https://doi.org/10.31891/2219-9365-2025-83-26
УДК 004.89:004.42:004.415.25

БАБЕНКО Віталіна
Харківський національний університет імені В. Н. Каразіна
Харківський національний автомобільно-дорожній університет
https://orcid.org/0000-0002-4816-4579
e-mail: vita.babenko@gmail.com
ЛУЧЕНКО Сергій
Національний аерокосмічний університет "ХАІ"
https://orcid.org/0009-0006-9606-5774
e-mail: oleh.lantrat@gmail.com
БІЛИК Олександр
Харківський національний університет радіоелектроніки
https://orcid.org/0009-0007-4846-1585
e - mail: oleksandr.bilyk@nure.ua
ДРОЗДИК Євгеній
Харківський національний автомобільно-дорожній університет
https://orcid.org/0009-0001-0036-1194
e-mail: evgeniy.d97@gmail.com

ВІЗУАЛЬНА СИСТЕМА НАЛАШТУВАННЯ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ ТА ДАНИХ

У статті представлено розробку візуальної системи для проєктування та налаштування алгоритмів машинного навчання та даних, спрямовану на зменшення складності створення моделей глибокого навчання. Незважаючи на стрімкий розвиток штучного інтелекту та нейронних мереж упродовж останніх десятиліть, процес побудови та конфігурації моделей залишався доступним переважно лише для спеціалістів із ґрунтовними знаннями програмування. Дослідження вирішує цю проблему шляхом створення програмного забезпечення з відкритим кодом Learn2Learn, яке завдяки графічному інтерфейсу дозволяє користувачам створювати, налаштовувати та навчати моделі глибокого навчання практично без написання коду.

Робота підкреслює практичну значущість інтеграції графічних інтерфейсів користувача (GUI) у процеси машинного навчання. Це відкриває можливості для ширшої аудиторії – дослідників, викладачів і початківців – взаємодіяти з нейронними мережами більш інтуїтивно. Система структурована навколо ключових етапів машинного навчання: завантаження та передобробка даних, побудова архітектури моделей, вибір функцій втрат і алгоритмів оптимізації, а також моніторинг процесу навчання через візуалізацію метрик. На відміну від більшості існуючих інструментів, Learn2Learn підтримує інтеграцію користувацьких dataloader'ів і шарів нейронних мереж, що забезпечує гнучкість, співмірну з традиційним програмуванням, водночас значно знижуючи поріг входу.

У статті детально розглянуто реалізовані функціональні можливості: візуальне проєктування моделей за допомогою drag-and-drop, інтерактивне налаштування параметрів, відображення повідомлень про помилки у зручному форматі та інтегровані підказки щодо вибору архітектурних рішень. Система підтримує різні типи даних – зображення, текстові та числові дані – і забезпечує їхню передобробку завдяки аугментації. Використання PyTorch як обчислювальної основи та PyQt6 для створення інтерфейсу дозволило поєднати простоту використання та технічну строгість.

Порівняльний аналіз з аналогічними візуальними інструментами засвідчив переваги Learn2Learn у гнучкості, розширюваності та зручності відлагодження моделей. Особлива увага приділяється освітній складовій: система не лише спрощує налаштування моделей, але й допомагає інтерпретувати результати навчання. У висновках зазначається, що створений прототип суттєво знижує бар'єри для входу у сферу глибокого навчання, об'єднуючи доступність графічних інтерфейсів із можливостями традиційного програмування. Перспективи подальших досліджень полягають у розширенні функціоналу, додаванні стандартних датасетів, попередньо навчених моделей і підтримки розподіленого навчання.

Ключові слова: машинне навчання, глибоке навчання, графічний інтерфейс, нейронні мережі, передобробка даних, візуалізація, Learn2Learn.

BABENKO Vitalina
V. N. Karazin Kharkiv National University
Kharkiv National Automobile and Highway University
LUCHENKO Sergiy
National Aerospace University – Kharkiv Aviation Institute.
BILYK Oleksandr
Kharkiv National University of Radio Electronics
DROZDYK Yevhenii
Kharkiv National Automobile and Highway University

VISUAL SYSTEM FOR SETTING UP MACHINE LEARNING ALGORITHMS AND DATA

The article presents the development of a visual system for designing and configuring machine learning algorithms and datasets, aimed at reducing the complexity of building deep learning models. Despite the rapid growth of artificial intelligence and

neural networks in recent decades, the creation and configuration of models have remained accessible only to specialists with strong programming skills. This work addresses the gap by developing Learn2Learn, an open-source software tool with a graphical interface that allows users to construct, customize, and train deep learning models almost entirely without coding.

The study emphasizes the practical relevance of integrating graphical user interfaces (GUIs) into machine learning workflows, enabling a broader range of users, including researchers, educators, and beginners, to interact with neural networks more intuitively. The system is structured around key stages of machine learning: data loading and preprocessing, model construction, selection of loss functions and optimization algorithms, and monitoring of training progress through visualized metrics. Unlike most existing tools, Learn2Learn supports the integration of custom dataloaders and model layers, ensuring flexibility comparable to traditional coding while significantly lowering the entry threshold.

The article provides an overview of implemented functionalities: visual model construction using drag-and-drop neural network layers, interactive parameter adjustment, real-time error visualization, and integrated recommendations for model design. The program supports diverse data types, including images, text, and numerical values, and allows preprocessing through augmentation techniques. By combining PyTorch as the computational backbone with PyQt6 for GUI design, the authors demonstrate how the system maintains both usability and technical rigor.

A comparative analysis with existing visual tools highlights the advantages of Learn2Learn in flexibility, expandability, and error handling. In particular, user-friendly error messages and interactive hints help prevent common mistakes, making the system not only a development tool but also an educational platform. The authors emphasize that Learn2Learn is still at a prototyping stage, with future improvements planned, such as integration of standard datasets, pre-trained models, and distributed training on remote servers.

