Аналіз ефективних для активації архітектур нейронних мереж для задач машинного навчання

16 квітня 2025

Виконав: Захаров Дмитро Олегович[1]
Науковий керівник: Ігнатович Світлана Юріївна[2]

[1]Студент групи МП41 IV курсу (перший бакалаврський рівень), спеціальності 113 “Прикладна математика” освітньої програми “Прикладна математика”.

[2]Доктор фіз.-мат. наук, професор кафедри прикладної математики.

План

Вступ

Причини

Зараз, використання нейронних мереж є стандартом у багатьох задачах машинного навчання.

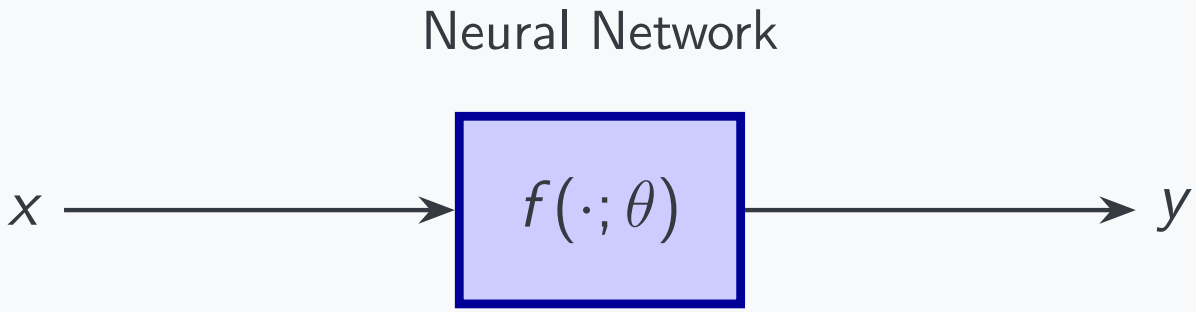


Рис.: Нейронна мережа є певною параметризованою функцією $f(\cdot;\theta)$, що перетворює вхідні дані x у вихідні y.

Найбільш поширена архітектура — це MLP (**M**ulti-**L**ayer **P**erceptron), що записується як:

$$\boldsymbol{x}^{(\ell+1)} = \sigma(\boldsymbol{W}^{(\ell)}\boldsymbol{x}^{(\ell)} + \boldsymbol{b}^{(\ell)}), \quad \ell \in [L].$$

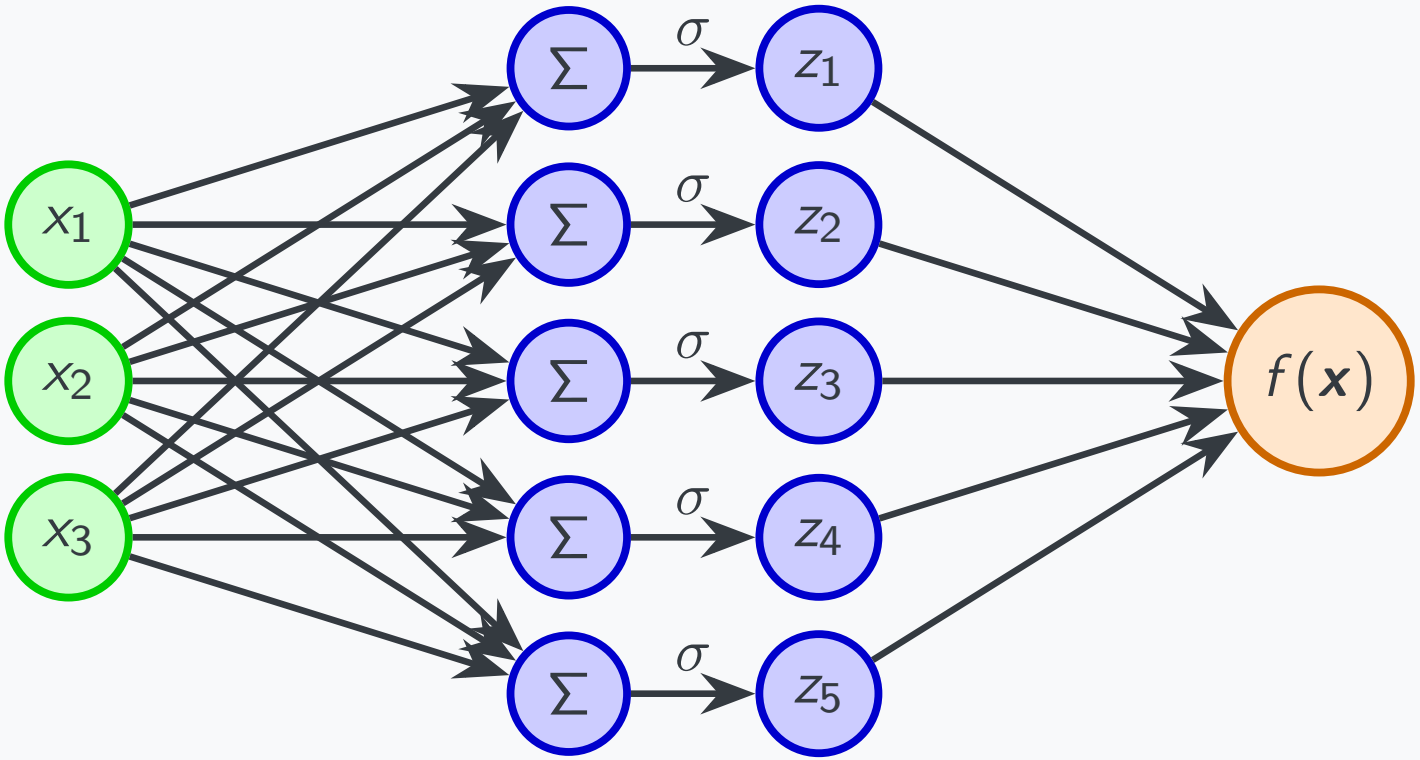
В такому разі, $f(\boldsymbol{x}) = \boldsymbol{x}^{(L)}$ для $\boldsymbol{x}^{(0)} = \boldsymbol{x}$.

