НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра прикладної математики

КУРСОВА РОБОТА

із дисципліни «Криптографічні методи захисту інформації»

на тему:

Моделювання створення та криптоаналізу симетричних систем шифрування з метою забезпечення конфіденційної комунікації

Виконав:
студент групи КМ-81
Верзун П.В.

Керівник:
ст. викладач Бай Ю. П.

Київ — 2020

Зміст

Вступ

Зі збільшенням доступності квантових хмарних обчислювальних систем та все більшою кількістю випадків незаконної публікації чи присвоєння конфіденційних даних, питання зокрема про нові методи забезпечення конфіденційності стало особливо актуальним.

Впродовж останніх років архітектури нейронних мереж відійшли від задач вивчення залежностей в розмічених даних та імітування чи екстраполяції прихованих імовірнісних розподілів. Задачі, що вирішуються нейронними мережами стають все більш абстрактними, а зі збільшенням обчислювальних швидкостей та ростом їх доступності зросла і кількість проведених досліджень у цій сфері.

Зокрема, було запропоновано багато методів забезпечення конфіденційності передачі даних по незахищеним каналам, що базуються не на існуючих математичних методах шифрування, а на вивчених методах розшифрування і шифрування даних без заданої заздалегідь інструкції щодо алгоритму.

У даній курсовій роботі я хочу зосередитися на роботі 2016 року від Google Brain на тему “Learning to protect communication with adversarial neural cryptography”. У цій роботі за допомогою 3х нейронних мереж - Alice, Bob та Eve було побудовано мульти-агентну систему, що імітує класичну задачу криптоаналізу забезпечення конфіденційності при передачі інформації по незахищеному каналу.

Мета роботи - відтворити результати із роботи Google Brain, дослідити ліміти даної системи, її сильні та слабкі сторони, а також визначити її місце серед схожих досліджень та перспективи даного напряму.

Основні відомості про обрану архітектуру, а також криптографічні характеристики досліджуваної системи

Як було вже було згадано, запропонована мульти-агентна система складається з трьох агентів, кожен з яких має свою ціль та архітектуру - Alice, Bob та Eve. Перед тим як описувати архітектуру та принцип роботи системи, необхідно ввести декілька понять з глубокого навчання:

Adversarial Neural Networks (або дослівний переклад: змагальні нейронні мережі) - це такі нейронні мережі, в серці архітектури яких два агенти, один з яких має ціль досконало виконувати деяку поставлену задачу та генерувати дані. Такий агент/агенти також часто називаються генераторами. Другий агент має класифікувати згенеровані дані як “справжні”, або “згенеровані”, такий агент також називають дискримінатором (звісно задача може бути сформульована і більш комплексним чином).

В досліджуваній роботі ціль агентів Alice та Bob - вести конфіденційну комунікацію, а ціль Eve - “підслуховувати” за двома іншими агентами. Принцип роботи системи можна узагальнити в діаграмі на рис. 1 в даному випадку Alice та Bob - генератори даних, а Eve дискримінатор. Більш точно - задача агентів це мінімізація спільних бітів між здогадкою Eve та реальнім текстом. Для цього в процесі навчання мережі самі обирають деяку чітко визначену криптосистему. Треба розуміти, що інтерпретація нейронних мереж це дуже складна задача, дана робота не намагається передати чітко як працює вивчена криптосистема, але ми можемо проаналізувати її на криптостійкість та ефективність в процесі навчання.

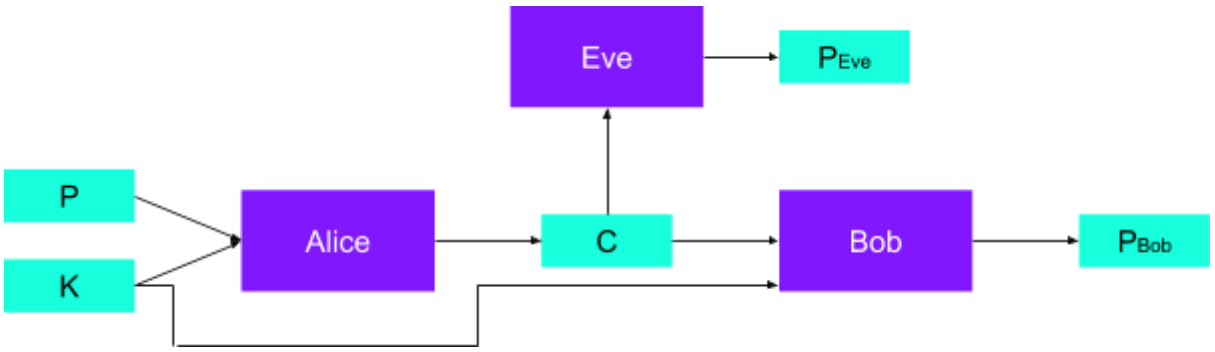

Рис. 1 Alice, Bob та Eve з симетричною криптосистемою. P - plain text, відкритий текст, а C - ciphertext, зашифрований текст, K - секретний ключ

Alice бажає надіслати Bob одне конфіденційне повідомлення. Повідомлення P - вхідне для Alice. Коли Alice обробляє вхідний текст P, ця мережа повертає C - зашифрований текст. І Bob, і Eve отримують C, обробляють його та намагаються відновити P. Ми представляємо що вони обчислюють відповідно за P Bob і P Eve. Alice, Bob мають перевагу над Eve: вони поділяють секретний ключ K. Ми розглядаємо K як додатковий вхід для Alice та Bob. Ми приймаємо один свіжий ключ K на відкритий текст P, але, принаймні на цьому абстрактному рівні, ми не вважаємо, що K і P мають однакову довжину.

Alice, Bob та Eve - нейронні мережі, кожна з яких має параметри, які ми записуємо відповідно $\theta_A, \theta_B, \theta_E$ Оскільки θ_A і θ_B не повинні бути рівним, побудовані нейронними мережами алгоритми шифрування та дешифрування не повинні бути однаковими, навіть якщо Alice та Bob мають однакову архітектуру. В загальному випадку, ці моделі вивчають які біти K та P потрібно поєднати, щоб Eve не розшифрувала більше половини зашифрованих бітів. Всі використані нейронні мережі працюють з кортежами чисел із типу float, а не з послідовності бітів. Іншими словами, K, P, P Bob, P Eve, і C - це кортежі чисел із плаваючою комою.