The article concludes that the developed system significantly reduces barriers to entry into deep learning by providing an accessible, extensible environment for model creation. The prototype illustrates the feasibility of unifying the simplicity of visual interfaces with the flexibility of code-based programming, opening prospects for both educational applications and practical research in artificial intelligence.

Keywords: machine learning, deep learning, graphical interface, neural networks, data preprocessing, visualization, Learn2Learn.

Стаття надійшла до редакції / Received 04.08.2025
Прийнята до друку / Accepted 24.08.2025

ПОСТАНОВКА ПРОБЛЕМИ В ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

За останні кілька років штучний інтелект і нейромережі проникли практично у всі сфери життя. І хоча теоретичні викладки в цій галузі з'явилися вже давно, можливість створити програми, що мають практичне застосування, з'явилася порівняно недавно, коли на початку двохтисячних років накопичилася достатня кількість даних, а також досягли певного рівня обчислювальні потужності та їх вартість.

Нейросети використовуються дуже широко, ви безпосередньо взаємодієте з ними, коли шукайте що-небудь у пошуковій системі, дізнаєтеся маршрут до потрібної вам точки (з урахуванням пробок, звичайно), перевіряєте прогноз погоди на найближчі дні, заїжджаєте на паркування з розпізнаванням номерів, а якщо пощастить, то можете навіть поговорити з нейромережею. Це найпростіші приклади, з якими ми стикаємося щодня. Не менш широко нейромережі використовуються і в професійних сферах: соціології, журналістиці, лінгвістиці та багатьох інших областях.

У той же час практично всі сучасні веб-сервіси, комп'ютерні програми та мобільні програми базуються на графічному інтерфейсі. Наступний за популярністю інтерфейс - командний рядок, що використовується тільки в конкретних областях, найчастіше пов'язаних із програмуванням. Це досить логічно, враховуючи той факт, що сама робота в командному рядку багато в чому і є програмуванням, проте таку навичку мають далеко не всі, адже для її отримання потрібно кілька місяців, якщо не років, копіткого вивчення предмета.

Незважаючи на високу затребуваність нейромереж, єдиним способом їхньої побудови залишається програмування. У цій роботі ми досліджуємо можливості інтеграції альтернативного способу роботи з нейромережами - графічного інтерфейсу, а також постараємося розібратися, чому такі програми не з'явилися раніше.

ПОСТАНОВКА ЗАДАЧІ

Глобальна мета цієї роботи — створення програми з відкритим кодом, що дозволяє створювати та налаштовувати будь-які моделі глибокого навчання з учителем з нуля, яка не вимагає попередньо нічого, крім набору даних. Однак, глибоке навчання стрімко розвивається вже не один десяток років і за цей час було вигадано безліч трюків та інструментів і реалізація їх усіх потребує дуже багато часу, оскільки весь код автор пише наодинці. Тому завдання цієї роботи - створення прототипу універсального додатка з графічним інтерфейсом, демонстрація принципової можливості додавання більшості інструментів, а також виявлення тонких місць, які потребують додаткового опрацювання.

Вирішення кожної задачі в машинному навчанні можна розбити на три смислові частини, які вирішуються практично окремо один від одного: створення dataloader'a - класу, що видає дані у потрібному форматі, написання нейромережевої моделі, що приймає на вхід дані з dataloader'a, а також підбір метрик та аналіз їх графіків та результатів роботи моделі. Розглянемо кожне підзавдання докладніше.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

Сучасні дослідження в галузі машинного навчання свідчать про зростаючу потребу у створенні інструментів, що спрощують процес побудови та налаштування моделей для користувачів без глибокої програмної підготовки. Значна частина робіт [1] зосереджується на розробці графічних інтерфейсів користувача (GUI), які дозволяють будувати нейронні мережі у візуальній формі. Такі рішення роблять процес налаштування архітектур більше інтуїтивним і доступним, особливо в навчальних та дослідницьких цілях.

Значний вклад у розвиток подібних систем зроблено в межах проекту **Barista** [2], який забезпечує середовище для проектування та тренування глибоких нейронних мереж за допомогою блок-схем. Подібні підходи активно застосовуються і в освітньому середовищі - наприклад інструмент для викладання основ згорткових нейронних мереж у медицині [3].

Окремі дослідження зосереджуються на створенні пояснюваних систем, які не тільки спрощують налаштування моделей, але й допомагають інтерпретувати їх результати [4]. Це особливо актуально для критичних галузей, де потрібна прозорість процесу прийняття рішень.

Разом із тим, більшість існуючих інструментів мають обмежень функціонал : вони або орієнтовані на вузьке коло завдань, або не дозволяють інтегрувати нові елементи архітектури без використання коду Це створює передумови для подальших досліджень у напрямку універсальних систем, які б поєднували гнучкість традиційного програмування з простотою використання графічних інтерфейсів.

Робота з даними. Існує безліч способів зберігання даних. Для кожного з них потрібний свій Dataloader, що враховує специфіку їхнього зберігання [5]. Однак, на практиці, в більшості завдань зустрічаються лише кілька різних типів, ось деякі з них:

- І вхідні дані, і розмітка зберігаються в одному текстовому файлі
- Дані та розмітка зберігаються в різних файлах
- Дані є безліч файлів (наприклад, зображень або документів), а розмітка зберігається в окремому текстовому файлі
- Дані є безліч файлів, які розташування визначає розмітку (наприклад, зображення різних класів лежать у різних папках)

Також потрібно враховувати тип даних, оскільки різні типи можуть вимагати різну передобробку [6]. Так, наприклад, якщо зображення необхідно призвести до певного розміру перед тим, як передати на обробку моделі, то для текстових документів таке перетворення невизначене, хоча методи зберігання можуть збігатися [7]. Найчастіші з типів також можна коротко перерахувати:

