Міністерство освіти і науки України
ДВНЗ «Прикарпатський національний університет імені Василя Стефаника»
Кафедра комп'ютерної інженерії та електроніки
(повна назва кафедри)

Славський Михайло Васильович

УДК 004:681.5

Спеціальність 123 «Комп'ютерна інженерія та електроніка»
(шифр та назва спеціальності)

Кваліфікаційна робота
на здобуття освітньо-кваліфікаційного рівня магістр
(бакалавр, спеціаліст, магістр)

Маршрутизація літального апарату з використанням Gool Map і штучної нейромережі

Aircraft routing using Google Maps and an artificial neural network

Науковий керівник:
кандидат технічних наук,
доцент кафедри комп'ютерної
інженерії та електроніки
Голота В.І.
Рецензент:

Івано-Франківськ
2024

Форм.	Зона	Поз.	*Позначення*	*Найменування*	К-ть	Прим.
			123. УДК 004:681.5	Пояснювальна записка	1	
			123. УДК 004:681.5	Схема структури коду	1	

					123. УДК 004:681.5
Змн.	*Арк.*	*№ докум.*	*Підп.*	*Дата*	
Разробив		Славський М.В.			Специфікація
Перевірив		Голота В.І.			
Н. Конт.					

Літ.	*Арк.*	*Аркушів*
	2	

Затвердив					

АНОТАЦІЯ

Моделі продуктивності, які використовуються в процесі розробки літальних апаратів, залежать від припущень і наближень, пов'язаних з інженерними рівняннями, які використовуються для їх створення. Розробка та реалізація цих дуже складних інженерних моделей зазвичай пов'язані з більш тривалим процесом розробки. У цьому дослідженні пропонується недетермінований підхід, коли методи машинного навчання з використанням штучних нейронних мереж використовуються для прогнозування конкретних параметрів літака з використанням доступних даних. Цей підхід дає результати, які не залежать від рівнянь, що використовуються в звичайних методах моделювання характеристик літальних апаратів, і покладаються на стохастичні дані та їх розподіл для вилучення корисних моделей. Щоб перевірити життєздатність підходу, виконується тематичне дослідження, яке порівнює звичайну модель продуктивності, що описує дистанцію крену при зльоті, зі значеннями, згенерованими нейронною мережею з використанням легкодоступних даних польоту. Нейронна мережа отримує як вхідні дані та навчається з використанням параметрів характеристик літака, включаючи атмосферні умови (температура повітря, тиск повітря, щільність повітря), робочі характеристики (конфігурація закрилків, налаштування тяги, MTOW тощо) та умови злітно-посадкової смуги (мокра, суха). , кут нахилу тощо).

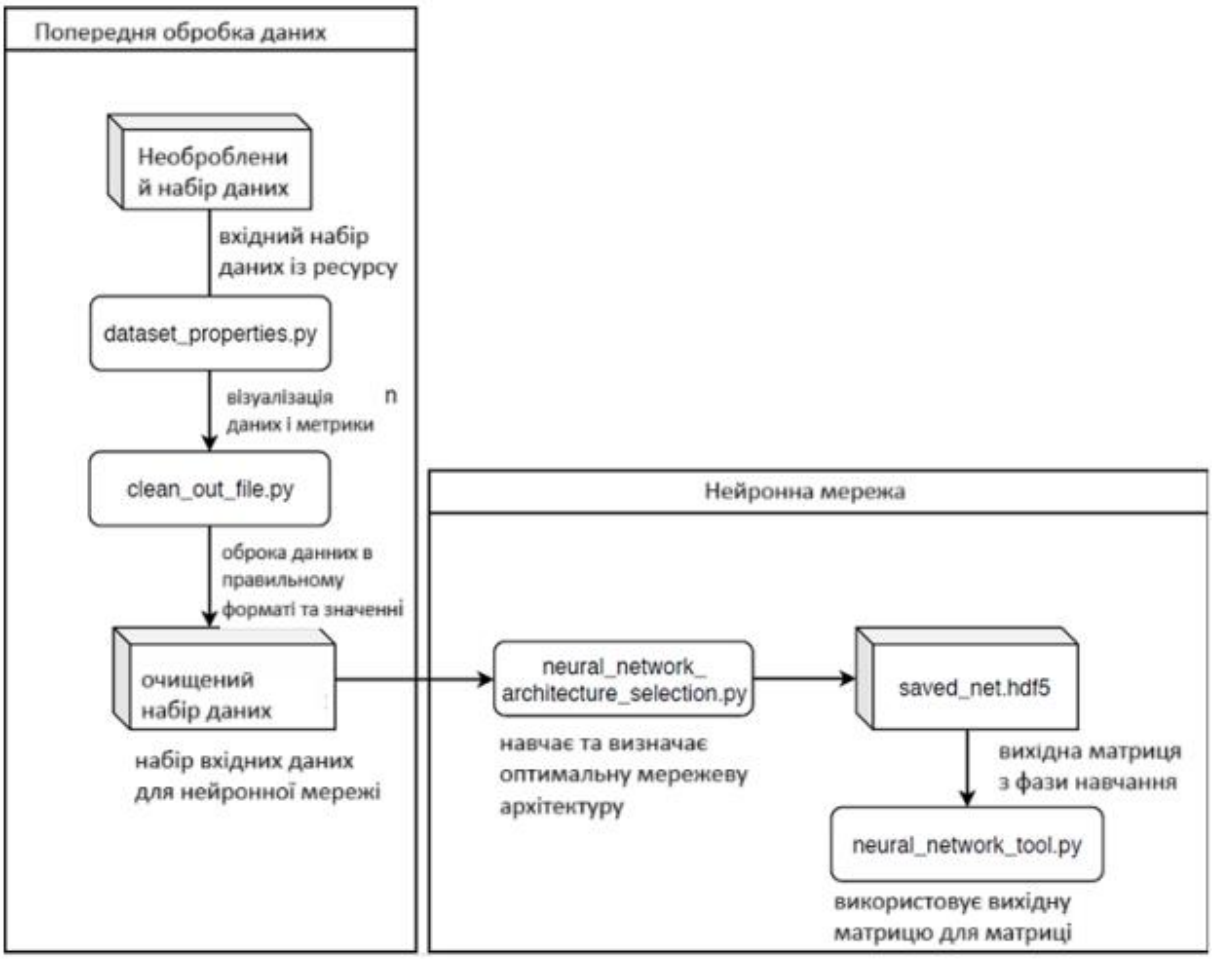
Дана робота складається зі вступу, 4 розділів, висновка, 30 джерел використаної літератури, додатка, 29 рисунків, 12 таблиць.

Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Славський М.В.			Анотація	*Літ.*	*Арк.* 3 / *Аркушів* 1
Перевірив		Голота В.І.					
Н. Контр.		.					
Затвердив							

ABSTRACT

Performance models used in the aircraft development process are dependent on the assumptions and approximations associated with the engineering equations used to produce them. The design and implementation of these highly complex engineering models are typically associated with a longer development process. This study proposes a non-deterministic approach where machine learning techniques using Artificial Neural Networks are used to predict specific aircraft parameters using available data. The approach yields results that are independent of the equations used in conventional aircraft performance modeling methods and rely on stochastic data and its distribution to extract useful patterns. To test the viability of the approach, a case study is performed comparing a conventional performance model describing the takeoff ground roll distance with the values generated from a neural network using readily-available flight data. The neural network receives as input, and is trained using, aircraft performance parameters including atmospheric conditions (air temperature, air pressure, air density), performance characteristics (flap configuration, thrust setting, MTOW, etc.) and runway conditions (wet, dry, slope angle, etc.).

This work consists of an introduction, 4 chapters, conclusion, 30 sources of used literature, an appendix, 29 figures, 12 tables.



Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Славський М.В.			Схема структури коду	*Літ.*	*Арк.* 5 / *Аркушів* 1
Перевірив		Голота В.І.					
Н. Контр.		.					
Затвердив							

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Фізико-технічний факультет
Кафедра комп’ютерної інженерії та електроніки

Пояснювальна записка
до кваліфікаційної роботи на тему
Маршрутизація літального апарату з використанням Gool Map і штучної нейромережі

					123.Ел(м)-21.03		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розробив		Славський М.В.			. Пояснювальна записка	Літ.	Арк. 6 / Аркушів 93
Перевірив		Голота В.І.					
Н. Контр.		.					
Затвердив							

ПЕРЕЛІК СКОРОЧЕНЬ

IEEE - Institute of Electrical and Electronics Engineers

ШНМ - штучні нейронні мережі

ШІ – штучний інтелект

MAE - mean absolute error

RNN - recurrent neural network

БПЛА - Безпілотний літальний апарат

AEO - All Engines Operating

AFM - Aircraft Flight Manual

AI - Artificial Intelligence

ANN - Artificial Neural Network

ASD - Accelerated Stop Distance

CNN - Convolutional Neural Network

LTSM - Long Term Short Term Memory

LT - Logic Theorist

ML - Machine Learning

MAPE - Mean Average Percentage Error

MSE - Mean Squared Error

NN - Neural Network

NASA - National Aeronautics and Space Administration

OEI - One Engine Inoperative

OEM - Original Equipment Manufacturer

TOD - Takeoff Distance

TOD_N or TOD_{AEO} - TOD with all engines operating

TOD_{N-1} or TOD_{OEI} - TOD with one engine inoperative

TOFL - Takeoff Field Length

TOR - Takeoff run

ЗМІСТ

ВСТУП

Сучасний процес проектування літаків розвинувся, щоб включити системи дедалі більшої складності. Щоб краще зрозуміти ці системи, розробляються інженерні моделі, де більшість із цих моделей є детермінованими за своєю природою. Детермінована модель — це система, у якій усі вихідні параметри можна обчислити на основі їх співвідношення з іншими значеннями параметрів, що впливають на систему, та їхні початкові умови. Іншими словами, це система, для якої всі можливі стани зрозумілі і, певною мірою, передбачувані. Багато інженерних проблем можна вирішити за допомогою детермінованих моделей. Детерміновані методи виявилися успішними в програмах, де механізми, що використовуються для опису повної поведінки системи, можуть бути повністю зрозумілі або зрозумілі достатньо, щоб мати можливість описати явище з мінімальною кількістю прийнятної помилки. Закони руху Ньютона є прикладами важливих детермінованих моделей. Коли вони наносяться на тіло, кажуть, що майбутні результати тіла можна передбачити або визначити його поточною ситуацією; і якщо їх застосувати до двох ідентичних тіл за однакових умов, результат для обох тіл буде однаковим.

У сфері проектування літаків багато детермінованих систем відіграють фундаментальну роль у процесі проектування, наприклад, моделі, що описують рух літака в часі та просторі; моделі відстеження та прогнозування стану бортових систем літака (тобто паливних систем, електричних систем, систем контролю середовища); моделі контролю та впливу на стан систем керування польотом; або моделі, що описують аеродинамічні або структурні навантаження на компоненти літака. Хоча ці моделі та їхнє успішне використання заслуговують на увагу, слід розуміти, що всі детерміновані моделі мають властиві обмеження, деякі з яких перелічені нижче[1]:

- Можливо, що емпіричних даних для розробки детермінованої моделі буде недостатньо або взагалі не буде.
- Прогнозовані результати можуть виходити за межі допустимих похибок у порівнянні з емпіричними даними.

- Дуже складні детерміновані моделі можуть бути дорогими з точки зору обчислень і вимагати більшого часу виконання.
- Експертні знання системи, що розробляється, завжди потрібні для створення нової детермінованої моделі.
- Якщо система надто складна, щоб мати можливість розробити детерміновані рівняння, що визначають її поведінку, розробити таку детерміновану модель може бути неможливо або надзвичайно важко.