Ця система, хоча і елементарна з точки нейронних мереж, достатня для демонстрації можливості вивчення та створення ефективних алгоритмів шифрування та дешифрування на їх основі. Зокрема існує можливість для Alice та Bob прийняти ключ K за абсолютно безпечний та проводити шифрування та дешифрування за допомогою простого XOR, але така поведінка не вимагається від моделей.

Якщо оцінювати криптостійкість та структуру системи, то хоча в даному дослідженні не проводилося вичерпного аналізу вивченого методу шифрування, було зроблено кілька спостережень. По-перше, отриманий алгоритм залежить від ключа: зміна ключа та утримання відкритого тексту результують в різному фінальному зашифрованого тексті. Метод також залежить від вхідного тексту, що й потрібно для успішного конфіденційного спілкування. Однак це не просто XOR. Зокрема, вихідні значення часто є значеннями з плаваючою комою, відмінні від 0 і 1. Більше того, ефект зміни на один біт ключа, або відкритого тексту поширюється на безліч елементів у зашифрованому тексті, не обмежуючись одним бітом як це було б з XOR. Тобто мінімальна зміна топології ключа приводить до повної зміни топології зашифрованого тексту. Однобітове перекидання ключа зазвичай спричиняє суттєві зміни в 3-6 з 16 елементів у зашифрованому тексті та менші зміни в інших елементах. А так як однією з цілей для агентів є не тільки як защифрутати текст, а й зрозуміти які біти шифрувати, а які - ні, в результуючому алгоритмі біти відкритого тексту є рівномірно розподіленими по шифротексту.

Детальний опис алгоритму

Задачі (objectives) для агентів

Неформально, мета Eve проста: реконструювати P точно (іншими словами, мінімізувати помилку між P і P Eve). Alice та Bob хочуть чітко комунікувати (мінімізувати помилку між P і P Bob), але також приховувати їх спілкування від Eve. Відповідно до сучасних криптографічних визначень, ми не вимагаємо, щоб зашифрований текст C «виглядав випадковим» для Єви. Зашифрований текст може навіть мати очевидні метадані, які ідентифікують їх як такі. Тому Eve не має на меті розрізняти C від випадкового значення, отриманого з деякого розподілу. У цьому відношенні цілі Eve протиставляються класичним для дискримінаторів в GAN. З іншого боку, можна спробувати переформулювати мету Eve з точки зору розрізнення шифротекстів, побудованих з двох різних відкритих текстів.

Враховуючи ці цілі, замість того, щоб навчати кожного агента Alice та Bob окремо, виконувати деякі відома криптосистеми, ми навчаємо їх спілкуватися успішно і “перемогти” Eve без попередньо визначеного уявлення про те, яку криптосистему вони можуть для цього виявити. Як і у визначеннях GAN, ми хотіли б, щоб Alice та Bob перемогли найкращу можливу версію Eve, а не фіксовану Eve. Тому ми припускаємо розподіл по простому тексту та ключам та формулюємо наші цілі для Alice та Bob з точки зору очікуваних значень.

Ми знаходимо $A(\theta_A, P, K)$ для виводу Alice на вхідне P, K, знаходимо $B(\theta_B, P, K)$ для вихідного результату для Bob для отриманої C, K і знаходимо $E(\theta_E, C)$ для виходу Eve на вхід C. Введемо функцію відстані d для текстів. Для того щоби оцінювати роботу алгоритму потрібно простів вхідних текстів зробити метричним. Хоча точний вибір

цієї метрики не є вирішальним, для конкретності ми візьмемо відстань в метричному просторі L1:

$$d\,(P_i,\ P_i') \ =\ \sum_{i}^{N} |P_i \ -\ P_i'|$$

де N - довжина відкритих текстів (в нашому випадку - 16 бітів, але вона може бути будь-якою, потрібно лише повторити процесс навчання для інших значень). А loss функція наприклад для Eve:

$$L_E\,(\theta_A,\ \theta_E,\ P,\ K) \ =\ d\,(P,\ E\,(\theta_E,\ A\,(\theta_A,\ P,\ K)))$$

Інтуїтивно $L_E\,(\theta_A,\ \theta_E,\ P,\ K)$ відображає, наскільки Eve помиляється, коли відкритим текстом є P, а Ключ - K. Ми також визначаємо loss функцію для Eve над розподілом у відкритих текстах і ключах приймаючи очікуване значення:

$$L_E\,(\theta_A,\ \theta_E) = \ E_{P,K}\,(d\,(P,\ \ E\,(\theta_E,\ A\,(\theta_A,\ P,\ K))))$$

Ми отримуємо "оптимальну Eve", мінімізуючи таку loss функцію:

$$O_E\,(\theta_A) \ =\ argmin(\theta_E\,(L_E\,(\theta_A,\ \theta_E))$$

Подібним чином, ми визначаємо помилку реконструкції для кожного прикладу для Bob і поширюємо її на розподіл відкритих текстів та ключів:

$$L_B\,(\theta_A,\ \theta_B,\ P,\ K) \ =\ d\,(P,\ B\,(\theta_B,\ A\,(\theta_A,\ P,\ K)))$$

$$L_B\,(\theta_A,\ \theta_B) \ =\ E_{P,K}(d\,(P,\ B\,(\theta_B,\ A\,(\theta_A,\ P,\ K),\ K))$$

Ми визначаємо функцію втрат для Alice та Bob, комбінуючи L_B та оптимальне значення L_E:

$$L_{AB}\,(\theta_A,\ \theta_B) \ =\ L_B\,(\theta_A,\ \theta_B) - L_E\,(\theta_A,\ O_E\,(\theta_A))$$

Ця комбінація відображає, що Alice та Bob мінімізують помилку реконструкції Bob та максимізують помилку реконструкції “оптимальної Eve”.

Ми отримуємо "оптимальних Alice та Bob", мінімізуючи

$L_{AB}(\theta_A, \theta_B)$:

$$(O_A, O_B) = argmin_{(\theta_A, \theta_B)}(L_{AB}(\theta_A, \theta_B))$$

Ми пишемо “оптимальне” в лапках, оскільки не потрібно мати єдиного загального мінімуму. Загалом, є багато рівно оптимальних рішень для Alice та Bob. Як простий приклад, припускаючи, що ключ такого ж розміру, як відкритий текст і зашифрований текст, Аліса і Боб можуть XOR відкритий текст із зашифрованим текстом відповідно до будь-якої перестановки ключа, і всі перестановки однаково підходять, поки Alice та Bob використовують одну і ту саму перестановку; більше того, з тим, як ми будуємо наші мережі, усі перестановки однаково ймовірні.