Про силу нейронних мереж

Вхідний шар
m нейронів

Прихований шар
n нейронів

Вихідний шар
1 нейрон



Theorem (Universal Approximation Theorem)

Якщо σ не є поліномом та $L \geq 2$, то f є щільною в $L^{\infty}(K)$ для будь-якої компактної множини $K \subseteq \mathbb{R}^m$.

Безпека нейронних мереж

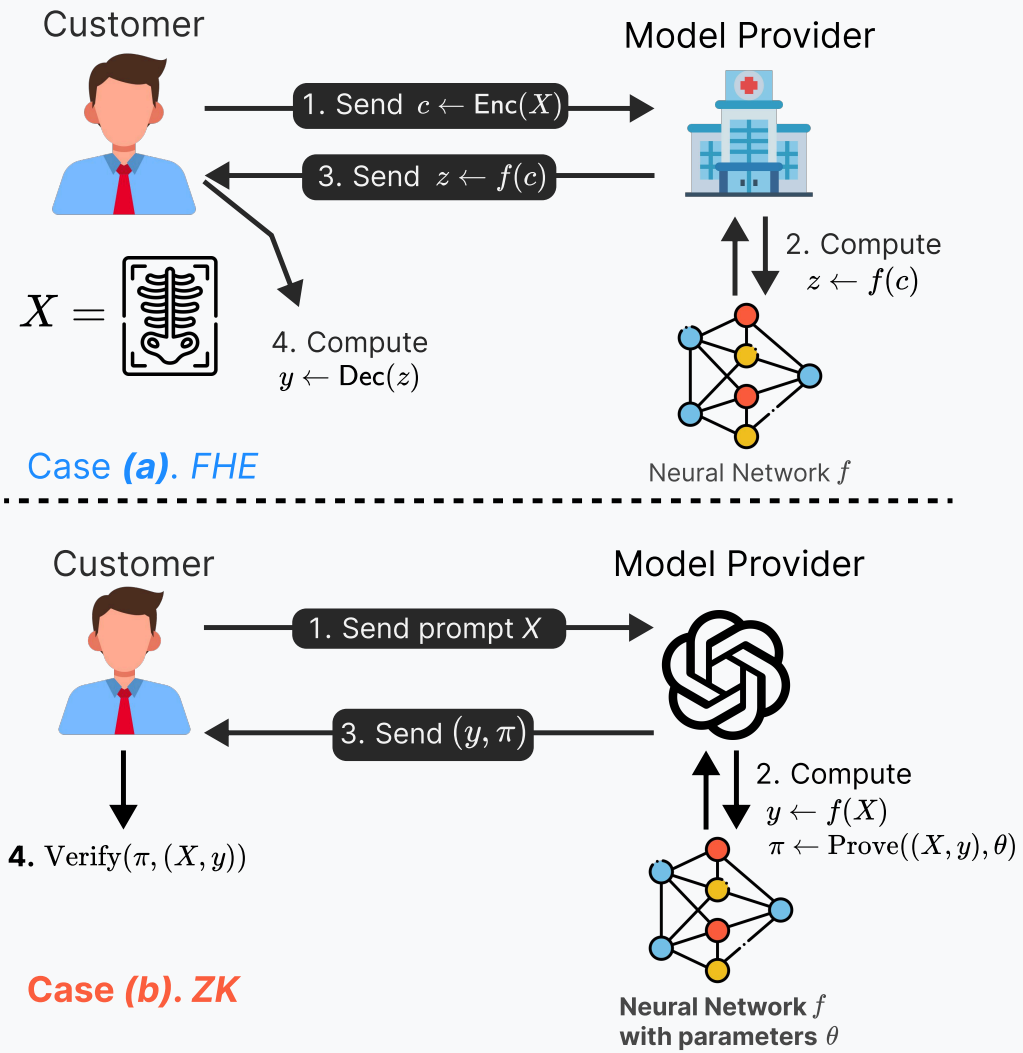


Рис.: Приклади використання нейронних мереж у сферах безпеки.