Ці обмеження сприяли тому, що недетерміновані моделі почали досліджувати як засіб вирішення деяких обмежень детермінованих моделей, які зустрічаються в аерокосмічній промисловості, де типово мати систему високої складності з дуже великі обсяги даних, для яких майже завжди потрібні експертні знання. Недетермінована модель описує систему, для якої поведінка параметрів системи вважається стохастичною і для якої неможливі детерміновані зв'язки. Цей тип підходу до моделювання базується на ймовірності та статистиці, які вводять випадковість у моделі таким чином, що результати моделі можна розглядати як розподіли ймовірностей, а не як унікальні значення. Таким чином, недетерміновані методи можуть давати різні результати після кількох прогонів для того самого набору задач і, отже, зазвичай пов'язані з інтервалами невизначеності параметрів для точкових оцінок і прогнозів. Залежно від наявної проблеми та інструментів, доступних для її вирішення, може бути практичним перетворити суто детерміновані проблеми в недетерміновані проблеми шляхом введення стохастичних змінних. Були проведені дослідження щодо того, як прогнозні результати відрізняються залежно від того, який підхід використовується між детермінованим і недетермінованим методами для того самого набору проблем.

Предметом даної роботи є моделі нейронних мереж та їх застосування у сфері авіації, зокрема для оптимізації маршрутизації малих літальних апаратів за допомогою Google Maps.

Мета даної роботи полягає в розробці маршрутизації літального апарату з використанням Google Map і штучної нейронної мережі.

Завдання:

1. Розглянути основні відомості про моделі нейромереж;
2. Дослідити вибрані характеристики літального апарату;
3. Розглянути процес розробки нейронної мережі;
4. Розробити архітектуру нейромережі;
5. Реалізувати програмну частину нейромережі для літального апарату.

РОЗДІЛ 1. ОСНОВНІ ВІДОМОСТІ ПРО МОДЕЛІ НЕЙРОМЕРЕЖ

1.1. Історія розвитку нейромереж

Ідея нейронних мереж виникла, як не дивно, як модель функціонування нейронів у мозку, названа «коннекціонізмом» і використовувала зв'язані схеми для імітації розумної поведінки. У 1943 році нейрофізіолог Уоррен МакКаллох і математик Уолтер Піттс представили просту електричну схему. Дональд Хебб розвинув цю ідею у своїй книзі «Організація поведінки» (1949), припустивши, що нервові шляхи зміцнюються з кожним наступним використанням, особливо між нейронами, які мають тенденцію активуватися одночасно, таким чином починаючи довгий шлях до кількісного визначення складних процесів мозок[2].

Дві основні концепції, які є попередниками нейронних мереж:

1. «Порогова логіка» — перетворення безперервного входу на дискретний вихід

2. «Геббіанське навчання» — модель навчання, заснована на пластичності нейронів, запропонована Дональдом Геббом у його книзі «Організація поведінки», яку часто підсумовують фразою: «Клітини, які працюють разом, з'єднуються разом».

Обидва запропоновані в 1940-х роках. У 1950-х роках, коли дослідники почали намагатися перевести ці мережі на обчислювальні системи, перша мережа Хебба була успішно реалізована в MIT у 1954 році.

Приблизно в цей час Френк Розенблатт, психолог Корнелла, працював над розумінням відносно простіших систем прийняття рішень, які лежать в основі та визначають її реакцію. Намагаючись зрозуміти та кількісно оцінити цей процес, він запропонував ідею персептрона в 1958 році, назвавши його перцептроном Mark I. Це була система з простим співвідношенням вхідних даних і вихідних даних, змодельована на основі нейрона Мак-Каллоха-Піттса, запропонованого в 1943 році Ворреном С. Мак-Каллохом, нейробіологом, і Уолтером Піттсом, логіком, щоб пояснити складні процеси прийняття рішень у мозку за допомогою лінійного порогу. Нейрон Мак-Каллоха-Піттса приймає вхідні дані, отримує

зважену суму та повертає «0», якщо результат нижче порогового значення, і «1» в іншому випадку (рис.1.1.).

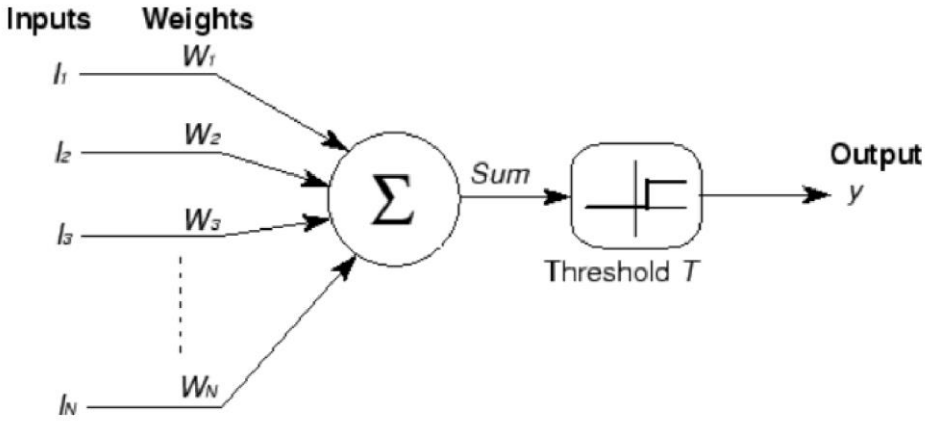


Рисунок 1.1. Нейрон Мак-Калоха-Піттса

Краса перцептрона Марк-1 полягала в тому, що його ваги «вивчалися» через послідовно передані вхідні дані, мінімізуючи при цьому різницю між бажаним і фактичним результатом(рис.1.2.).[3]