Навчання починається з випадкової ініціалізації мереж Alice та Bob. Мета навчання - дійти від цього стану до (O_A, O_B) , або близько до (O_A, O_B) .

Архітектура Alice, Bob, and Eve.

Так як наша ціль це ефективна комунікація, а не навчання якогось певного алгоритму шифрування, ми хочемо створити таку архітектуру, що достатня для вивчення таких алгоритмів “змішування” як XOR, але не функціонує як якийсь конкретний алгоритм.

З цією метою в даній роботі обрали наступну архітектуру “mix & transform”.

Він має перший повністю з’єднаний (FC-fully connected) шар, де кількість виходів дорівнює кількості входів. Біти відкритого тексту і ключа подаються в цей шар FC. Оскільки кожен вихідний біт може бути лінійною комбінацією всіх вхідних бітів, цей рівень дозволяє, але не вимагає, змішування між ключем і бітами відкритого тексту.

Зокрема, цей шар може переставляти біти. Шар FC супроводжується послідовністю конволюційних (згорткових) шарів, останній з яких видає

вихідний текст розміру, що підходить для відкритого тексту або зашифрованого тексту.

Ці згорткові шари вчаться застосовувати деяку функцію до груп бітів, змішаних попереднім шаром без апріорного уточнення, якою має бути ця функція. Відмітимо, що зворотній порядок (згортковий шар, а потім FC) набагато частіше зустрічається в програмах обробки зображень. Нейронні

мережі, розроблені для цих додатків, часто використовують згортки.

Програмна реалізація алгоритму

Для тренування нейронної мережі довжина ключа, тексту та шифру обрана 32 біти. Для кожного з трьох агентів кількість шарів нейронної мережі - 5, а кількість прихованих нейронів у кожному шарі - 64. Розмір батча з даними, на яких рахується градієнт та оновлюються ваги нейронних мереж агентів на кожній ітерації - 4096
Для оптимізації використано алгоритм Adam, а крок, що задає швидкість збіжності методу - 0.001

У оригінальній статті та при повторенні експерименту було отримано наступний графік еволюції у часі (вісь абсцис) помилково реконструйованих бітів (по осі ординат) кожним з агентів

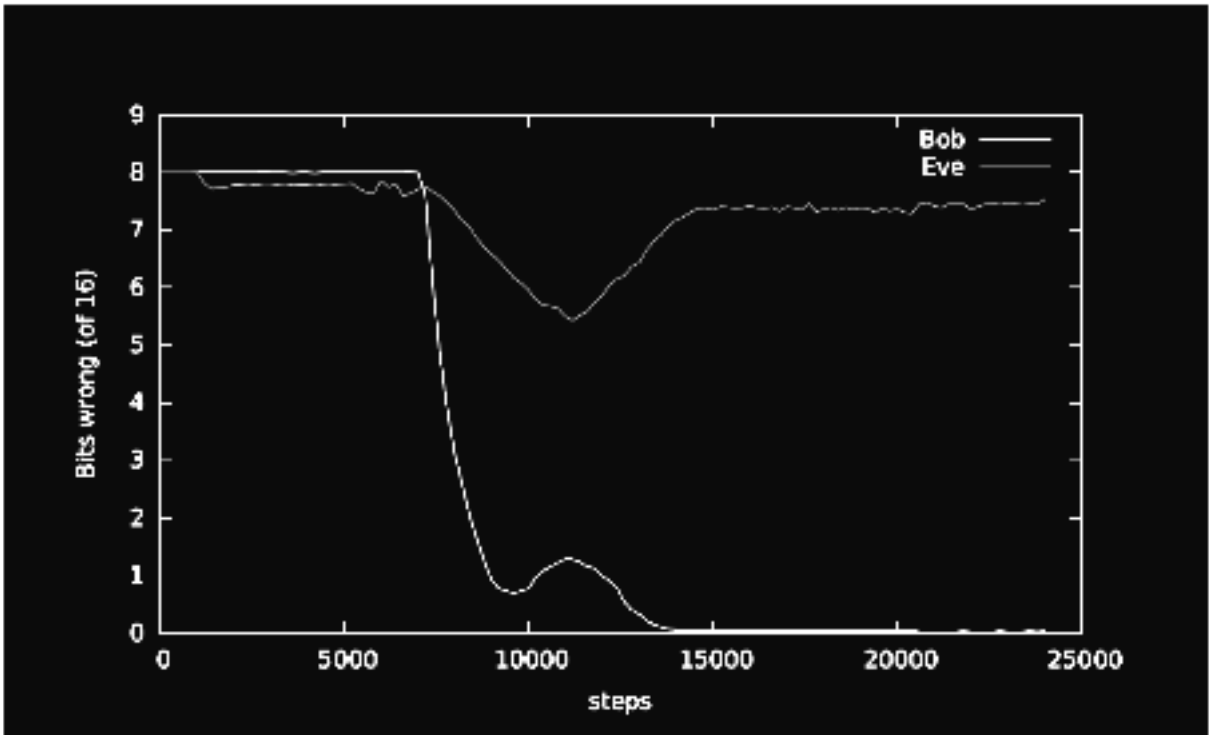

Рис. 2: Еволюція помилок реконструкції оригінального шифротексту агентами Бобом та Євою залежно від кількості ітерацій навчання. Графіки показують помилку у вигляді кількість неправильно вгаданих бітів у повідомленні, усередненену для розміру міні-партії у 4096 різних реалізацій P.

Кожна точка на графіку є середньою помилкою для 4096 прикладів.

Ідеальним результатом реконструкції буде помилка рівна нулю. Помилка реконструкції Eve та Bob рівні 8 (половина бітів знайдені неправильно). У цьому прикладі обидві помилки реконструкції мають високі значення спочатку. Через певну кількість ітерацій навчання, з часом Alice та Bob починають спілкуватися досить ефективно, але ще таким чином, щоб це дозволяло Eve покращувати її розуміння тексту, про що свідчить поступове падіння зеленого графіку помилки Eve з 7000 до 10 000 ітерації навчання.