- Чисельні характеристики (речові, цілочисленні), рядки з малою кількістю значень (класи), а також їх списки
- Зображення
- Відео
- Текстові документи
- Аудіо

Також на пристрій Dataloader'a впливає і тип завдання, що визначає тип розмітки [8]. Їх уже не так багато, хоч регулярно з'являються нові.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

Глобальне завдання: "вшити" кожен dataloader з найбільш часто зустрічаються в програмі. Однак, передбачити всі можливі способи зберігання даних і типи завдань ніяк не вийде (як мінімум, оскільки з'являються нові типи завдань). Тому в цій роботі ми додамо можливість користувачеві завантажувати в програму власний dataloader за допомогою вбудованого текстового редактора.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

1.1. Створення моделі

Модель у глибокому навчанні - це складна, нелінійна функція, що є композицією простих базових функцій - шарів. Серед них зазвичай виділяють окремо нелінійності та регуляризацію [9], але в цій роботі ми будемо розглядати їх нарівні зі звичайними шарами, наприклад лінійним, згортковим і т.п.

Нейронні мережі можна представити як орієнтований граф, з деякими «особливими» вершинами, інцидентними ребрам, що входять «з нізвідки» або «в нікуди», саме так зазвичай візуалізують нейромережі і в наукових працях. Ребра «з нізвідки» - це найперші шари, які отримують на вхід дані з dataloader'a, а ребра «в нікуди» передають свій вихід у функцію втрат.

Така візуалізація дійсно дуже зручна і дозволяє легко зрозуміти як влаштована модель, якщо знати, як влаштований кожен шар окремо. Саме так і виглядає редактор моделі в нашій програмі : На полотні, яке можна переміщати, розміщуються прямокутники-шари, вибравши які можна налаштувати параметри, подивитися на розмірність входу і виходу, або докладніше вивчити його пристрій, прочитавши опис шару.

За аналогією з попередньою секцією, для того щоб додаток не поступався в гнучкості традиційному програмуванню моделей, можна додати функцію створення своїх власних шарів, функцій втрат і алгоритмів оптимізації [10]. Для цього, по суті, потрібно повторити функціонал dataloader'a, підлаштувавши його під відповідну специфіку. Однак, у цій роботі ця функція додана не буде: принципова можливість додавання продемонстрована, а реалізовувати функцію довелося б на шкоду іншим можливостям прототипу.

1.2. Інтерпретація метрик

Навчання нейронної мережі - це завдання оптимізації деякої функції, а саме функції втрат, яка підбирається залежно від завдання. Найчастіше використовуються одні й самі функції втрат, наприклад, для завдань класифікації чи регресії часто-густо використовуються квадратичне середнє чи крос ентропія [12]. Тому необхідно додати якнайбільше «популярних» функцій втрат до бібліотеки програми, щоб їх можна було використовувати без застосування програмування.

Після вибору функції втрат починається навчання нейромережі. Найчастіше, якщо десь були допущені помилки і модель навчається некоректно, це стає зрозуміло за поведінкою функції втрат або інших метрик. Тому під час навчання практично завжди прописують побудову графіків до різних метрик [13]. У нашій програмі також прописано автоматичну побудову та оновлення графіків у процесі навчання.

2. Обґрунтування

Типова реалізація алгоритму навчання на PyTorch має такий вигляд: (Алгоритм 1).

Algorithm 1 Схема алгоритму навчання нейромережі на PyTorch

dataloader = dataloader_ init ( ) model = model_init () loss = loss_init ()
optim = optim_init (model.params) for data in dataloader out = model(data) loss_value = loss(out) loss_value.backward () optim.step () update_plots (loss_value) end for

Зауважимо, що ініціалізації об'єктів тут ніяк не залежать один від одного (крім об'єкта-оптимізатора, єдине обмеження якого полягає в необхідності бути ініціалізованим після ініціалізації моделі нейромережі), що дозволяє налаштовувати їх незалежно один від одного. Саме за рахунок цієї властивості, графічний інтерфейс має великий потенціал: кожну деталь алгоритму можна настроювати окремо у своєму вікні, спроектованому з урахуванням специфіки своєї деталі.

Крім того, графічний інтерфейс має багато переваг у порівнянні з традиційним підходом. Ось деякі з них:

2.1. Доброзичливі повідомлення про помилки

Головна слабкість графічного інтерфейсу — менша гнучкість порівняно з чистим програмуванням, дає тим щонайменше деякі переваги, адже кількість місць, де модель потенційно може зламатися, як і кількість причин поломки також зменшується [14]. Завдяки цьому з'являється можливість перебрати всі можливі місця, де можуть з'являтися помилки та скласти коментарі, які пояснюють користувачеві, що він зробив не так і що необхідно зробити для коректної роботи нейромережі.

Подібні повідомлення, звичайно, є і в програмуванні, але їх формат часто далекий від доброзичливого: користувачеві показується місце, на якому програма завершила роботу, але причина поломки може знаходитися в зовсім іншому місці.

2.2. Інтерактивність

Графічний інтерфейс також дозволяє взаємодіяти з моделлю перед тим, як модель буде готова до використання. Цей підхід дозволяє виводити інформацію про модель безпосередньо під час побудови (наприклад, сумарна кількість параметрів моделі, розмір тензора на виході з кожного шару тощо) [15]. Ця інформація також полегшує пошук помилок, дозволяючи їх виправляти ще до запуску нейромережі.

Крім підказок про помилки, програма може давати поради щодо використання. Наприклад, у яких випадках варто використовувати той чи інший шар нейромережі, нелінійність або регуляризацію, давати посилання на теоретичні обґрунтування, а також описувати гіперпараметри, на що вони впливають і як правильно їх налаштувати.

2.3. Малий поріг входу

При правильному виконанні описані вище переваги дозволяють значно знизити поріг входу. Відсутність необхідності програмувати вже сама по собі робить це, а за наявності в програмі посилань і пояснень зменшується кількість необхідних технічних знань, адже частину необхідної інформації можна отримати в самому додатку.