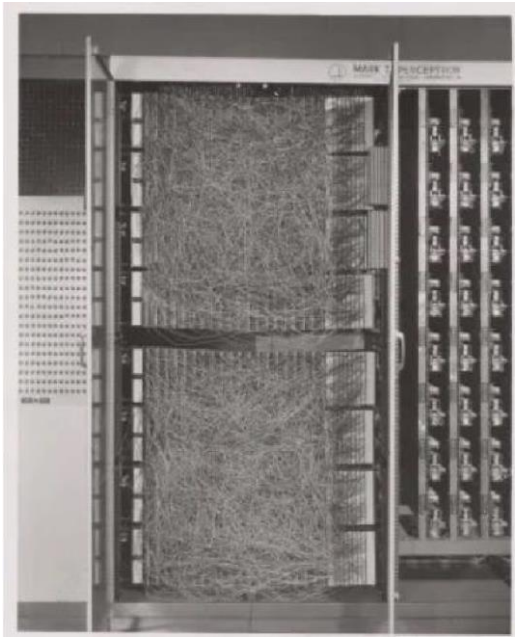


Рисунок 1.2. Перша відома реалізація персептрона Mark I. Машина була підключена до камери, яка використовувала фотоелементи з сульфіду кадмію 20×20 для створення 400-піксельного зображення. Основною видимою особливістю є патчборд, який дозволяє експериментувати з різними комбінаціями вхідних функцій. Праворуч від нього знаходяться масиви потенціометрів, які реалізували адаптивні ваги

Цей перцептрон міг лише навчитися розділяти лінійно роздільні класи, роблячи просту, але нелінійну схему або нездоланним бар'єром(рис.1.3.).

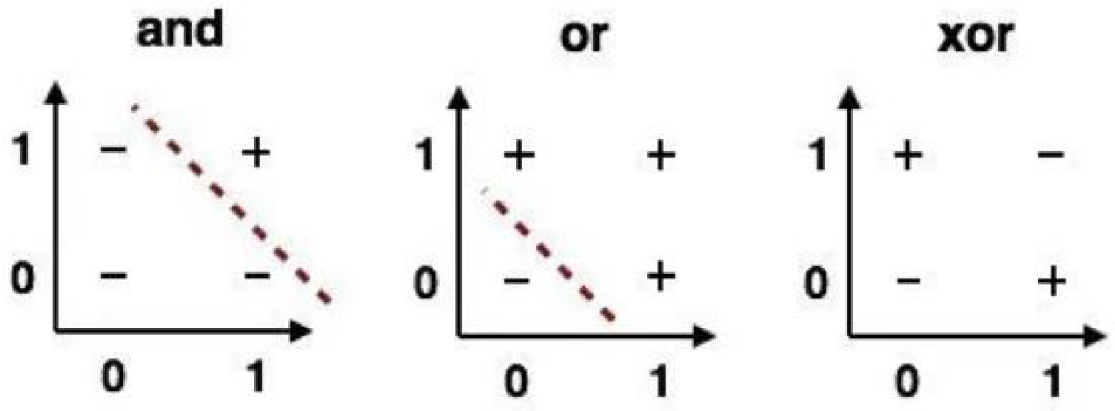


Рисунок 1.3. Перцептрон

Незважаючи на заплутаність і дещо незадовільну появу використання машинного навчання для кількісної оцінки систем прийняття рішень, окрім мозку, сьогоднішні штучні нейронні мережі — це не що інше, як кілька рівнів цих перцептронів.

Приблизно в цей час все почало розвиватися швидко для нейронних мереж, і в 1959 році в Стенфорді Бернард Уідроу та Марчіан Гофф розробили першу нейронну мережу, успішно застосовану до проблеми реального світу. Ці системи отримали назви ADALINE і MADALINE після використання в них кількох ADAptive LINEar Elements, останній з яких був спеціально розроблений для усунення шуму в телефонних лініях і все ще використовується сьогодні. Однак ці штучні нейрони відрізнялися від персептронів тим, що вони повертали як вихідні дані, що в даному випадку було зваженим входом.

Як і у випадку з кожним незначним удосконаленням технології штучного інтелекту в історії, ці ранні успіхи викликали зростаючий ажіотаж щодо можливостей і потенціалу нейронних мереж, у той час як дослідники стикалися з одним блокпостом за іншим. На піку ажіотажу навколо цих «мислячих машин» NYtimes опублікував цю статтю про потенціал нейронних мереж.

Все це закінчилося в 1969 році з публікацією книги «Перцептрони» Марвіна Мінскі, засновника MIT AI Lab, і Сеймура Пеперта, директора

лабораторії. У книзі переконливо доводилося, що єдиний підхід Розенблата до нейронних мереж не може бути ефективно перетворений на багаторівневі нейронні мережі. Щоб оцінити правильні відносні значення ваг нейронів, розподілених між шарами на основі кінцевого результату, знадобиться кілька, якщо не нескінченна кількість ітерацій, і обчислення займе дуже багато часу.

Мінскі у своєму тексті виклав ці та інші проблеми з нейронними мережами та фактично привів ширшу наукову спільноту, а головне фінансові установи, до висновку, що подальші дослідження в цьому напрямку нікуди не приведуть. Вплив цього тексту був потужним і призупинив фінансування до такої міри, що протягом наступних 10–12 років ніхто з найбільших дослідницьких установ того часу, а отже й менших, не брався за будь-який проект, який мав би основу приречені нейронні мережі. Почалася епоха, яку зараз називають «зимою ШІ».