Потім, приблизно на ітерації з номером 10 000, Alice та Bob фіксують прогрес Eve та починається ускладнення моделі шифрування, через що помилка Bob починає трохи зростати, але у той самий час помилка Eve зростає досить сильно. Приблизно на 15 000 кроках цілі навчання ефективно досягаються, де знайдено таку модель шифрування, яка доволі складна для того, щоб її могла розгадати Eve, маючи доступ тільки до C та P.

Приклад проміжних результатів отриманих у даній курсовій роботі під час навчання наприкінці першої епохи та наприкінці всього навчання наведені в Додатку 4.

Графіки навчання агентів:

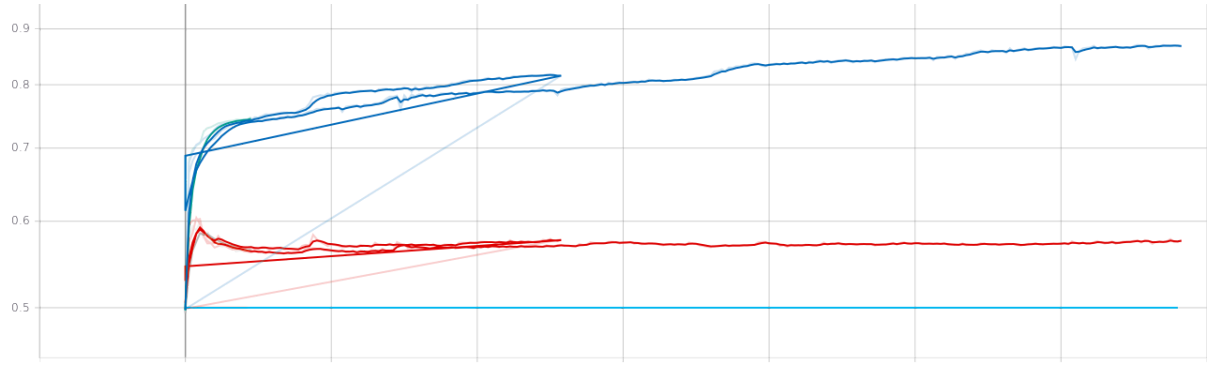

Рис. 3 Графік залежності точності агентів від ітерації навчання

Значення точності на початку та наприкінці навчання:

Name	Smoothed	Value	Step
initial/Bob	0.677	0.7049	750
initial/Eve	0.5843	0.6043	750
initial/random	0.5	0.5	750
kursach_run_14/Bob	0.6683	0.7037	750
kursach_run_14/Eve	0.5749	0.5888	750
kursach_run_14/random	0.5	0.5	750

Name	Smoothed	Value	Step
initial/Bob	0.8679	0.8669	68.25k
initial/Eve	0.5757	0.5769	68.25k
initial/random	0.5	0.5	68k
kursach_run_14/Bob	0.7446	0.7469	4.5k
kursach_run_14/Eve	0.5649	0.5656	4.5k
kursach_run_14/random	0.5	0.5	4.25k

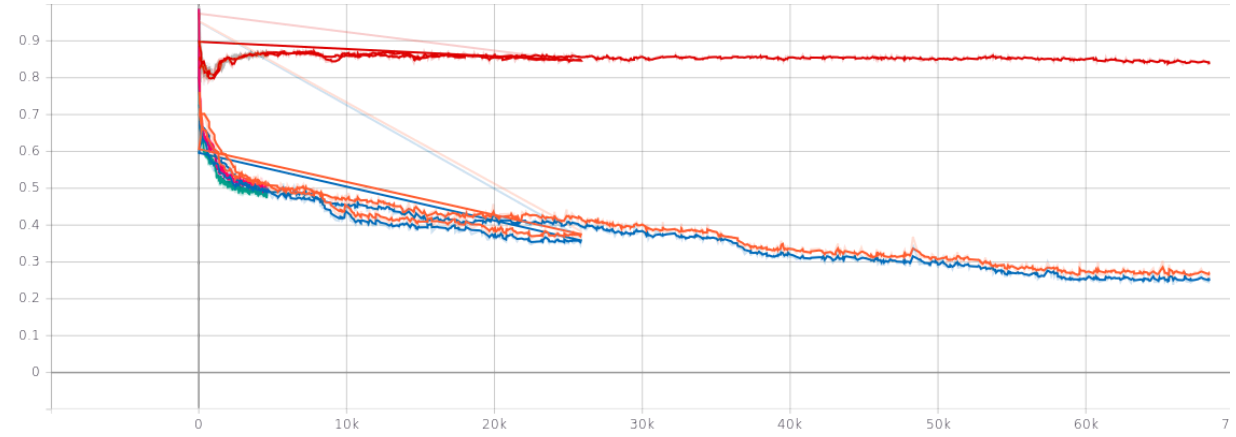

Рис. 4 Графік залежності значення loss функції агентів від ітерації навчання

Значення loss функції на початку та наприкінці навчання:

Name	Smoothed	Value	Step
initial/AB	0.6566	0.653	475
initial/Bob	0.6224	0.6109	475
initial/Eve	0.8272	0.7983	475
kursach_run_14/AB	0.6353	0.6325	465
kursach_run_14/Bob	0.6007	0.599	466
kursach_run_14/Eve	0.8118	0.8239	466

Name	Smoothed	Value	Step
initial/AB	0.2726	0.2696	66.21k
initial/Bob	0.2515	0.2491	66.21k
initial/Eve	0.8473	0.8505	66.21k
kursach_run_14/AB	0.5261	0.5268	2.446k
kursach_run_14/Bob	0.501	0.5088	2.445k
kursach_run_14/Eve	0.8558	0.857	2.445k

Висновки

У цій роботі ми демонструємо, що нейронні мережі можуть навчитися захищати комунікації. Навчання- не вимагає прописування певного набору криптографічних алгоритмів, а також зазначення способів застосування цих алгоритмів, але базується лише на специфікації цілі навчання як задачі оптимізації.