Безумовно, зовсім непідготовлений користувач навряд чи впорається з написанням нетривіальної нейромережі, але зрозуміти їх пристрій буде набагато легше, якщо не доведеться відволікатися на програмування, а вся необхідна теорія міститься в одному місці.

3. Про програму

Learn2Learn повністю написана мовою Python, графічний інтерфейс реалізований за допомогою набору розширень для графічного фреймворку Qt, а саме PyQt6. Побудова та робота з моделлю практично

повністю відбувається на основі бібліотеки PyTorch, у кількох місцях також використовуються бібліотеки scikit-learn та albumentations

3.1. Вікно створення та відкриття проектів

При запуску програми відкривається стартове вікно, в якому можна створити новий проект, або відкрити існуючий. Проект являє собою папку в певній директорії, що містить файл конфігурації проекту config.json, що містить інформацію про проект. Під час створення нового проекту, відповідний файл генерується виходячи з значень головного файлу конфігурації програми: main_ config.json. Для роботи з файлами конфігурації було створено спеціальний клас Config. Він влаштований таким чином, що при зміні значення будь-якого параметра ці зміни автоматично зберігаються у відповідний файл.

3.2. Вікно проекту

Вікно проекту ділиться на дві частини: основну та контейнер меню. Більшу частину екрана займає основна, вона ділиться на п'ять вкладок: Налаштування, Розмітка даних, Дані, Модель, Навчання, у кожний момент часу активна одна вкладка. У правій частині екрана розташований контейнер меню, який містить меню останнього об'єкта, з яким взаємодіяв користувач.

Для зручності розробки та використання програми були реалізовані два класи: MenuWidget та WidgetWithMenu. Для кожного віджету, що успадковується від WidgetWithMenu, можна реалізувати свій клас, що успадковується від MenuWidget і передати його як параметр при виклику функції super, разом з екземпляром класу-контейнера. Тоді при викликі методу activate_menu у відповідному контейнері меню відображатиметься меню даного віджету. Так, кожна вкладка має власне меню і при перемиканні вкладок відкривається меню тієї, що була відкрита.

3.3. Вкладка «Налаштування»

Дозволяє змінювати деякі значення у файлі конфігурації проекту.

3.4. Вкладка «Розмітка даних»

У цій вкладці можна розмітити зображення для виявлення об'єктів.

3.5. Вкладка «Дані»

Має два підрозділи: Завантажувач даних та Передобробка

3.5.1. Вкладка «Завантажувач даних»

Завантажувач даних є вбудованим текстовим редактором з підсвічуванням синтаксису python. Для створення dataloader'a традиційним способом програміст створює спеціальний клас, що успадковується стандартного класу Dataset з модуля torch.utils.data. Цей клас має визначити такі методы:

- __ init __ — метод, який відповідає за ініціалізацію даних
- __ getitem __ — метод, який повертає елемент датасету за індексом
- __ len __ — метод, який повертає розмір датасету

Крім того, існує опція завантаження коду з файлу: для цього потрібно натиснути кнопку *Завантажити* і вибрати бажаний файл.

3.5.2. Вкладка «Переробка»

Для аугментації передбачено вкладку Передобробка. У лівій частині основного екрана відображається необроблене зображення, яке можна вибрати, натиснувши на відповідну кнопку. Праворуч розміщено зображення після обробки. Між ними є простір, в якому розташовуються перетворення, що застосовуються. Ці перетворення можна додавати та видаляти за допомогою відповідних кнопок у меню.

На даний момент реалізовані такі перетворення:

- InvertImg — Застосовує до зображення «негатив», віднімаючи значення кожного пікселя з 255
- Equalize — Наближає розподіл значень пікселів до рівномірного, таким чином збільшуючи контраст
- CLAHE — Застосовує локальну адаптивну гістограму
- ToSepia - Застосовує фільтр сепії
- GaussianBlur — Застосовує фільтр гауса
- HueSaturationValue — Випадково змінює тон, насиченість та яскравість кожного пікселя.
- RandomContrast — Випадково змінює контраст зображення
- Resize — Змінює розмір зображення

Під час вибору віджету кожного перетворення в меню праворуч з'являється налаштування параметрів відповідного перетворення. Натискання кнопки «Оновити» проганяє вхідне зображення через задані перетворення та оновлює трансформоване зображення.

Для збереження поточного dataloader'a та застосовуваних шарів передобробки необхідно натиснути на відповідну кнопку в самому низу екрану

3.6. Вкладка «Модель»

Містить дві підвкладки : Конструктор та Параметри навчання. Перша призначена для створення нейромережі, а в другій можна вибрати функцію помилки та алгоритм оптимізації.

3.6.1. Вкладка «Конструктор»

В основній частині екрана розташоване полотно. Розміри полотна не дозволяють повністю відобразити його на екрані, тому в кожний момент часу відображається лише його частина. Цю частину можна переміщати за допомогою миші або повзунків, розташованих праворуч і знизу від полотна. Меню вкладки містить кнопки, що дозволяють створювати на ньому шари нейромережі (рис. 1). За умовчанням на полотні розташовані вхідний і вихідний шари, перший передає в нейромережі дані, а другий отримує на вхід результат роботи нейромережі, на відміну від інших шарів їх неможливо видалити.

Шар виглядає на полотні як прямокутник фіксованого розміру. Посередині розташований текст виду <тип шару >_ <id>. У верхній та нижній частинах шару розташована певна кількість кнопок, що залежить від типу шару. Верхні кнопки уособлюють вхідні дані шару, нижні вихідні. Для створення зв'язку між двома шарами потрібно спочатку натиснути одну з кнопок вихідних даних першого шару, а потім кнопку вхідних даних у другого. Після цього між відповідними кнопками проводиться лінія, що позначає зв'язок між шарами. Кожна вхідна кнопка може приймати не більше одного зв'язку, таким чином між сполучними лініями і кнопками входу утворюється ін'єкція і для вибору певного зв'язку достатньо натиснути на відповідну їй кнопку. Кожен шар зберігає інформацію, з якими шарами він пов'язаний, завдяки чому модель будується за лінійний час.