Відлига цієї десятирічної зими почалася в 1982 році в Національній академії наук, коли Джон Хопфілд представив свою статтю про те, що стало відомо як мережа Хопфілда, а того ж року на американо-японській конференції з кооперативних і конкурентних нейронних мереж Японія оголосила, що має намір розпочати роботу над нейронними мережами п’ятого покоління. Це призвело до того, що фінансування знову почне надходити зі скарбниці нації, яка боїться залишитися позаду. Невдовзі Американський інститут фізики в 1985 році організував щорічну зустріч «Нейронні мережі в обчислювальній техніці», за якою в 1987 році відбулася перша Міжнародна конференція з нейронних мереж Інституту інженерів з електротехніки та електроніки (IEEE)[4].

Однак це було велике повторне відкриття концепції, яка вже існувала з 60-х років, що допомогло нейронним мережам вибратися з передчасної могили. Зворотне розповсюдження, метод, розроблений дослідниками з 60-х років безперервно розвивався протягом зими штучного інтелекту, був інтуїтивним методом, який приписував зменшення значущості кожній події, коли хтось повертався далі в ланцюжку подій. Першою людиною, яка побачила їхній потенціал для нейронних мереж і розв’язала питання про те, як це можна було б застосувати для MLP, був Пол Вербос, який частково надихнувся їх

застосуванням до людського розуму та роботою Фрейда щодо зворотного потоку присвоєння кредитів, написав докторську дисертацію, в якій викладає їх важливість. Однак ця робота не була помічена ніким із спільноти, доки Паркер не опублікував звіт про свою роботу в M.I.T. у 1985 році. Лише після повторного відкриття Румельхартом, Хінтоном і Вільямсом і перевидання в чіткій і детальній структурі ця техніка захопила спільноту штурмом. Ті ж автори також звернулися до конкретних недоліків, викладених Мінським у його публікації 1969 року.

Зворотне розповсюдження разом із градієнтним спуском формує основу та електростанцію нейронних мереж. У той час як Gradient Descent постійно оновлює та переміщує ваги та зміщення до мінімуму функції вартості, зворотне поширення оцінює градієнт вартості відносно вагові коефіцієнти та зміщення, величина та напрямок яких використовуються градієнтним спуском для оцінки розміру та напрямку поправок до параметрів вагових коефіцієнтів та зміщення (рис.1.4).

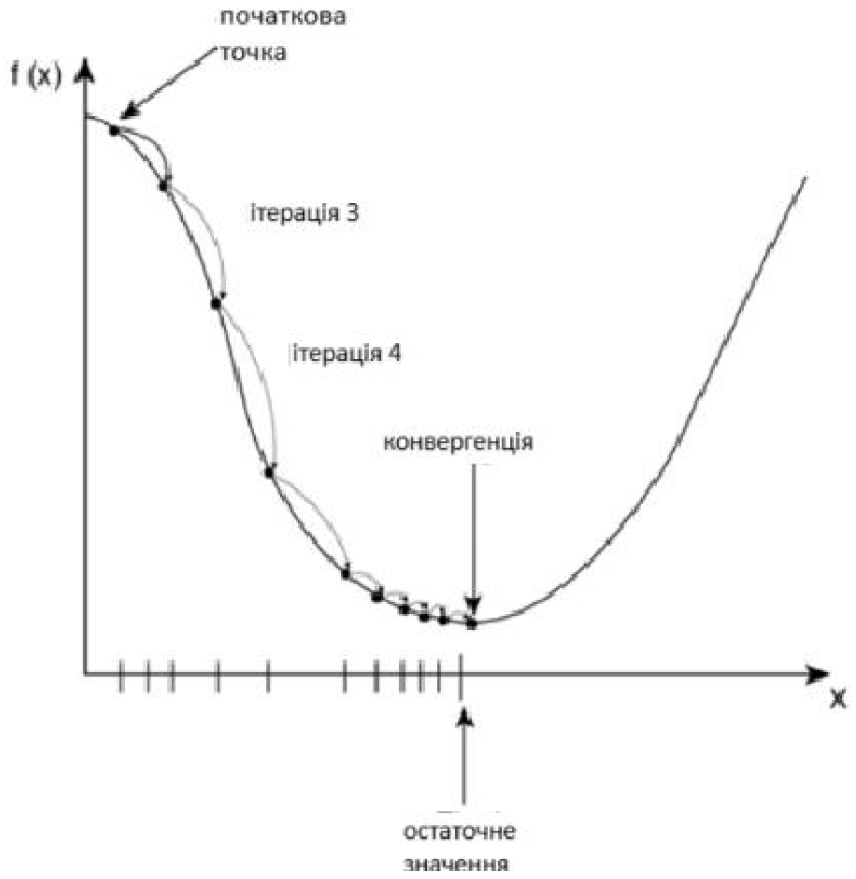


Рисунок 1.4. Простий візуальний опис руху до мінімумів 2d-функції. Розмір кроку стрибка визначається значенням градієнта в кожній точці.

Таким чином, до 1990-х років нейронні мережі точно повернулися, цього разу справді захоплюючи уяву світу й нарешті виправдавши, якщо не перевершивши, його очікування.

1.2. Сфера застосування нейромереж в технічних системах

Штучний інтелект приніс революційні зміни у широку сферу IT. Багато разів побачите рекламу або пропозиції щодо реклами речей, якими користуєтеся, і дивуватиметесь, як це можливо. Знову ж таки, можна задатися питанням, як Siri та Google Assistants слухають і виконують підказки відповідно до вказівок. Усі ці сценарії є результатом штучного інтелекту, де штучні нейронні мережі (ШНМ) формують елемент, який спрямовує машини щодо того, як реагувати на підказки.

Так само, як функціонує людський мозок, функціонують нейронні мережі. Таким чином, нейронна мережа є способом, за допомогою якого ШІ спонукає комп'ютер обробляти інформацію та діяти як людський мозок.

Це просто процес машинного навчання, відомий як глибоке навчання, і він залежить від нейронів або вузлів, які багатошарові, як людський мозок. Завдяки нейронним мережам комп'ютери можуть адаптуватися до системи, безперервно навчатися та постійно вдосконалюватись для кращого досвіду. Отже, це означає, що штучні нейронні мережі намагаються якомога більше розпізнавати обличчя, підсумовувати документи та вирішувати складні проблеми точніше.