Доколи існуватиме потреба зашифровувати дані, існуватиме потреба ламати існуючі алгоритми шифрування. У даній роботі розглянуто систему з агенту - винахіднику нових шифрів та методів конфіденційної комунікації і агенту - криптоаналитику. Агенти Alice та Bob знаходять все кращі і кращі алгоритми шифрування з кожною ітерацією, у той час як Eva намагається розгадати яким шифром користуються Alice та Bob.

Таким чином виходить, що Eva - свого роду самонавчаючийся криптоаналітик, який намагається зламати шифр, що з часом удосконалюється.

Еволюційна модель навчання, запропонована в даній роботі, дуже нагадує еволюцію методів в реальній історії криптографії, де на все більш складні шифри робили більш витончені атаки.

Було показано, що поточна архітектура здатна покращувати як алгоритм шифрування, так і спосіб злому, про що свідчить форма графіків кількості неправильно відгаданих бітів повідомлення. Усереднення по великій кількості повідомлень для кожної точки на графіку, свідчить про статистичну значимість результатів моделювання.

Сучасна криптографія - це більше, ніж шифрування. Подальша робота може розглядати інші завдання, наприклад, стеганографія, генерація псевдовипадкових чисел або перевірка цілісності. Нарешті, нейронна

мережі можуть бути корисними не тільки для криптографічного захисту, а й для атак.

Список літератури

1. “Learning to protect communication with adversarial neural cryptography” - Martín Abadi and David G. Andersen, 2016
2. Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Info-GAN: Interpretable representation learning by information maximizing generative adversarial nets. CoRR, abs/1606.03657, 2016
3. Sébastien Dourlens. Applied neural-cryptography. 1996.
4. Harrison Edwards and Amos J. Storkey. Censoring representations with an adversary. CoRR, abs/1511.05897, 2015. Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate to solve riddles with deep distributed recurrent Q-networks. CoRR, abs/1602.02672, 2016a. Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to com-
5. municate with deep multi-agent reinforcement learning. CoRR, abs/1605.06676, 2016b.
6. Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor S. Lempitsky. Domain-adversarial training of neural networks. CoRR, abs/1505.07818, 2015.
7. D. R. Stinson, Cryptography: Theory and Practice (CRC Press 1995)
8. M. Rosen-Zvi, I. Kanter and W. Kinzel, Cryptography based on neural networks: analytical results (2002)

9. M. Rosen-Zvi, E. Klein, I. Kanter and W. Kinzel, Mutual learning in a tree parity machine and its application to cryptography, Phys. Rev. E (2002)
10. R. Urbanczik, private communication

Додатки

Додаток 1. Код моделей

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class Alice(nn.Module):
        def __init__(self, hp):
        super(Alice, self).__init__()
        self.depth = hp.alice.depth
        self.hidden = hp.alice.hidden

        self.mlp = nn.ModuleList(
        [nn.Linear(hp.data.plain + hp.data.key, self.hidden)]
        + [nn.Linear(self.hidden, self.hidden) for _ in range(self.depth - 1)]
        )
        self.last = nn.Linear(self.hidden, hp.data.cipher)

        def forward(self, p, k):
        print("Alice's forward")
        print("Input p={}, k={}".format(p, k))
        x = torch.cat((p, k), dim=-1)

        for idx, layer in enumerate(self.mlp):
        if idx == 0:
                x = F.relu(layer(x))
        else:
                x = F.relu(x + layer(x))

        x = torch.tanh(self.last(x))
        print("Output calculated by Alice {}".format(x))
        return x


class Bob(nn.Module):
        def __init__(self, hp):
        super(Bob, self).__init__()
        self.depth = hp.bob.depth
        self.hidden = hp.bob.hidden
```

```
        self.mlp = nn.ModuleList(
        [nn.Linear(hp.data.cipher + hp.data.key, self.hidden)]
        + [nn.Linear(self.hidden, self.hidden) for _ in range(self.depth - 1)]
        )
        self.last = nn.Linear(self.hidden, hp.data.plain)

        def forward(self, c, k):
        print("Bob's forward")
        print("Input c={}, k={}".format(c, k))

        x = torch.cat((c, k), dim=-1)

        for idx, layer in enumerate(self.mlp):
        if idx == 0:
                x = F.relu(layer(x))
        else:
                x = F.relu(x + layer(x))

        x = torch.tanh(self.last(x))
        print("Output calculated by Bob {}".format(x))
        return x


class Eve(nn.Module):
        def __init__(self, hp):
        super(Eve, self).__init__()
        self.depth = hp.eve.depth
        self.hidden = hp.eve.hidden

        self.mlp = nn.ModuleList(
        [nn.Linear(hp.data.cipher, self.hidden)]
        + [nn.Linear(self.hidden, self.hidden) for _ in range(self.depth - 1)]
        )
        self.last = nn.Linear(self.hidden, hp.data.plain)

        def forward(self, c):
        print("Eve's forward")
        print("Input c={}".format(c))

        x = c

        for idx, layer in enumerate(self.mlp):
        if idx == 0:
```

```
        x = F.relu(layer(x))
else:
        x = F.relu(x + layer(x))

x = torch.tanh(self.last(x))
print("Output calculated by Eve", x)
return x
```

Додаток 2. Код алгоритму навчання

```
import os
import math
import tqdm
import torch
import torch.nn as nn
import torch.nn.functional as F
import itertools
import traceback

from .model import Alice, Bob, Eve
from .validation import validate


def train(args, trainloader, valloader, writer, logger, hp, hp_str):
        alice = Alice(hp).cuda()
        bob = Bob(hp).cuda()
        eve = Eve(hp).cuda()

        optim_e = torch.optim.Adam(eve.parameters(), lr=hp.train.adam.lr)
        optim_ab = torch.optim.Adam(
        list(alice.parameters()) + list(bob.parameters()), lr=hp.train.adam.lr
        )

        step = 0

        criterion = nn.L1Loss()

        try:

        alice.train()

        bob.train()

        eve.train()

        for epoch in itertools.count(0):

        if epoch % hp.log.validation == 0:

                with torch.no_grad():

                validate(hp, args, alice, bob, eve, valloader, writer, step, epoch)
```

```
loader = tqdm.tqdm(trainloader)
for plainE, keyE, plainAB, keyAB in loader:
        plainE = plainE.cuda()
        keyE = keyE.cuda()
        plainAB = plainAB.cuda()
        keyAB = keyAB.cuda()