Безпека нейронних мереж

- Як ми можемо надіслати вхід x нейронній мережі f без розкриття x? Наприклад, чи може людина з зображенням рентгенівського знімка x надіслати його алгоритму f для діагностики, не показуючи знімок x (див. [5])?
- Чи можемо ми дійсно довіряти виходу нейронної мережі y? Інакшими словами, маючи вхід x, як ми можемо переконатись, що дійсно $y = f(x)$ (див. [2])?
- Як ми можемо довести, що f дійсно була натренована на неконфідеційних даних (див. [3])? Чи можемо ми тренувати нейронну мережу на зашифрованому наборі даних (див. [1])?

Зауваження

Першою та третьою задачами займається **гомоморфне шифрування** (FHE), а другою — **доведення нульового знання для алгоритмів машинного навчання** (ZKML).

Інструменти

Гомоморфне шифрування

- Гомоморфне шифрування — це шифрування, що дозволяє виконувати обчислення над зашифрованими даними без їх розшифрування.
- Це означає, що ми можемо виконувати обчислення на зашифрованих даних, не знаючи самих даних.

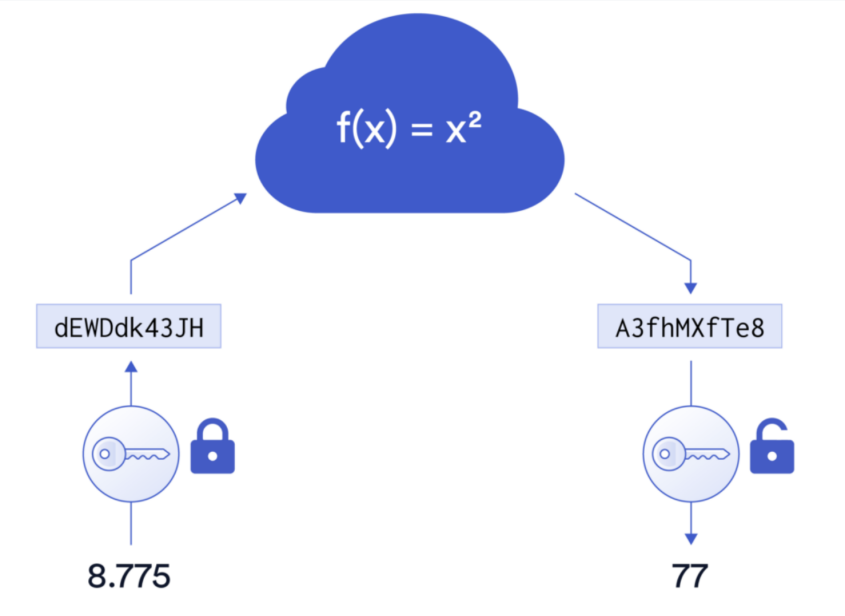


Рис.: Ілюстрація гомоморфного шифрування. Взято з https://chain.link/education-hub/homomorphic-encryption.

Гомоморфне шифрування: Означення

Definition (Схема повністю гомоморфного шифрування)

Схема повністю гомоморфного шифрування (FHE) — це алгоритми $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Eval})$:

- $\mathrm{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{ek})$: З огляду на параметр безпеки $\lambda \in \mathbb{N}$, видає секретний ключ sk, який зберігається в таємниці користувачем, та публічний ключ для обчислень ek.
- $\mathrm{Enc}(\mathsf{sk}, \mu) \to c$: З огляду на секретний ключ sk та повідомлення μ, видає шифротекст c.
- $\mathrm{Dec}(\mathsf{sk}, c) \to \mu$: З огляду на секретний ключ sk та шифротекст c, видає повідомлення μ.
- $\mathrm{Eval}(\mathsf{ek}, f, c_1, \ldots, c_\ell) \to \widetilde{c}$: З огляду на ключ для обчислень ek, функцію $f \in \mathcal{F}$ з підтримуваного класу функцій $\mathcal{F}$, та шифротексти $c_1, \ldots, c_\ell$, видає шифротекст результату застосування функції f до відповідних відкритих значень.

Гомоморфне шифрування: Властивості

Коректність: для будь-якої допустимої функції $f \in \mathcal{F}$ та вхідних повідомлень $\mu_1, \ldots, \mu_\ell$, маємо

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \widetilde{c}) = f(\mu_1, \ldots, \mu_\ell) \;\middle|\; \begin{array}{c} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c_j \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_j), j \in [\ell], \\ \widetilde{c} \leftarrow \mathsf{Eval}(\mathsf{ek}, f, c_1, \ldots, c_\ell) \end{array}\right] \geq 1 - \mu(\lambda),$$

де $\mu(\lambda)$ є нехтовною функцією: $\lim_{\lambda \to \infty} \mu(\lambda)\lambda^\gamma = 0$ для всіх γ.

