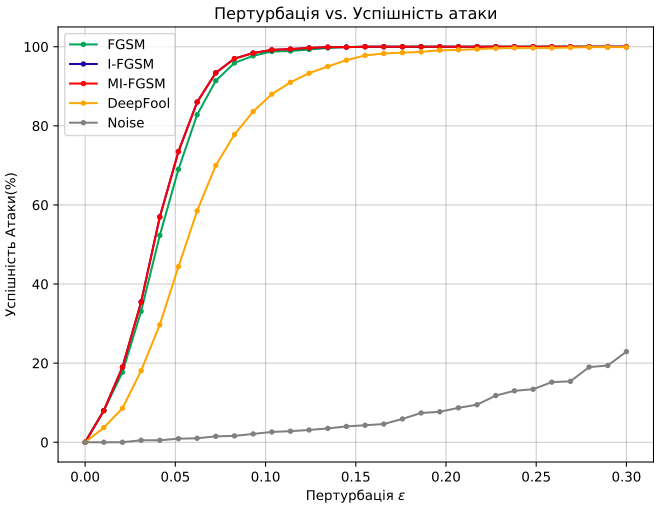

Рис. 5.1: Графік залежності успішності атаки від величини пертурбації. Нижньою межею буде виступати випадковий шум.

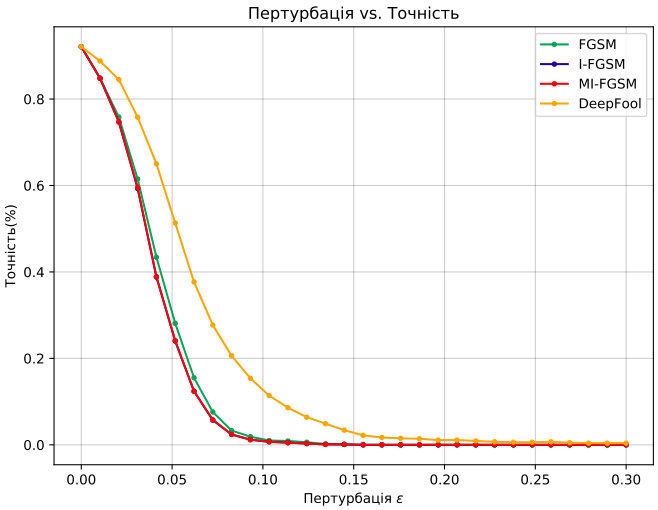

Рис. 5.2: Графік залежності успішності класифікації моделі від величини пертурбації. Початкова точність моделі $\sim 92\%$

5.2 Аналіз захисту

Для того щоб проаналізувати методи захисту, використовувались оригінальні тестові зображення, а також зображення які були модифіковані певним алгоритмом захисту. Далі для обох множин прикладів застосовувались розглянуті атаки, після чого були побудовані порівняльні графіки.

Randomization

З наведених графіків 5.4 можна побачити, що алгоритм випадкової зміни розмірності опинився неефективним для захисту даної моделі. Хоча була перевірена велика кількість різних варіантів параметрів, знайти такі, щоб точність моделі залишалась на високому рівні так і не вдалось. Навіть при дуже малих деформаціях зображень, класифікатор починав передбачати їх суттєво гірше і точність моделі знижувалась.

Pixel Deflection

Другий протестований метод захисту від атак на моделі машинного навчання продемонстрував себе значно краще ніж попередній. Навіть при відносно великій кількості зсунутих пікселів модель майже не втрачала точність. В результаті підбирання оптимальних параметрів для деформації вхідного зображення вдалось

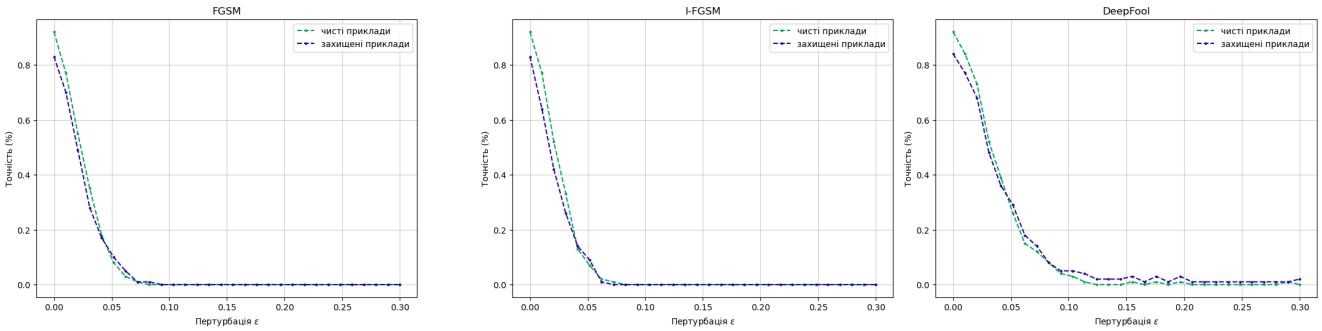

Рис. 5.3: Randomization

досягнути того, щоб при класифікації без генерування змагальних прикладів, тобто коли $\varepsilon = 0$, точність моделі не падала зовсім 5.4. Однак, використовуючи ці параметрах успішність проведених атак знижувалась досить слабо. Тому попри те, що цей метод в правильних умовах ніяк не шкодить ефективності моделі, такий спосіб не можна вважати надійним захистом від атак.