        # Eve
        optim_e.zero_grad()
        cipher = alice(plainE, keyE).detach()
        out_e = eve(cipher)
        loss_e = criterion(plainE, out_e)
        loss_e.backward()
        optim_e.step()
        loss_e_temp = loss_e.item()

        # Alice & Bob
        optim_ab.zero_grad()
        cipher = alice(plainAB, keyAB)
        out_e = eve(cipher)
        out_b = bob(cipher, keyAB)
        loss_e = criterion(plainAB, out_e)
        loss_b = criterion(plainAB, out_b)
        loss_ab = loss_b + (1.0 - loss_e).pow(2)
        loss_ab.backward()
        optim_ab.step()
```

```
                loss_b = loss_b.item()
                loss_ab = loss_ab.item()

                # logging
                step += 1
                tmp = max(loss_ab, loss_b, loss_e_temp)
                if tmp > 1e8 or math.isnan(tmp):
                logger.error(
                "loss exploded AB %f B %f E %f" % (loss_ab, loss_b,
loss_e_temp)
                )
                raise Exception("Loss exploded")

                writer.log_train(loss_ab, loss_b, loss_e_temp, step)
                loader.set_description(
                "AB %.04f B %.04f E %.04f step %d"
                % (loss_ab, loss_b, loss_e_temp, step)
                )

        except Exception as e:
        logger.info("Exiting due to exception: %s" % e)
        traceback.print_exc()
```

Додаток 3. Код алгоритмів валідації

```
import torch


def validate(hp, args, alice, bob, eve, valloader, writer, step, epoch):
        alice.eval()
        bob.eval()
        eve.eval()

        correct_e = 0
        correct_b = 0
        for plain, key in valloader:
        plain = plain.cuda()
        key = key.cuda()
        cipher = alice(plain, key)
        outE = eve(cipher)
        outB = bob(cipher, key)
        correct_e += torch.sum(torch.abs(plain - outE) < 1).item() / hp.data.plain
        correct_b += torch.sum(torch.abs(plain - outB) < 1).item() / hp.data.plain

        acc_e = correct_e / len(valloader.dataset)
        acc_b = correct_b / len(valloader.dataset)

        writer.log_accuracy(acc_b, acc_e, step)
        print("Accuracy(%%): Bob %.1f Eve %.1f" % (100.0 * acc_b, 100.0 *
acc_e))
```

```
if epoch % 10 == 0 and epoch != 0:
torch.save(
{"epoch": epoch, "model_state_dict": alice.state_dict(),},
"./nncrypt/checkpoints/alice_epoch_{}.pth".format(epoch),
)
torch.save(
{"epoch": epoch, "model_state_dict": bob.state_dict(),},
"./nncrypt/checkpoints/bob_epoch_{}_acc_{}.pth".format(
        epoch, 100.0 * acc_b
),
)

torch.save(
{"epoch": epoch, "model_state_dict": eve.state_dict(),},
"./nncrypt/checkpoints/eve_epoch_{}_acc_{}.pth".format(
        epoch, 100.0 * acc_e
),
)

alice.train()
bob.train()
eve.train()
```

Додаток 4. Виконання шифрування та дешифрування під час навчання

```
Alice's forward
Input p=tensor([[-1.,  1., -1.,  ...,  1., -1.,  1.],
        [ 1.,  1.,  1.,  ...,  1., -1.,  1.],
        [ 1.,  1.,  1.,  ..., -1.,  1., -1.],
        ...,
        [ 1., -1.,  1.,  ..., -1., -1.,  1.],
        [ 1.,  1.,  1.,  ...,  1., -1., -1.],
        [-1.,  1., -1.,  ...,  1.,  1., -1.]]), k=tensor([[ 1., -1., -1.,  ...,  1.,  1., -1.],
        [-1., -1.,  1.,  ..., -1.,  1.,  1.],
        [-1., -1., -1.,  ...,  1.,  1.,  1.],
        ...,
        [ 1., -1.,  1.,  ..., -1.,  1., -1.],
        [-1.,  1.,  1.,  ..., -1.,  1.,  1.],
        [ 1.,  1., -1.,  ...,  1.,  1.,  1.]],)
C Alice = tensor([[-0.1897,  0.9674,  0.9583,  ...,  0.8362, -0.9598,  0.6463],
        [-0.4012,  0.8853,  0.8870,  ...,  0.8652, -0.9470,  0.4666],
        [-0.7048,  0.8810,  0.9013,  ...,  0.6606, -0.7868,  0.8985],
        ...,
        [-0.8244,  0.5529,  0.9508,  ..., -0.6710, -0.7416,  0.9625],
        [-0.4723,  0.9768,  0.9702,  ...,  0.3670, -0.9817,  0.6040],
        [-0.6797,  0.7105,  0.9819,  ..., -0.3688, -0.9169,  0.9801]],)

Eve's forward
Input c=tensor([[-0.1897,  0.9674,  0.9583,  ...,  0.8362, -0.9598,  0.6463],
        [-0.4012,  0.8853,  0.8870,  ...,  0.8652, -0.9470,  0.4666],
        [-0.7048,  0.8810,  0.9013,  ...,  0.6606, -0.7868,  0.8985],
        ...,
        [-0.8244,  0.5529,  0.9508,  ..., -0.6710, -0.7416,  0.9625],
        [-0.4723,  0.9768,  0.9702,  ...,  0.3670, -0.9817,  0.6040],
        [-0.6797,  0.7105,  0.9819,  ..., -0.3688, -0.9169,  0.9801]],
        )
P Eve = tensor([[-0.9999,  0.9984,  0.9997,  ...,  0.9608, -0.9102, -0.9999],
        [-0.9709,  0.9997,  0.9992,  ..., -0.6196, -0.9994, -0.9993],
        [-0.5531,  1.0000,  0.9986,  ..., -0.9992, -0.9996, -0.9945],
        ...,
        [-0.6828,  0.9942,  0.7527,  ..., -0.9241,  1.0000, -0.9999],
        [-0.9989,  1.0000,  0.9827,  ...,  0.8964, -0.8648, -1.0000],
        [-0.9851,  0.9997,  0.4112,  ...,  0.5502,  1.0000, -1.0000]])
Alice's forward
Input p=tensor([[ 1., -1.,  1.,  ..., -1., -1.,  1.],
        [-1.,  1., -1.,  ...,  1.,  1., -1.],
        [ 1.,  1., -1.,  ..., -1.,  1., -1.],
        ...,
        [ 1.,  1.,  1.,  ..., -1.,  1.,  1.],
        [-1.,  1., -1.,  ..., -1., -1., -1.],
        [ 1., -1.,  1.,  ...,  1., -1., -1.]],), k=tensor([[-1., -1., -1.,  ...,  1.,  1., -1.],
        [-1., -1., -1.,  ..., -1.,  1.,  1.],
        [ 1.,  1., -1.,  ...,  1.,  1.,  1.],
        ...,
        [-1.,  1., -1.,  ..., -1.,  1., -1.],
        [ 1.,  1., -1.,  ...,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  ..., -1.,  1., -1.]])
C Alice tensor([[-0.1364,  0.9953,  0.9346,  ...,  0.1875, -0.9590,  0.7007],
        [-0.8745,  0.6376,  0.9778,  ..., -0.6396, -0.9457,  0.7682],
```