Нейронні мережі відіграють важливу роль у регулюванні та контролі роботи деяких секторів, включаючи автомобільну, охорону здоров'я та фінанси, серед багатьох інших. Щоб краще зрозуміти нейронні мережі, нижче наведено деякі програми, які слід знати:

1. Соціальні медіа

Соціальні мережі, будучи широкими, тепер покращили загальний досвід користувачів завдяки нейронним мережам. Штучні нейронні мережі мають можливість вивчати й аналізувати різних користувачів соціальних мереж за допомогою даних, якими вони щодня діляться. Крім того, інтелектуальний аналіз даних з різних платформ соціальних медіа досягається за допомогою багаторівневого персептрона ШНМ[5].

Більше того, MLP прогнозує тенденції соціальних медіа за допомогою різноманітних методів навчання, таких як MSE-середня квадратична помилка, MAE (середня абсолютна помилка), щоб аналізувати улюблені сторінки користувачів під час використання соціальних мереж. Зважаючи на це, очевидно, що штучна нейронна мережа є найкращим підходом для аналізу даних користувача.

2. Розпізнавання обличчя

Система розпізнавання облич зараз на замовлення сучасних технологій. Система запрограмована на розпізнавання та зіставлення облич людей із відповідними цифровими зображеннями. Ця технологія зазвичай використовується в офісах для виконання певних записів, коли система запрограмована на перевірку людського обличчя та зіставлення його з переліченими ідентифікаторами, доступними в базі даних.

CNN-згорткові нейронні мережі відіграють ключову роль в обробці зображення та розпізнаванні облич. За допомогою нейронних мереж можна обробляти та завантажувати велику кількість зображень у базу даних для подальшої обробки, а потім для навчання.

CNN має рівні вибірки, які відіграють ключову роль у правильному оцінюванні, коли моделі використовуються для отримання точних результатів розпізнавання.

3. Аерокосмічна

Аерокосмічна інженерія - це загальний термін, що охоплює широку область літаків і космічних апаратів. Серед важливих областей, де нейтральні мережі застосовуються в аерокосмічній техніці, можна відзначити моделювання ключових динамічних симуляцій, захист систем управління літаками, високоефективне автопілотування та діагностику несправностей.

TNN-нейронна мережа із затримкою часу застосовується при моделюванні систем з нелінійною динамікою, забезпечуючи сильнішу динаміку нейронним мережам і розпізнаючи незалежні ознаки позиціонування. Це означає, що

алгоритм, побудований на нейронних мережах із затримкою, запрограмований на розпізнавання та перевірку шаблонів.

У космічному кораблі найважливішим аспектом є безпека пасажирів. Тому алгоритм такої системи запрограмований на підвищення точної системи автопілота.

4. Прогнозування фондового ринку

Інвестиції – це широка тема, де ризики пов’язані з вищим відсотком. Як наслідок, передбачити, що зміниться, а що залишиться постійним, може бути дещо важко. Однак рівень непередбачуваності був високим до винаходу нейронних мереж. З нейтральними мережами все змінилося на краще.

Через нейтральні мережі багатошаровий персептрон (система, яка забезпечує зворотний зв’язок зі штучним інтелектом) може передбачити успішний фондовий ринок у реальному часі. MLP складається з різних рівнів вузлів, кожен з яких підключений до успішного вузла. У результаті стає легше аналізувати минулі результати на фондовому ринку, робити річні прибутки та обчислювати коефіцієнти неприбутковості.

5. Охорона здоров'я

Охорона здоров’я – ще один сектор, де застосовуються нейтральні мережі. Через нейтральні мережі бачимо прояв старої приказки: здоров’я – це багатство. Зараз люди використовують переваги передових технологій для покращення сфери охорони здоров’я.

Серед основних секторів охорони здоров’я, де нейтральна мережа відіграє важливу роль, – ультразвук, комп’ютерна томографія та рентгенівське випромінювання.

Через CNN система може обробляти зображення, а потім дані, отримані з тестів і доступні через моделі нейронної мережі. Це залежить від наданої послуги.

RNN-рекурентна нейронна мережа відіграє ключову роль у розпізнаванні та перевірці голосів із систем.[6]

Сьогодні медичний персонал може легко використовувати функції розпізнавання голосу, щоб отримати інформацію про пацієнта завдяки нейронним мережам.

Наразі експерти працюють над дослідженням GNN-генеративних нейтральних мереж, призначених для виявлення та класифікації різних наркотиків відповідно.

6. Захист

В ідеалі кожна країна повинна мати міцну команду для захисту. У результаті міжнародний домен може отримати доступ до даних кожної країни під час військових операцій через нейтральні мережі. Більше того, розвинені країни використовують нейтральні мережі для зміцнення своєї системи захисту. Наприклад, Японія, Великобританія та США належать до технологічно розвинутих країн, які використовують нейронні мережі для формування своєї оборонної системи.

Серед сфер, де нейтральні мережі відіграють ключову роль у оборонному секторі, можна відзначити місцезнаходження об’єктів, аналіз збройних нападів і логістику. Крім того, нейтральні мережі використовуються для управління автоматизованими безпілотниками, морськими та повітряними патрулями для створення стратегії ідеальної системи захисту.

За допомогою CNN легко виявити існування незаконних маршрутів (підводних мін) між країнами.

7. Прогнозування погоди

До появи штучного інтелекту прогнози погоди були лише приблизними і не зовсім точними. Мета прогнозу погоди – передбачити майбутні погодні умови та вжити запобіжних заходів, якщо це необхідно. Інформація про погоду має вирішальне значення, оскільки вона може допомогти передбачити можливе лихо в найближчому майбутньому, таким чином допомагаючи людям, які оточують певне середовище, вжити необхідних заходів безпеки.