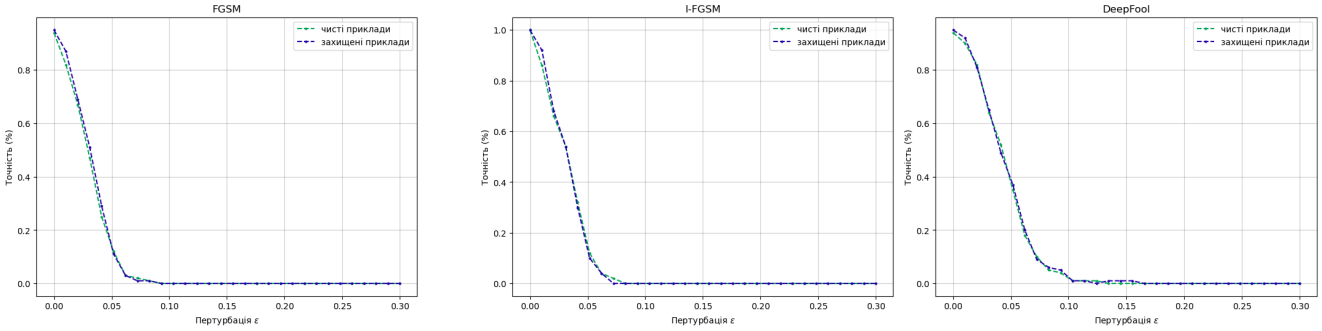

Рис. 5.4: Pixel Deflection

Загалом можна зробити висновок що методи захисту які базуються на ідеї випадкових деформацій зображень є не дуже ефективними у випадку лінійних моделей машинного навчання. Основна причина цього полягає в тому, що для того щоб захиститись від атак такими методами необхідно досить суттєво деформувати вхідне зображення, а через те що лінійні методи не здатні побудувати достатньо складні зв'язки між даними, вони є досить вразливі до такий деформацій. Отже, точність передбачення на деформованих зображеннях може падати досить швидко і такі методи захисту не можуть бути використані належним чином.

Розділ 6

Висновок

В результаті роботи була реалізована лінійна модель машинного навчання, на якій були проаналізовані декілька методів генерування змагальних прикладів. Серед них були розглянуті однокроковий алгоритм FGSM, його ітераційні варіації, а також DeepFool метод. В межах L_∞ норми найефективнішими методами опинились ітераційна модифікація та momentum оптимізація алгоритму FGSM. Також були розглянуті два методи захисту які базуються на ідеї руйнування структури змагальних прикладів через додавання в зображення контрольованого шуму. За результатами експериментів, метод випадкової зміни розмірності зображення опинився неефективним для захисту лінійних моделей машинного навчання. Інший метод захисту Pixel Deflection стабільно зменшував ймовірність успішної атаки, але за отриманими результатами можна сказати, що він теж не є абсолютно надійним захистом від розглянутих методів атак.

Бібліографія

[1] Ian J. Goodfellow, Jonathon Shlens та Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2014. arXiv: 1412.6572 [stat.ML].

[2] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi та Pascal Frossard. *DeepFool: a simple and accurate method to fool deep neural networks*. 2015. arXiv: 1511.04599 [cs.LG].

[3] Alexey Kurakin, Ian Goodfellow та Samy Bengio. *Adversarial examples in the physical world*. 2016. arXiv: 1607.02533 [cs.CV].

[4] Yinpeng Dong та ін. *Boosting Adversarial Attacks with Momentum*. 2017. arXiv: 1710.06081 [cs.LG].

[5] Xiaoyong Yuan та ін. *Adversarial Examples: Attacks and Defenses for Deep Learning*. 2017. arXiv: 1712.07107 [cs.LG].

[6] Alexey Kurakin та ін. *Adversarial Attacks and Defences Competition*. 2018. arXiv: 1804.00097 [cs.CV].

[7] Muzammal Naseer, Salman H. Khan та Fatih Porikli. *Local Gradients Smoothing: Defense against localized adversarial attacks*. 2018. arXiv: 1807.01216 [cs.CV].

[8] Aaditya Prakash та ін. *Deflecting Adversarial Attacks with Pixel Deflection*. 2018. arXiv: 1801.08926 [cs.CV].

[9] Gokula Krishnan Santhanam та Paulina Grnarova. *Defending Against Adversarial Attacks by Leveraging an Entire GAN*. 2018. arXiv: 1805.10652 [stat.ML].

[10] Yinpeng Dong та ін. *Benchmarking Adversarial Robustness*. 2019. arXiv: 1912.11852 [cs.CV].