```
        [-0.6231,  0.9973,  0.9698,  ...,  0.3350, -0.9349,  0.9446],
        ...,
        [-0.3426,  0.7593,  0.9617,  ..., -0.5377, -0.9605,  0.0915],
        [-0.9766,  0.9809,  0.9828,  ..., -0.8059, -0.9297,  0.8130],
        [-0.7687,  0.9983,  0.8903,  ...,  0.2387, -0.8722,  0.9331]],)
Eve's forward
Input c=tensor([[-0.1364,  0.9953,  0.9346,  ...,  0.1875, -0.9590,  0.7007],
        [-0.8745,  0.6376,  0.9778,  ..., -0.6396, -0.9457,  0.7682],
        [-0.6231,  0.9973,  0.9698,  ...,  0.3350, -0.9349,  0.9446],
        ...,
        [-0.3426,  0.7593,  0.9617,  ..., -0.5377, -0.9605,  0.0915],
        [-0.9766,  0.9809,  0.9828,  ..., -0.8059, -0.9297,  0.8130],
        [-0.7687,  0.9983,  0.8903,  ...,  0.2387, -0.8722,  0.9331]])

P Eve = tensor([[-0.9954, -0.9461, -0.9140,  ...,  1.0000, -0.9103, -0.9318],
        [-0.9973,  0.9791,  0.9112,  ...,  0.9916,  1.0000, -1.0000],
        [-0.9951,  0.9873, -0.1313,  ...,  0.9660,  0.9385, -0.9999],
        ...,
        [-0.9405, -0.9755,  0.5179,  ...,  0.9983,  1.0000, -0.9983],
        [-0.9385,  1.0000,  0.9487,  ..., -0.9632,  0.9999, -1.0000],
        [-0.9926,  0.9998,  0.0733,  ...,  0.3560,  0.1447, -0.9999]])
Bob's forward
Input c=tensor([[-0.1364,  0.9953,  0.9346,  ...,  0.1875, -0.9590,  0.7007],
        [-0.8745,  0.6376,  0.9778,  ..., -0.6396, -0.9457,  0.7682],
        [-0.6231,  0.9973,  0.9698,  ...,  0.3350, -0.9349,  0.9446],
        ...,
        [-0.3426,  0.7593,  0.9617,  ..., -0.5377, -0.9605,  0.0915],
        [-0.9766,  0.9809,  0.9828,  ..., -0.8059, -0.9297,  0.8130],
        [-0.7687,  0.9983,  0.8903,  ...,  0.2387, -0.8722,  0.9331]],), k=tensor([[-1., -1., -1.,  ...,  1.,  1., -1.],
        [-1., -1., -1.,  ..., -1.,  1.,  1.],
        [ 1.,  1., -1.,  ...,  1.,  1.,  1.],
        ...,
        [-1.,  1., -1.,  ..., -1.,  1., -1.],
        [ 1.,  1., -1.,  ...,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  ..., -1.,  1., -1.]])
P Bob:
tensor([[ 0.9998,  0.9643, -0.9964,  ...,  0.9996, -1.0000, -0.9979],
        [ 0.2772, -0.7386, -1.0000,  ...,  0.9996,  1.0000, -1.0000],
        [-0.9904,  1.0000,  0.9950,  ..., -0.7874,  0.8601, -1.0000],
        ...,
        [ 1.0000, -0.9999, -1.0000,  ..., -0.0600,  0.9956,  0.0121],
        [ 1.0000,  1.0000, -0.9995,  ..., -0.9999, -0.8623, -1.0000],
        [ 1.0000,  1.0000, -0.9872,  ..., -0.9778, -1.0000, -0.9999]],)

AB = 0.6763, B = 0.6378, E = 0.8092, step = 250
```

Приклад проміжних результатів навчання наприкінці навчання:

```
Alice's forward
Input p=tensor([[ 1.,  1., -1.,  ..., -1., -1., -1.],
        [ 1., -1.,  1.,  ..., -1.,  1., -1.],
        [ 1., -1.,  1.,  ..., -1.,  1., -1.],
        ...,
        [ 1.,  1., -1.,  ..., -1.,  1.,  1.],
        [ 1., -1., -1.,  ..., -1.,  1., -1.],
        [-1.,  1., -1.,  ..., -1., -1., -1.]]), k=tensor([[ 1.,  1.,  1.,  ...,  1., -1., -1.],
        [ 1.,  1.,  1.,  ...,  1.,  1., -1.],
        [-1., -1.,  1.,  ...,  1., -1., -1.],
        ...,
        [-1.,  1., -1.,  ..., -1.,  1.,  1.],
```