Кожен шар можна переміщати відносно полотна, натиснувши на нього і тримаючи ліву кнопку миші затиснутою, переміщуючи в потрібне місце. При виборі шару у контейнері меню, розташованому праворуч, відкривається меню налаштування параметрів цього шару. На даний момент у програму вбудовані такі типи шарів:

1. Шари – функції активації (Non-linear activations) [16]

Такі шари отримують на вхід тензор $X = (x_i)$, $i \in N^k$ та повертають тензор $(f(x_i))$, $i \in N^k$, де f - функція активації, також звана нелінійністю

- ReLU - $f(x) = \max(x, 0)$
- Sigmoid - $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

2. Згорткові шари [17] (Convolution layers) Застосовують до вхідного тензора n -мірну згортку. Вхідний тензор може бути розмірності $n + 1$ чи $n + 2$. Проходить вікном розміру $((1), C_{in}, k_1, k_2, \ldots, k_n)$, також званим ядром по всьому вхідному *тензору* разом з відступами розміру $((0), 0, p_1, p_2, \ldots, p_n)$ з кроком $((1), , , , ,)$ застосовуючи лінійне перетворення значення вікна. Між елементами ядра може бути додані проміжки, що залежать від осі $((0), 0, d_1, d_2, \ldots, d_n)$ Якщо на вхід подається тензор розміру $((N), C_{in}, D_1, D_2, \ldots, D_n)$, вихідний тензор буде мати $((N), C_{out}, D'_1, D'_2, \ldots, D'_n)$, где D'_i:

$$D'_i = \left\lfloor \frac{D_i + 2 \cdot p_i - d_i \cdot (k_i - 1) - 1}{s_i} + 1 \right\rfloor$$

Параметри:

- in_channels (C_{in}) - Число каналів вхідного тензора
- out_channels (C_{out}) — кількість каналів вихідного тензора
- kernel_size * ($k_1, \ldots, k_n$) - розмір ядра
- stride * ($s_1, \ldots, s_n$) - крок ядра по кожній осі
- padding * ($p_1, \ldots, p_n$) - Розміри відступів по кожній осі
- padding_mode - спосіб заповнення відступів. Може приймати 4 значення:

∗ *zeros* - Заповнює відступи нулями:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

∗ *reflect* - Відображає значення відступів щодо меж тензора

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 6 & 5 & 4 & 5 & 6 & 5 & 4 \\ 3 & 2 & 1 & 2 & 3 & 2 & 1 \\ 6 & 5 & 4 & 5 & 6 & 5 & 4 \\ 3 & 2 & 1 & 2 & 3 & 2 & 1 \end{pmatrix}$$

∗ *replicate* — кожне значення відступу дорівнює найближчому елементу з основного тензора

$$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix} \to \begin{pmatrix}1 & 1 & 1 & 2 & 3 & 3 & 3\\ 1 & 1 & 1 & 2 & 3 & 3 & 3\\ 4 & 4 & 4 & 5 & 6 & 6 & 6\\ 4 & 4 & 4 & 5 & 6 & 6 & 6\end{pmatrix}$$

∗ *circular* - Заповнює відступи по колу вздовж кожної осі

$$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix} \to \begin{pmatrix}5 & 6 & 4 & 5 & 6 & 4 & 5\\ 2 & 3 & 1 & 2 & 3 & 1 & 2\\ 5 & 6 & 4 & 5 & 6 & 4 & 5\\ 2 & 3 & 1 & 2 & 3 & 1 & 2\end{pmatrix}$$

- dilation * ($d_{1},..., d_{n}$) - Розмір проміжків між елементами ядра
- groups — кількість груп, на які розбиваються вхідні та вихідні канали
- bias — якщо *True,* додає до виходу зміщення, що навчається.

* — параметр може приймати на вхід одне число *a,* що інтерпретуватиметься як вектор розміру n : ($a,...,a$)

- Conv1d - Одновимірна згортка
- Conv2d - Двовимірна згортка

3. • Linear — Повнозв'язний лінійний шар: застосовує лінійне перетворення до вхідних даних, $y = xA^T + b$. Якщо на вхід подається тензор розмірності ($d_{1},..., d_{n},H_{in}$), на виході вийде тензор розмірності ($d_{1},..., d_{n},H_{out}$), відповідно матриця A має розмір $H_{in} \times H_{out}$ Параметри:
   - in_features (H_{in}) - розмір вхідних даних
   - out_features (H_{out}) - розмір вихідних даних
   - bias — Якщо *True,* додає до виходу зміщення, що навчається.
4. Об'єднуючі шари (Pooling layers) застосовують до вхідного тензора n -вимірне об'єднання за певним правилом. Вхідний тензор може бути розмірності $n + 1$ чи $n + 2$. Проходить вікном розміру ((1),C,k_{1},k_{2},..., k_{n}), також званим ядром по всьому вхідному *тензору* разом з відступами розміру ((0), 0,p_{1},p_{2},..., p_{n}) з кроком ((1),,,,, застосовуючи перетворення до значень вікна, у разі перетворення — операція взяття максимуму за значенням вікна (max pooling). Між елементами ядра можуть бути додані проміжки, що залежать від осі ((0), 0,d_{1},d_{2},..., d_{n})

Якщо на вхід подається тензор розміру ((N), C, $D1$, $D2$,.., D_{n}), вихідний тензор матиме розмір $((N), C, D'_1, D'_2, \ldots, D'_n)$, где D'_iобчислюється за формулою:

$$D'_i = \left\lfloor \frac{D_i + 2 \cdot p_i - d_i \cdot (k_i - 1) - 1}{s_i} + 1 \right\rfloor$$