Семантична безпека: для всіх повідомлень μ_0, μ_1 виконується

$$\left\{(\mathsf{ek}, c_0) : \begin{array}{c} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c_0 \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_0) \end{array}\right\} \approx_C \left\{(\mathsf{ek}, c_1) : \begin{array}{c} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c_1 \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_1) \end{array}\right\}$$

Тут $\approx_C$ означає обчислювальну еквівалентність.

Доведення нульового знання

- Доведення нульового знання — це протокол, що дозволяє одній стороні (доказнику) довести іншій стороні (перевіряючому), що вона знає певну інформацію, не розкриваючи цю інформацію.
- Це означає, що доказник може довести перевіряючому, що він знає секрет, не розкриваючи сам секрет.

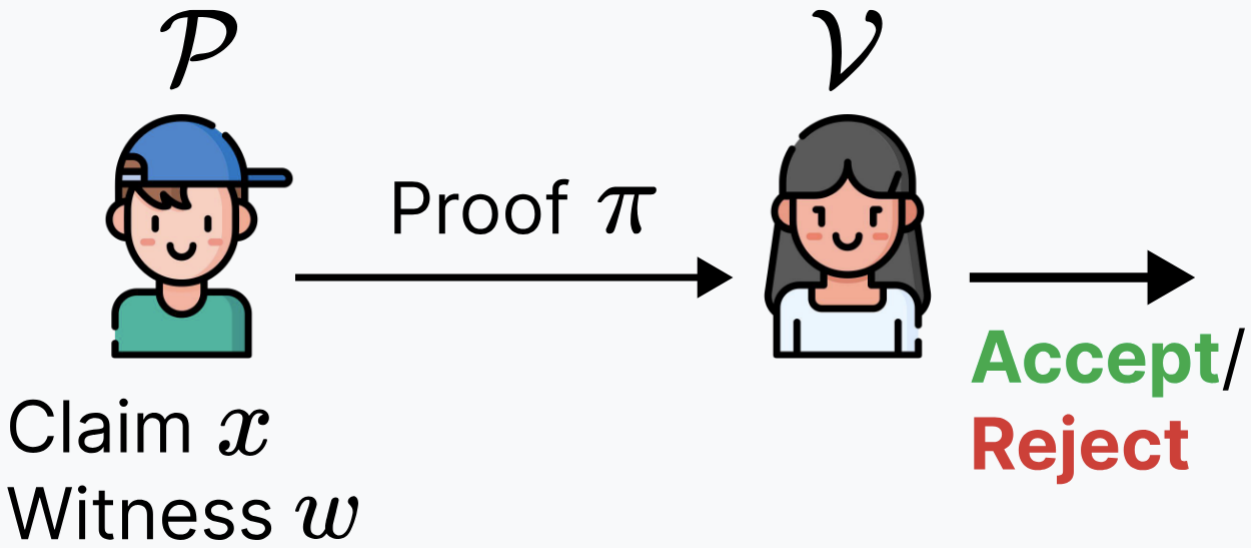


Рис.: Ілюстрація доведення нульового знання.

Доведення нульового знання: означення

Definition

zk-SNARK — це протокол доведення з нульовим розголошенням (zero-knowledge proof system), який дозволяє доводити знання свідка $\mathbb{w}$ для твердження $\mathbb{x}$ у відношенні $\mathcal{R}$ без розголошення самого свідка. zk-SNARK складається з таких алгоритмів:

- $\mathrm{KeyGen}(1^\lambda) \to (\mathsf{pp}, \mathsf{vp})$: алгоритм генерації ключів, що створює параметри для доведення pp та параметри для верифікації vp на основі параметра безпеки $\lambda \in \mathbb{N}$.
- $\mathrm{Prove}(\mathsf{pp}, \mathbb{x}, \mathbb{w}) \to \pi$: алгоритм доведення, що генерує доказ π для твердження $\mathbb{x}$ зі свідком $\mathbb{w}$ на основі параметрів доведення pp.
- $\mathrm{Verify}(\mathsf{vp}, \pi, \mathbb{x}) \to \{0, 1\}$: алгоритм перевірки, що перевіряє, чи доказ π є дійсним для твердження $\mathbb{x}$, і повертає біт 1, якщо доказ дійсний, і 0 — інакше.

Доведення нульового знання: властивості

- **Коректність:** для всіх $\mathbb{x}$ та $\mathbb{w}$ для яких $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ та $(\mathsf{pp}, \mathsf{vp}) \leftarrow \mathrm{KeyGen}(1^\lambda)$, $\Pr[\mathrm{Verify}(\mathsf{vp}, \pi, \mathbb{x}) = 1] = 1 - \mu(\lambda)$, де доведення генерується як $\pi \leftarrow \mathrm{Prove}(\mathsf{pp}, \mathbb{x}, \mathbb{w})$.
- **Непідробність:** для параметрів $(\mathsf{pp}, \mathsf{vp}) \leftarrow \mathrm{KeyGen}(1^\lambda)$ та $(\mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{vp})$ за умови $(\mathbb{x}, \mathbb{w}) \notin \mathcal{R}$, маємо $\Pr[\mathrm{Verify}(\mathsf{vp}, \pi, \mathbb{x}) = 1] = \mu(\lambda)$ для будь-якого поліноміально-випадкового алгоритму $\mathcal{A}$, що генерує π.
- **Нульове знання:** якщо $\mathrm{Verify}(\mathsf{vp}, \pi, \mathbb{x}) = 1$, то не існує алгоритму, який може дістати інформацію про $\mathbb{w}$ з π та $\mathbb{x}$ зі значною перевагою.
- **Ефективність:** зазвичай, розмір доведення $|\pi| = \mathcal{O}_\lambda(\log^\alpha n, |\mathbb{x}|)$ та час верифікації $T_\mathcal{V} = \mathcal{O}_\lambda(\log^\beta n, |\mathbb{x}|)$ (себто полілогарифмічна асимптотика) для деяких α, β.