Через MLP, RNN і CNN тепер можна прогнозувати точні погодні умови заздалегідь. Більше того, багатошарові моделі ANN мають можливість

передбачати погоду на 15 днів наперед, що покращує загальний досвід. Поєднання різноманітних нейтральних мереж може покращити прогнозування температур.

8. Перевірка підпису та аналіз почерку

Перевірка підписів має вирішальне значення для визначення життєздатності даних особи. Такі установи, як банки, використовують цю технологію для перевірки особи людей.

Підробка підпису досить поширена серед багатьох фінансових установ, тому потрібна складна програма для підпису для перевірки деталей.

Через ANN тепер можна відрізнити справжні підписи від підроблених. Найкраща частина полягає в тому, що ANN можна використовувати для офлайн- та онлайн-додатків.

Зараз багато компаній вирішують використовувати ШІ та машинне навчання у своїй діяльності. У результаті очікується, що майбутнє нейронних мереж буде розвиватися в усьому світі. Тепер користувачі по всьому світу зможуть користуватися налаштованими нейтральними мережевими програмами для кращого досвіду роботи в різних сферах. Найкращі приклади:

1. Мобільні та веб-програми пропонують персоналізовані параметри на основі історії та активності в соціальних мережах.

2. Зростання гіперінтелектуальних інструментів VA, таких як Siri та Google Assistant, що полегшує роботу.

1.3. Огляд нейронних мереж та їх програмні і апаратні реалізації

Штучні нейронні мережі можна вважати однією з популярних предметних галузей інформатики. Причиною цього є їхня здатність виконувати критично важливі завдання, пов’язані зі штучним інтелектом, такі як класифікація та розпізнавання зображень, виявлення шахрайства з кредитними картками, розпізнавання захворювань тощо.

Простіше кажучи, нейронна мережа — це набір алгоритмів, призначених для розпізнавання закономірностей або зв’язків у заданому наборі даних. Ці глибокі нейронні мережі в основному є обчислювальними системами, розробленими для

імітації того, як людський мозок аналізує та обробляє інформацію. Щоб повністю зрозуміти, як працюють нейронні мережі, важливо зрозуміти структуру та функціонування цих систем.

Наприклад, поширеним прикладом нейронної мережі є розпізнавання зображень, коли мережа вчиться ідентифікувати об’єкти в зображеннях на основі шаблонів, які вона виявляє.

З іншого боку, нейронна мережа складається з нейронів, з’єднаних між собою як мережа, і ці нейрони є математичними функціями або моделями, які виконують обчислення, необхідні для класифікації відповідно до заданого набору правил.

Розуміння того, як працює нейронна мережа, має важливе значення для розуміння фундаментальних концепцій машинного навчання та програм штучного інтелекту[7].

Перш ніж перейти до вивчення того, як саме працює нейронна мережа, потрібно знати, з чого складається нейронна мережа. Звичайна нейронна мережа складається з кількох шарів, які називаються вхідним шаром, вихідним шаром і прихованими шарами. У кожному шарі кожен вузол (нейрон) з’єднаний з усіма вузлами (нейронами) наступного шару за допомогою параметрів, які називаються «вагами».

Нейронні мережі складаються з вузлів, які називаються перцептронами, які виконують необхідні обчислення та виявляють особливості нейронних мереж. Ці перцептрони намагаються зменшити кінцеву похибку вартості шляхом коригування вагових параметрів. Крім того, перцептрон можна розглядати як нейронну мережу з одним шаром.

З іншого боку, багатошарові перцептрони називають глибокими нейронними мережами. Перцептрони активуються, коли є задовільний вхід.

Тепер давайте перейдемо до обговорення точних етапів роботи нейронної мережі.

1. Спочатку набір даних повинен бути поданий на вхідний рівень, який потім буде перенесено на прихований рівень.

2. Зв’язки, які існують між двома шарами, випадковим чином призначають ваги вхідним даних.

3. До кожного входу додається зсув. Зсув – це константа, яка використовується в моделі, щоб найкраще відповідати заданим даним.

4. Зважена сума всіх вхідних даних буде надіслана до функції, яка використовується для визначення активного стану нейрона шляхом обчислення зваженої суми та додавання зсуву. Ця функція називається функцією активації.

5. Вузли, які повинні запускатися для вилучення функцій, визначаються на основі вихідного значення функції активації.

6. Кінцевий результат мережі потім порівнюється з необхідними позначеними даними набору даних, щоб обчислити кінцеву помилку вартості. Помилка вартості насправді говорить, наскільки «погана» мережа. Тому хочемо, щоб помилка була якомога меншою.

7. Вага регулюється за допомогою зворотного поширення, що зменшує похибку. Цей процес зворотного поширення можна розглядати як центральний механізм, якому навчаються нейронні мережі. По суті, він точно налаштовує ваги глибокої нейронної мережі, щоб зменшити собівартість.

Простіше кажучи, під час навчання нейронної мережі зазвичай обчислюємо втрати (значення помилки) моделі та перевіряємо, зменшено вони чи ні. Якщо помилка перевищує очікуване значення, потрібно оновити параметри моделі, такі як ваги та значення зміщення. Можемо використовувати модель, коли втрати нижчі за очікувану межу похибки(рис.1.5).

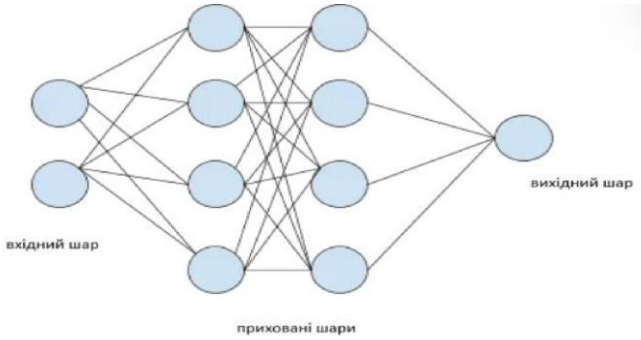


Рисунок 1.5. Візуалізація нейронної мережі

Нейронні мережі можна легко описати за допомогою наведеної вище схеми. Світло-блакитні кружечки представляють перцептрони, які обговорювали раніше, а лінії представляють зв’язки між штучними нейронами.