```
        [-1.,  1.,  1.,  ...,  1., -1.,  1.],
        [ 1., -1., -1.,  ..., -1., -1., -1.]])
C Alice tensor([[-0.0493,  0.1416,  0.0553,  ...,  0.9809,  0.9255, -0.5470],
        [-0.6279,  0.4888,  0.6481,  ...,  0.9579,  0.9314, -0.7191],
        [ 0.4222, -0.1602,  0.3245,  ...,  0.9525,  0.9236, -0.5876],
        ...,
        [-0.0865, -0.3181,  0.2931,  ...,  0.9834,  0.9573, -0.8903],
        [-0.0402,  0.5967,  0.4588,  ...,  0.9904,  0.9723, -0.8498],
        [ 0.4257,  0.5392, -0.2013,  ...,  0.9669,  0.9578, -0.6087]],)
Eve's forward
Input c=tensor([[-0.0493,  0.1416,  0.0553,  ...,  0.9809,  0.9255, -0.5470],
        [-0.6279,  0.4888,  0.6481,  ...,  0.9579,  0.9314, -0.7191],
        [ 0.4222, -0.1602,  0.3245,  ...,  0.9525,  0.9236, -0.5876],
        ...,
        [-0.0865, -0.3181,  0.2931,  ...,  0.9834,  0.9573, -0.8903],
        [-0.0402,  0.5967,  0.4588,  ...,  0.9904,  0.9723, -0.8498],
        [ 0.4257,  0.5392, -0.2013,  ...,  0.9669,  0.9578, -0.6087]],
        device='cuda:0')
P Eve tensor([[ 1.0000,  0.9228, -1.0000,  ...,  1.0000, -1.0000,  1.0000],
        [ 1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -0.9985],
        [ 1.0000,  1.0000, -1.0000,  ...,  1.0000,  1.0000,  1.0000],
        ...,
        [ 1.0000,  1.0000, -1.0000,  ...,  1.0000,  1.0000,  1.0000],
        [ 1.0000, -1.0000, -1.0000,  ..., -1.0000,  1.0000,  0.9996],
        [ 1.0000, -1.0000, -1.0000,  ..., -1.0000,  1.0000, -1.0000]],)
Alice's forward
Input p=tensor([[-1., -1., -1.,  ..., -1.,  1., -1.],
        [-1., -1., -1.,  ..., -1., -1.,  1.],
        [ 1.,  1.,  1.,  ...,  1.,  1.,  1.],
        ...,
        [-1., -1., -1.,  ..., -1., -1.,  1.],
        [ 1., -1.,  1.,  ...,  1.,  1.,  1.],
        [ 1.,  1., -1.,  ...,  1., -1.,  1.]]), k=tensor([[ 1., -1.,  1.,  ..., -1.,  1.,  1.],
        [ 1.,  1., -1.,  ..., -1.,  1., -1.],
        [ 1., -1.,  1.,  ...,  1., -1., -1.],
        ...,
        [ 1., -1., -1.,  ...,  1., -1.,  1.],
        [-1., -1.,  1.,  ...,  1.,  1., -1.],
        [ 1.,  1., -1.,  ..., -1., -1., -1.]])
C Alice tensor([[ 0.3812,  0.0767, -0.2101,  ...,  0.9363,  0.9419, -0.5616],
        [-0.1765,  0.3123, -0.1618,  ...,  0.9767,  0.9721, -0.8509],
        [ 0.5195,  0.3808,  0.2565,  ...,  0.9709,  0.9436, -0.7453],
        ...,
        [ 0.7632, -0.3302, -0.6372,  ...,  0.9582,  0.9808, -0.9196],
        [ 0.0112,  0.4411,  0.4064,  ...,  0.9851,  0.9606, -0.7699],
        [ 0.0203, -0.1700, -0.2210,  ...,  0.9621,  0.9494, -0.8102]],)
Eve's forward
Input c=tensor([[ 0.3812,  0.0767, -0.2101,  ...,  0.9363,  0.9419, -0.5616],
        [-0.1765,  0.3123, -0.1618,  ...,  0.9767,  0.9721, -0.8509],
        [ 0.5195,  0.3808,  0.2565,  ...,  0.9709,  0.9436, -0.7453],
        ...,
        [ 0.7632, -0.3302, -0.6372,  ...,  0.9582,  0.9808, -0.9196],
        [ 0.0112,  0.4411,  0.4064,  ...,  0.9851,  0.9606, -0.7699],
        [ 0.0203, -0.1700, -0.2210,  ...,  0.9621,  0.9494, -0.8102]],)
P Eve tensor([[ 1.0000, -1.0000,  1.0000,  ..., -1.0000,  1.0000, -1.0000],
        [-1.0000, -0.9982, -1.0000,  ...,  1.0000, -1.0000,  1.0000],
        [ 1.0000,  1.0000, -1.0000,  ...,  1.0000,  1.0000,  1.0000],
        ...,
        [-1.0000,  1.0000, -0.9216,  ..., -1.0000,  1.0000, -1.0000],
        [ 1.0000,  1.0000, -1.0000,  ..., -1.0000,  1.0000,  1.0000],
        [-1.0000,  1.0000, -1.0000,  ..., -1.0000,  1.0000,  1.0000]],)
Bob's forward
Input c=tensor([[ 0.3812,  0.0767, -0.2101,  ...,  0.9363,  0.9419, -0.5616],
        [-0.1765,  0.3123, -0.1618,  ...,  0.9767,  0.9721, -0.8509],
        [ 0.5195,  0.3808,  0.2565,  ...,  0.9709,  0.9436, -0.7453],
        ...,
        [ 0.7632, -0.3302, -0.6372,  ...,  0.9582,  0.9808, -0.9196],
```

[0.0112, 0.4411, 0.4064, ..., 0.9851, 0.9606, -0.7699],
[0.0203, -0.1700, -0.2210, ..., 0.9621, 0.9494, -0.8102]],), k=tensor([[1., -1., 1., ..., -1., 1., 1.],
[1., 1., -1., ..., -1., 1., -1.],
[1., -1., 1., ..., 1., -1., -1.],
...,
[1., -1., -1., ..., 1., -1., 1.],
[-1., -1., 1., ..., 1., 1., -1.],
[1., 1., -1., ..., -1., -1., -1.]],)
P Bob tensor([[-1.0000, 0.6546, 1.0000, ..., 1.0000, -1.0000, -1.0000],
[-1.0000, -1.0000, -1.0000, ..., -1.0000, 0.9997, 1.0000],
[0.9894, 1.0000, 0.8372, ..., 1.0000, 1.0000, 1.0000],
...,
[-1.0000, -0.9997, 1.0000, ..., -1.0000, 1.0000, 0.9622],
[1.0000, 0.9915, -1.0000, ..., -1.0000, 1.0000, 1.0000],
[1.0000, -0.9999, -1.0000, ..., -1.0000, 1.0000, 1.0000]],)

AB = 0.5019 B = 0.4859 E = 0.8628 step 4635