Арифметичні ланцюги

Так чи інакше, щоб працювати над гомоморфно зашифрованими даними або протоколами доведення нульового знання, нам потрібно мати **арифметичні ланцюги** — це графи, що складаються з **вузлів** (операцій) та **проводів** (входів/виходів), які представляють обчислення над елементами (зазвичай скінченного) поля $\mathbb{F}_p$.

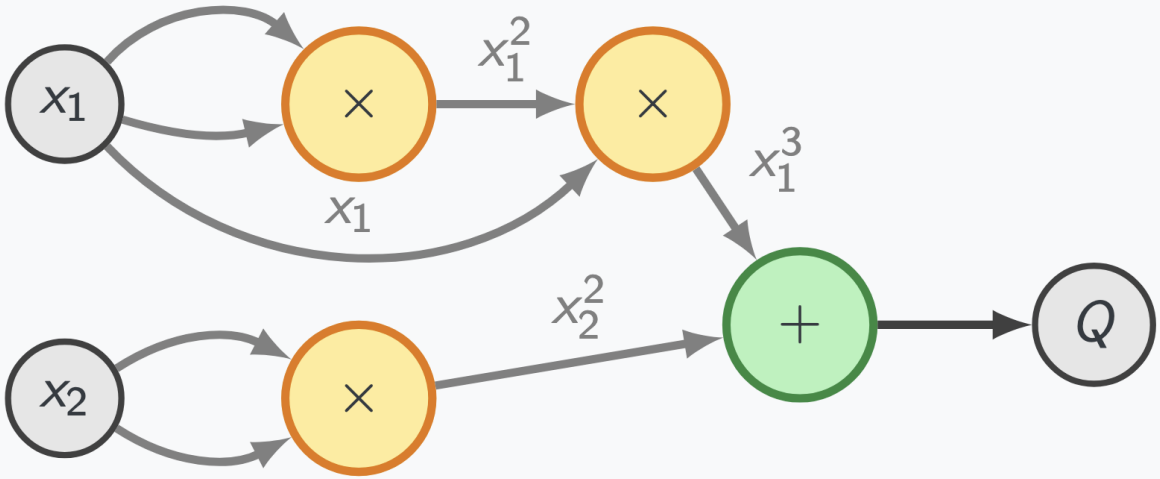


Рис.: Приклад арифметичного ланцюга для функції $x_1^3 + x_2^2$.

Нейронні мережі як арифметичні ланцюги

- Усі обчислення нейронної мережі (котрі проводяться над дійсним полем $\mathbb{R}$) можуть бути представлені арифметичними ланцюгами, що складаються з операцій **додавання** та **множення** над певним **полем** $\mathbb{F}_p$.
- Проте, кількість вузлів в такому графі дуже критично збільшує час на доведення та часто верифікації доведень для ZK та сильно збільшує час на обчислення для FHE (а іноді і накопичує помилки при обчисленнях).
- Тому, ми можемо спробувати зменшити кількість вузлів в графі, зберігаючи при цьому точність нейронної мережі.
- Це можна зробити за допомогою **активаційно-ефективних архітектур нейронних мереж**, які зберігають точність нейронної мережі, але зменшують кількість вузлів в графі за рахунок використання меншої кількості активаційних функцій.

Активаційно-ефективні архітектури нейронних мереж

Активаційні функції

Назва	Формула	Де використовується?
ReLU	$\max\{0, x\}$	Дешева нелінійність
LeakyReLU_α	$\max\{\alpha x, x\}$	Дешева нелінійність
Sigmoid $\sigma(x)$	$\frac{1}{1+e^{-x}}$	Бінарна класифікація
Tanh	$\frac{1-e^{-2x}}{1+e^{-2x}}$	Бінарна класифікація
Softmax	$\left\{ e^{x_i} / \sum_{j\in[n]} e^{x_j} \right\}_{i\in[n]}$	Мультикласифікація
Swish	$x\sigma(x)$	Ефективна нелінійність

Табл.: Найбільш поширені активаційні функції.

Питання

Як реалізувати такі активаційні функції, як ReLU, LeakyReLU, Sigmoid, Tanh, Softmax та Swish, у вигляді арифметичних ланцюгів, маючи лише операції додавання та множення?