Параметри:

- kernel_size * ($k_{1},..., k_{n}$) - розмір ядра
- stride * ($s_{1},..., s_{n}$) - крок ядра по кожній осі
- padding * ($p_{1},..., p_{n}$) - Розміри відступів по кожній осі
- dilation * ($d_{1},..., d_{n}$) - Розмір проміжків між елементами ядра
- ceil_mode — якщо *True,* для обчислення розміру вихідного тензора буде використовуватися округлення вгору

* — параметр може приймати на вхід одне чило (a), що інтерпретуватиметься як вектор розміру n : ($a,...,a$)

• MaxPool2d — Двовимірний максимальний пулінг

5. • Dropout [12] — Спеціальний шар, який під час навчання обнулює випадкові елементи вхідного тензора з ймовірністю p.
6. Константи - Шари, що нічого не приймають на вхід, що повертають фіксоване значення.
7. • ToType — Шар, який вводить тензор до заданого типу даних.
8. Арифметичні операції - такі шари отримують на вхід два шари, застосовують до них арифметичну операцію. Підтримують бродкастінг (Посилання на документацію PyTorch) , тобто по можливості наводять тензори до одного розміру, якщо ті відрізняються.

9. Зміна форми тензора • Flatten — Згладжує кілька сусідніх осей тензора на один. Іншими словами, якщо на вхід приходить тензор розміру ($a_{1,}...,a_{n,}b_{1,}...,b_{k,}c_{1,}.., c_{m}$), на виході вийде тензор розміру ($a_{1,}...,a_{n,}b_{1} \times b_{2} \times \cdots \times b_{k,} c_{1,}..$).

Кожному шару відповідає свій клас із модуля torch.nn, назвемо його робочим модулем. Всі необхідні функції та властивості шарів реалізовані в базовому класі Layer, тому при створенні нового шару необхідно лише прописати віджети, що приймають параметри і якщо робочий модуль не реалізований в модулі nn бібліотеки torch, потрібно створити клас, що успадковується від nn.Module, в якому буде реалізований метод forward.

Також під час створення шару можна налаштувати кількість вхідних і вихідних кнопок, передавши список їх імен метод ініціалізації наслідуваного класу (Layer). Під час навчання кожен шар приймає на вхід та повертає словники, ключі яких відповідають іменам кнопок. Оскільки більшість робочих модулів приймають на вхід і повертають по одному тензору, за замовчуванням кожен шар приймає словник виду { ′*in*′: *input* }, передає значення *input* у робочий модуль, отримуючи на виході тензор *output,* після чого повертає словник { ′*out*′: *output* }.

3.6.2. Вкладка «Параметри навчання»

У цій вкладці можна вибрати та налаштувати функцію втрат, а також алгоритм оптимізації.

Основний екран складається з двох рядків, перший відповідає функції втрат, другий алгоритм оптимізації. У перспективі можуть бути додані додаткові рядки, що відповідають, наприклад, планувальнику швидкості навчання (learning rate scheduler).

У кожному рядку є дві кнопки, першою написано назву поточної функції втрат чи оптимізатора. Якщо натиснути на неї, праворуч відкриється меню параметрів. Натискання на другу кнопку з написом *Змінити* відкриє меню вибору функції втрат або оптимізатор відповідно.

Функція втрат - це функція, що приймає на вхід передбачення моделі за вхідними даними разом з цільовими значеннями, що повертає число, або вектор, що характеризує якість передбачень. Додані в даний момент функції втрат працюють за таким принципом:

Вони мають рівно один параметр - reduction, який може приймати значення: *none, mean* або *sum.*

Нехай $x = (x_{1,}..., x_{N})$ - передбачення моделі, $y = (y_{1,}..., y_{N})$, де N - розмір батча. Тоді, якщо значення reduction - *none* :

$$l (x, y) = L = \{ l_{1,}..., l_{N} \}^{T,} \text{ де } \qquad l_{n} = f (x_{n,} y_{n})$$

Тут f – функція втрат для скалярних значень.

Якщо reducion має значення *mean,* тоді

$$l(x, y) = \frac{\sum_{l_i \in L} l_i}{N}$$

А якщо значення дорівнює *sum,* то

$l (x, y) = {}^{X} l_{i}$

$l_{i} \in L$

На даний момент додані такі функції втрат:

- MSELoss - $f (x, y) = (x - y)^{2}$
- BCELoss - $f (x, y) = y \cdot \log x + (1 - y) \cdot \log(1 - x)$.

4. Вкладка «Навчання»

Після того як було ініціалізовано dataloader, вибрано трансформації для передобробки, створено, скомпільовано та протестовано модель, а також обрано функцію втрат та алгоритм оптимізації, можна запустити навчання моделі. Для цього у вкладці «Навчання» необхідно натиснути кнопку *Запуск.* Після цього розпочнеться навчання, відповідно до алгоритму 1.

Внизу вкладки розташований індикатор прогресу, що показує поточну епоху та відсоток оброблених даних на ній (рис. 2). Посередині розташований графік історії значень функції помилки на тренувальній та валідаційній вибірках, а зверху розташоване меню, що дозволяє взаємодіяти з графіком: змінювати масштаб, змінювати кольори ліній та інше.