Метод #1: Апроксимація

Ідея

Використовувати **апроксимацію** активаційних функцій поліномами, які можуть бути реалізовані у вигляді арифметичних ланцюгів.

Назва	Апроксимація
ReLU	$0.47 + 0.5x + 0.09x^2$
Tanh	$0.51x - 0.04x^3 + 0.0011x^5$
Swish	$0.24 + 0.5x + 0.1x^2$

Табл.: Апроксимація поширених активаційних функцій поліномами.

Проблеми

Проблеми

- Зазвичай, гарна апроксимація лише на певному відрізку, отже погіршується точність обчислень.
- А якщо тренувати модель на активаціях, що задані цими поліномами? Тоді не виконується теорема універсальної апроксимації, а отже емпірично губиться точність.
- Отже, краще використовувати активації як є.

Будь-які експоненти не можуть бути реалізовані у вигляді арифметичних ланцюгів, тому що вони не є поліномами. Тому,

- Ми використовуємо тільки ReLU та LeakyReLU з параметром $\alpha = 2^{-d}$, що є степінню двійки.
- Замість Softmax або Sigmoid (якщо це активація в кінці), використовуємо функції максимуму.

Вартість ReLU

Lemma

Обчислення ReLU *вартує* $b+1$ *множень в арифметичному ланцюзі, де* $b = \lceil \log_2 \mathbb{F} \rceil$ *— кількість бітів в елементах поля* $\mathbb{F}$.

Доведення. Треба розкласти x на двійкові компоненти $x = \sum_{i=0}^{b-1} x_i 2^i$ та переконатись, що $x_i \in \{0, 1\}$. Щоб перевірити бінарність x_i, достатньо зробити b множень $x_i(1 - x_i) = 0$. Далі якщо x_j позначає знак числа, то достатньо обчислити $x_j x$, що вартує 1 множення. Отже, загальна вартість $b + 1$ множень.

Коментар. Зазвичай додавання та множення на константу — дешева операція, тому не будемо її враховувати.

Повнозв'язаний шар

Нехай $\boldsymbol{x} \in \mathbb{F}^m$ — вхідний вектор, $W \in \mathbb{F}^{n \times m}$ — матриця ваг, $\boldsymbol{b} \in \mathbb{F}^n$ — вектор зсувів, $\boldsymbol{y} \in \mathbb{F}^n$ — вихідний вектор. Тоді

$$\boldsymbol{y} = \sigma(W\boldsymbol{x} + \boldsymbol{b}).$$

Вартість

- mn множень для обчислення $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$.
- nb множень для обчислення $\sigma(\cdot)$.

Отже, загальна вартість — $n(m + b)$ множень. Чи можна краще?

Encoder-Decoder Шар

Encoder-Decoder Шар

Шар Encoder–Decoder між n нейронами на вході та m нейронами на виході з h прихованими одиницями визначається наступним чином. Нехай $E \in \mathbb{F}^{n \times h}$ — матриця енкодера та $D \in \mathbb{F}^{h \times m}$ — матриця декодера. Тоді:

$$f(\boldsymbol{x}; D, E) := D\sigma(E\boldsymbol{x})$$

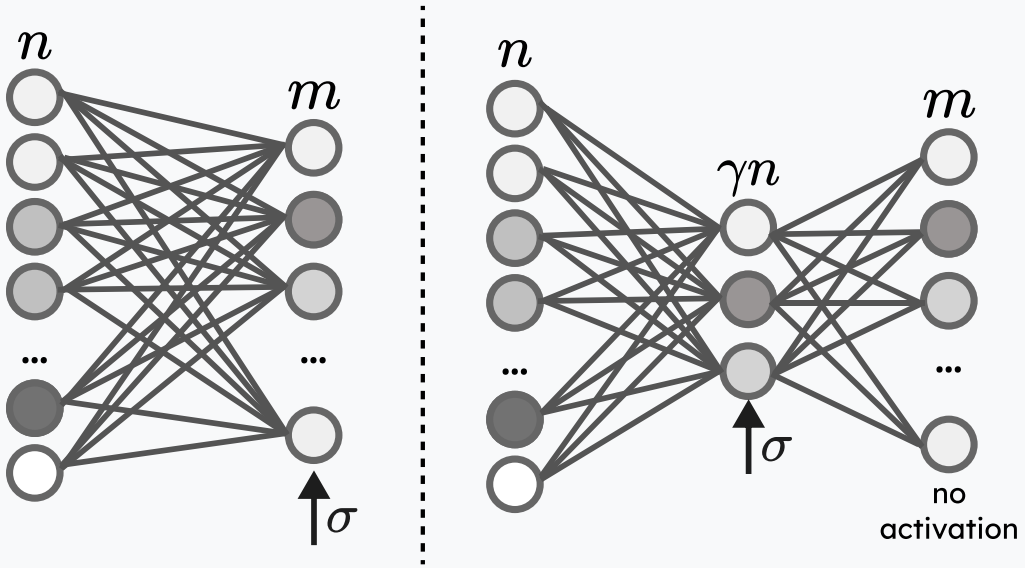


Рис.: Encoder-Decoder Шар.

Переваги Encoder-Decoder Шару

Lemma

Складність forward-pass шару дорівнює $\mathcal{O}(h(n+m+b))$ (на відміну від $O(n(m+b))$).

Example

Припустимо, що $n = 1000$, $m = 10000$, а $h = 10$. Нехай розмірність поля дорівнює $b = 254$. Тоді складність прямого проходу через нейронну мережу з одним шаром становить: $10000(1000 + 254) \approx 10^7$. Натомість, складність прямого проходу через нейронну мережу з шаром Encoder-Decoder дорівнює: $10(1000 + 10000 + 254) \approx 10^5$. Це дає **зменшення в 100 разів**, що є суттєвим!

Реалізація

```
def ed_model(hidden_units: int) -> tf.keras.models.Model:
    return tf.keras.models.Sequential([
        tf.keras.layers.Input(shape=(28,28,1)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(hidden_units, activation=None),
        tf.keras.layers.ReLU(),
        tf.keras.layers.Dense(100, activation=None),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
```

Рис.: Реалізація Encoder-Decoder Моделі.

Датасет

Рис.: Ми використовували Fashion-MNIST датасет [4].

Графік точності від фактору стискання

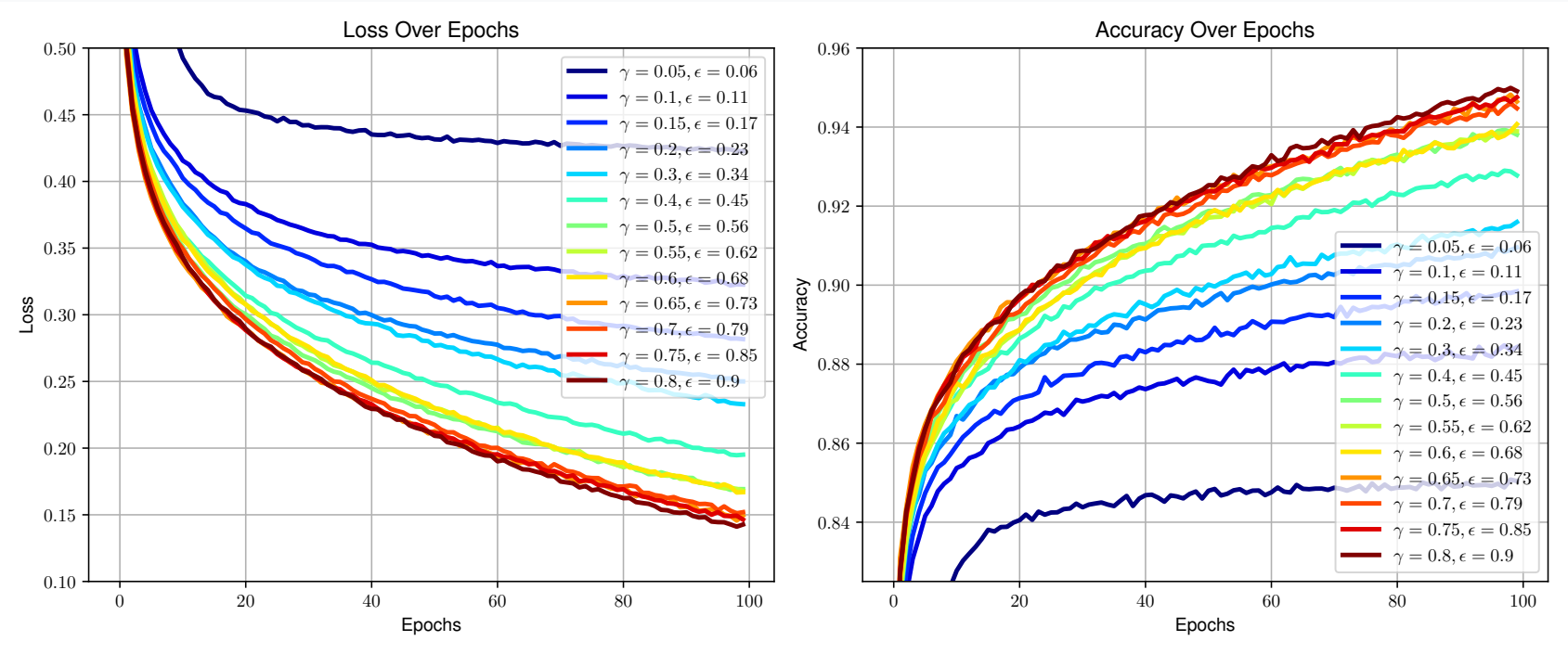


Рис.: Графік точності від фактору стискання.