5. Подальші покращення

Незважаючи на те, що за допомогою Learn2Learn вже можна створювати та налаштовувати нейромережі, програма все ще знаходиться на ранній стадії розробки: безліч функцій, які регулярно використовуються інженерами машинного навчання, поки що не були додані, через що 2L поки що складно використовувати на практиці. Серед функціоналу, який додаватиметься у майбутньому, можна перерахувати:

- Можливість налаштування девайсу, на якому навчається модель, а в перспективі можливість підключення до віддалених серверів та їх обчислювальних потужностей

ISSN 2219-9365

- Додавання можливості створення шарів та функцій втрат за аналогією з поточним способом створення dataloader'a
- Можливість створення шарів і функцій втрат графічним методом - поєднуючи вже існуючі шари в блоки і зберігаючи їх на пристрої
- Додавання бібліотеки вбудованих стандартних датасетів, таких як MNIST, CIFAR, Coco і т. д. і побудованих для них dataloader'ів
- Можливість додавання додаткових метрик на графіках під час навчання • Парсинг існуючих моделей, створених традиційним способом
- Додавання бібліотеки попередніх моделей, таких як ResNet, DenseNet і т.д.
- Поліпшення дизайну

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Таким чином, було створено додаток з графічним інтерфейсом, що дозволяє створювати та налаштовувати довільні моделі глибокого навчання з нуля практично без використання коду. Єдиний елемент, налаштування якого здійснюється традиційним способом, був спроектований з метою демонстрації різних підходів до візуального налаштування та можливості переходу до графічного інтерфейсу без втрати гнучкості інтерфейсу командного рядка.

Розроблена програма дозволяє користувачеві контролювати весь процес навчання: завантаження даних, побудова нейромережі, вибір методів оптимізації та аналіз якості. Система побудована таким чином, що її легко доповнювати та удосконалювати, головним досягненням роботи можна назвати розробку класу базового шару, завдяки якому можна легко додати до програми шар із довільним перетворенням даних.

Незважаючи на те, що можливості графічного інтерфейсу не були реалізовані повною мірою, вже зараз система бере на себе велику частину роботи користувача, за умовчанням відображаючи додаткову інформацію про об'єкти, розташовані на екрані, для отримання традиційним способом якої довелося б провести додаткову роботу.