Графік точності від фактору стискання

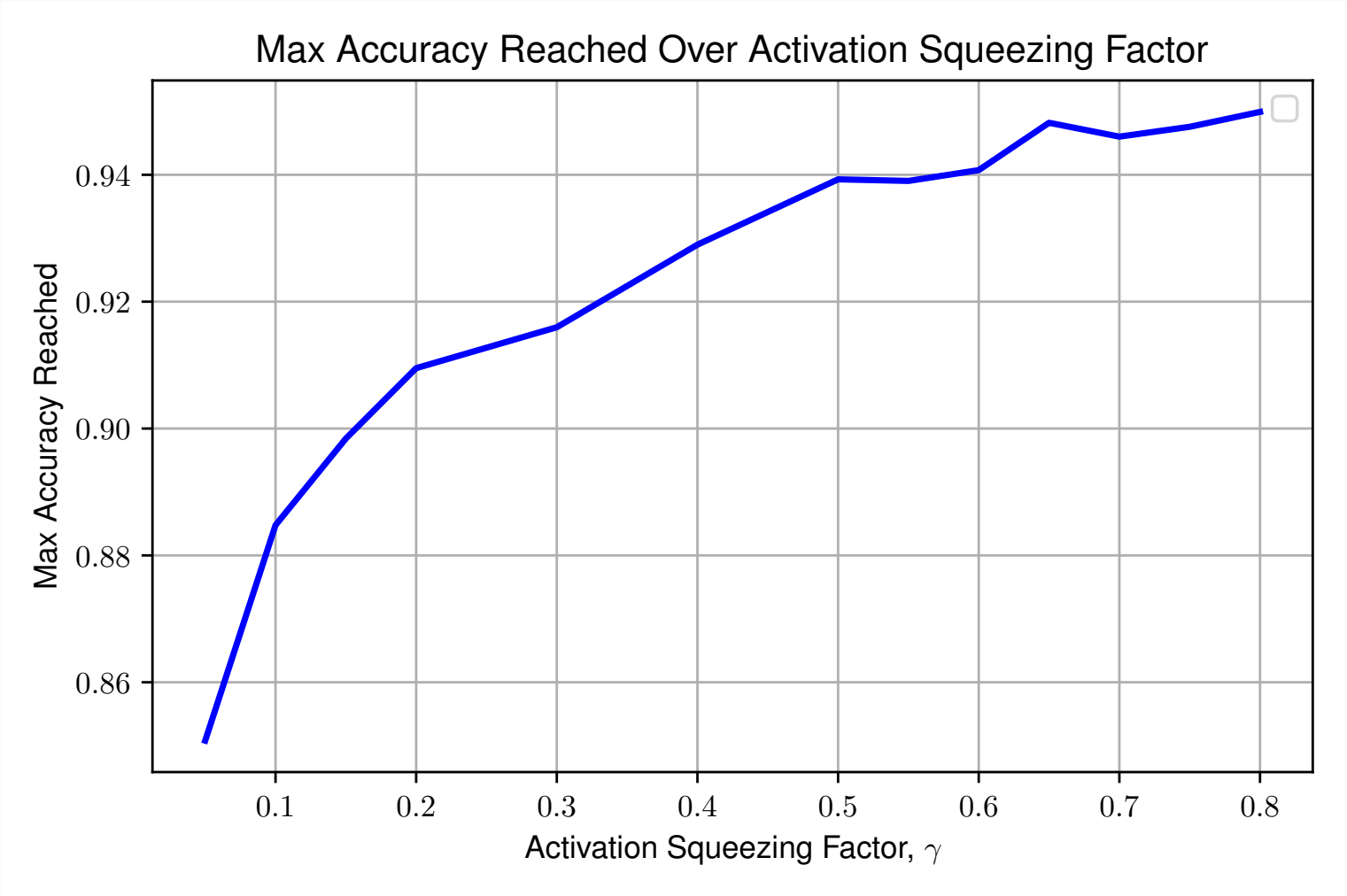


Рис.: Графік точності від фактору стискання.

Література I

[1] Luca Colombo, Alessandro Falcetta та Manuel Roveri. “Training Encrypted Neural Networks on Encrypted Data with Fully Homomorphic Encryption”. В: *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. WAHC ’24. Salt Lake City, UT, USA: Association for Computing Machinery, 2024, с. 64—75. ISBN: 9798400712418. DOI: 10.1145/3689945.3694802. URL: https://doi.org/10.1145/3689945.3694802.

[2] Tobin South та ін. *Verifiable evaluations of machine learning models using zkSNARKs*. 2024. arXiv: 2402.02675 [cs.LG]. URL: https://arxiv.org/abs/2402.02675.

[3] Haochen Sun та ін. *zkDL: Efficient Zero-Knowledge Proofs of Deep Learning Training*. Cryptology ePrint Archive, Paper 2023/1174. 2023. URL: https://eprint.iacr.org/2023/1174.

Література II

[4] Han Xiao, Kashif Rasul та Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. arXiv: 1708.07747 [cs.LG]. URL: https://arxiv.org/abs/1708.07747.

[5] Pengtao Xie та ін. *Crypto-Nets: Neural Networks over Encrypted Data*. 2014. arXiv: 1412.6181 [cs.LG]. URL: https://arxiv.org/abs/1412.6181.

Дякую за Вашу Увагу!