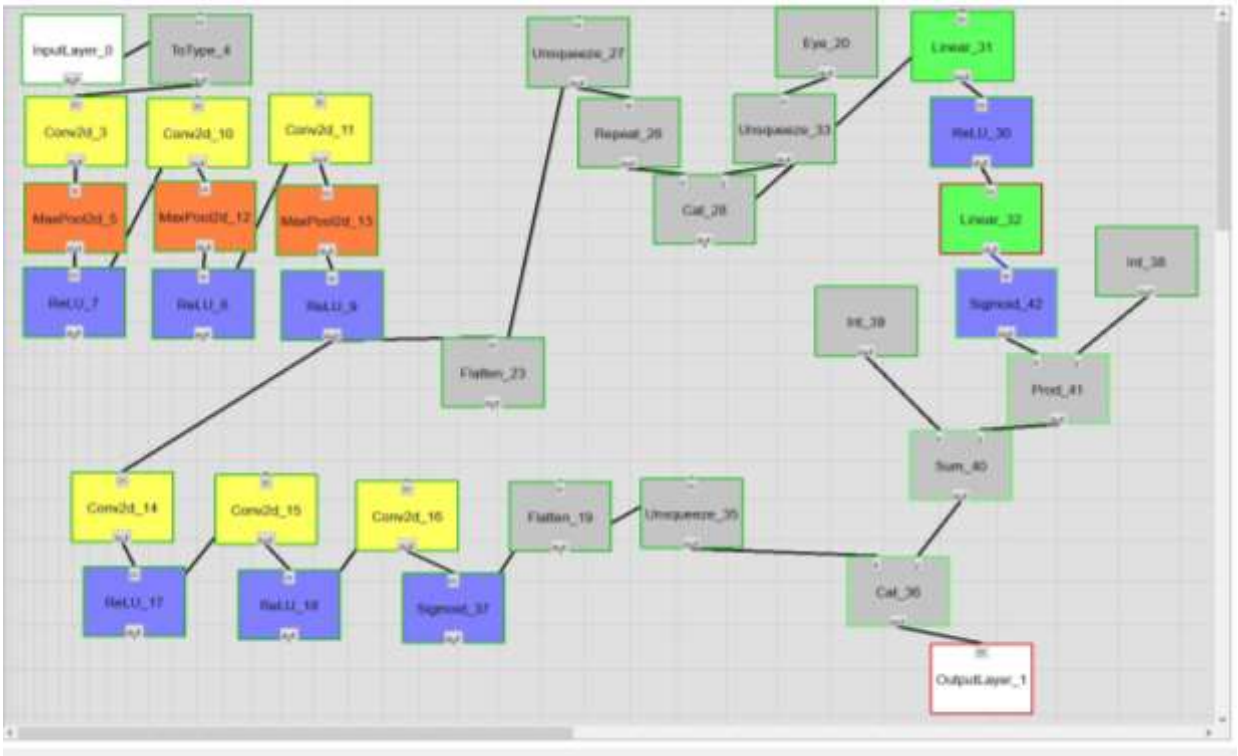

Рис. 1. Конструктор нейромережі

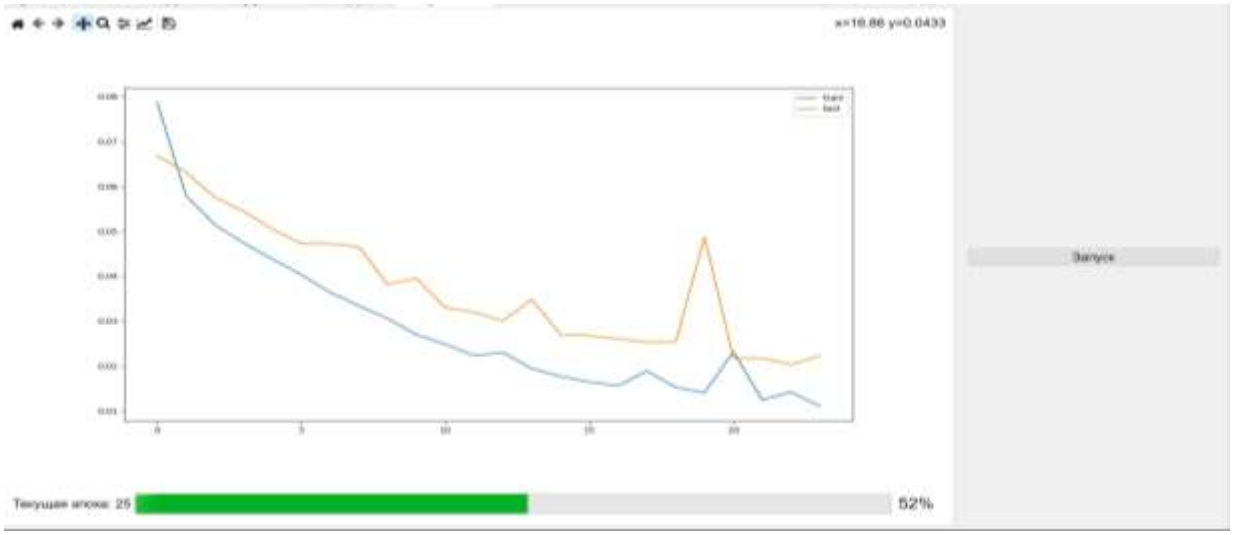

Рис. 2. Прогрес навчання та графік функції помилки

References

1. Huang, YL, та ін. (2022). *ExBrainable : На Open-source GUI для CNN-базованого EEG записування і моделі interpretation.* arXiv preprint arXiv:2201.04065. https://arxiv.org/abs/2201.04065
2. Jin, H., Wagner, MW, Ertl -Wagner, B., & Khalvati, F. (2022). На освітній графічний user interface to construct convolutional neural networks for teaching artificial intelligence in radiology. *Canadian Association of Radiologists Journal.* Advance online publication. https://doi.org/10.1177/08465371221144264
3. Klemm, S., та інші. (2018). *Barista: A graphical tool for designing and training deep neural networks.* arXiv preprint arXiv:1802.04626. https://arxiv.org/abs/1802.04626
4. O'Shea, K., & Nash, R. (2015). *An introduction to convolutional neural networks.* arXiv preprint arXiv:1511.08458. https://arxiv.org/abs/1511.08458
5. Baldi, P., & Sadowski, PJ (2013). Understanding dropout. In *Advances в Neural Information Processing Systems* (Vol. 26). https://papers.nips.cc/paper_files/paper/2013/hash/14acdf64020072a74d1cf17d7e6f5576-Abstract.html
6. Хінтон, Г.Е., Срівастава, Н., Крижевський, А., Сутскевер, І., & Салахутдінов, RR (2012). *Improving neural networks by preventing co-adaptation of feature detectors.* arXiv preprint arXiv:1207.0580. https://arxiv.org/abs/1207.0580
7. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. У *процедурах IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770–778). https://doi.org/10.1109/CVPR.2016.90
8. Babenko, V., Pronin, S., Aleksejeva, L., Vītola, Z., Sokolova, L., Dyuzhev, V., & Dyakova, N. (2024). Tools for Consumer Preference Analysis Based in Machine Learning. Journal of Information Technology Management, 16(4), 1-17. doi: 10.22059/jitm.2024.99048
9. Michael M. Maynard. 2003. Univac I. Encyclopedia of Computer Science. John Wiley and Sons Ltd., GBR, 1813-1814.
10. Pursky, O., Babenko, V., Danylchuk, H., Dubovyk, T., Buchatska, I., Dyvak, V. (2024). Recommender System of Site Information Content for Optimal Display in Search Engines. In: Joby, P.P., Alencar, M.S., Falkowski-Gilski, P. (eds) IoT Based Control Networks and Intelligent Systems. ICICNIS 2023. Lecture Notes in Networks and Systems, vol 789. Springer, Singapore. https://doi.org/10.1007/978-981-99-6586-1_10
11. Pilavakis, AJ (1989). The Bourne Shell. In: UNIX Workshop. Macmillan Computer Science Series. Palgrave, London. https://doi.org/10.1007/978-1-349-19900-6_3
12. Ramey C. Bash, Bourne-Again Shell //Proceedings of The Romanian Open Systems Conference & Exhibition (ROSE 1994), The Romanian UNIX User's Group (GURU). 1994. – С. 3-5.
13. Babenko, V., Mukashev, T., Liubokhynets, L., Iurchenko, M., Yefanov V., Lantrat, O., Kovtun Ye. (2025). Managing Transport Systems with Artificial Intelligence. Journal of Information Technology Management, 17(2), pp 13-27. DOI: 10.22059/jitm.2025.101574
14. Mahdi, Q. A., Shyshatskyi, A., Babenko, V., Bieliakov, R., Odarushchenko, E., Protas, N., Stasiuk, T., Rukavyshnikov, Y., Miziak, I., Lantrat, O. (2024). Development of a solution search method using artificial intelligence. Eastern-European Journal of Enterprise Technologies, 2 (4 (128)), 38–47. https://doi.org/10.15587/1729-4061.2024.300261
15. Mambetov, S., Ilhe, I., Babenko, V., Kulambayev, B., Fridman, O., Joldasbayev, S., Doroshenko, H., Gurko, O., Begimbayeva, Y., & Neronov, S. (2024). Detection and classification of threats and vulnerabilities on hacker forums based on machine learning. Eastern-European Journal of Enterprise Technologies, 3(9 (129), 16–27. https://doi.org/10.15587/1729-4061.2024.306522
16. Iurchenko, M., Šaltytė-Vaisiauskė, L., & Babenko, V. (2025). Quantifying asset price volatility with fractional Brownian motion. Technology Audit and Production Reserves, 3(4(83), 34–41. https://doi.org/10.15587/2706-5448.2025.329243
17. Klemm S. та ін. Barista-a graphical tool for designing and training deep neural networks // arXiv preprint arXiv:1802.04626. - 2018.