Розглядаючи один перцептрон, його роботу можна уявити наступним чином(рис.1.6.)

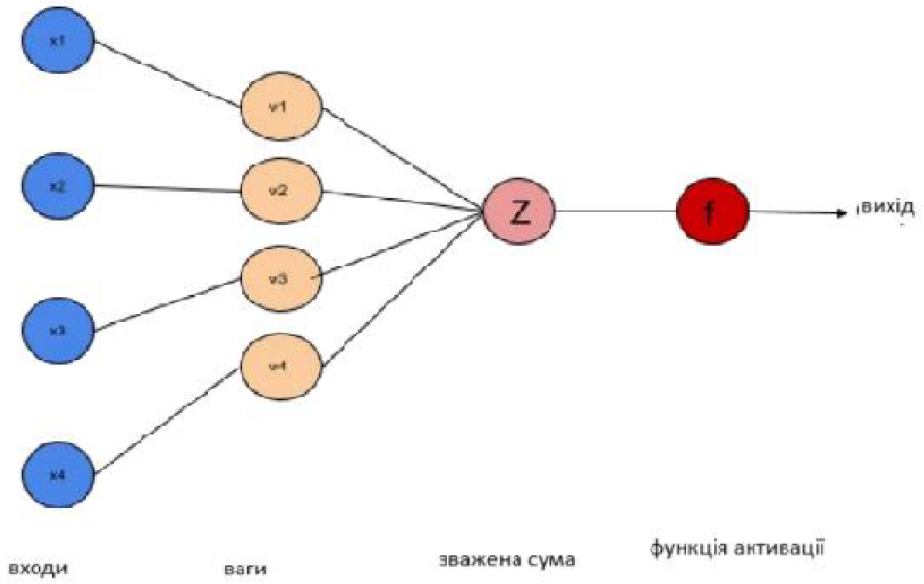


Рисунок 1.6. Один перцептрон

Коли вводити в модель дані з випадковими вагами, вона генерує їхню зважену суму. Розуміючи, як працює нейронна мережа, відповідно до цього значення функція активації визначає статус активації нейрона. Вихідні дані цього перцептрона можуть діяти як вхідні дані для наступного шару нейронів.

На даний момент розроблено багато типів нейронних мереж. Конволюційну нейронну мережу (CNN) і рекурентну нейронну мережу (RNN) можна вважати двома найпоширенішими типами нейронних мереж, які є основою для більшості попередньо навчених моделей у нейронних мережах.

1. Згорткові нейронні мережі[8]

CNN — це модель навчання під наглядом, яка складається з одного або кількох згорткових шарів. Спочатку ці згорткові шари застосовують функцію згортки на вході. Потім ці шари відправляються на наступний шар. Нейронам у шарі не обов’язково з’єднуватися з повним набором нейронів у наступному шарі.

Він з'єднується лише з його невеликою частиною. Результуючий результат є єдиним вектором, який включає оцінки ймовірності, які потім подаються на повністю пов’язані рівні. Згорткові нейронні мережі широко використовуються в областях розпізнавання зображень і обробки природної мови.

Працюючи з нейронними мережами, важливо розуміти, що таке зміщення та як воно працює. Зміщення можна розглядати як додатковий набір ваг у моделі, який не потребує жодних вхідних даних, і пов’язаний з виходом моделі, коли вона не має вхідних даних. Додавання константи до вхідного зсуву дозволяє зрушити функцію активації. Там зсув працює точно так само, як і в лінійному рівнянні:

$$y = mx + c$$

Тут упередженість в c.

Зміщення має важливе значення, оскільки воно завжди присутнє незалежно від вхідних даних і завдяки тому, що з ним мережа працює краще.

2. Рекурентні нейронні мережі

RNN — це широко поширена нейронна мережа, яка в основному використовується для розпізнавання мовлення та обробки природної мови (NLP). Він розпізнає послідовні характеристики даних і використовує шаблони для прогнозування наступного сценарію. У RNN вихідні дані попереднього кроку виступають як вхідні дані для наступного кроку.

RNN відрізняється своєю «пам’яттю», оскільки він отримує інформацію з попередніх входів, щоб впливати на поточний вхід і вихід.

3. Реалізація нейронної мережі

Розпізнавання рукописних чисел є однією з найпростіших нейронних мереж, яку може розробити новачок. Розробляємо та навчаємо нейронну мережу, яка передбачає певне зображення рукописної цифри. Тип нейронної мережі, який тут використовуємо, — це згорточна нейронна мережа (CNN).

Класичним прикладом нейронної мережі є розпізнавання рукописних цифр, коли нейронна мережа передбачає цифру(дод. А).

. Отже, давайте розглянемо кожну з цих функцій у коді та обговоримо, що там робиться.

loadDatset()

Ця функція в основному зосереджена на завантаженні необхідних даних і розділенні набору даних на чотири набори даних, які називаються tarinX, train_Y, test_X і test_Y. Тут train_X складається з рукописних зображень, які використовуються для навчання нашої моделі.

Train_Y — це мітка (фактичні чи рукописні цифри) зображень у наборі даних train_X. Тоді маємо test_X і test_Y. Цей тестовий набір даних використовується для оцінки системи або визначення точності розробленої моделі.

preprocess()

Ця функція використовується для перетворення значень пікселів із цілих чисел у числа з плаваючою точкою та нормалізації зображення в діапазоні 0–1. preprocess() повертає це нормалізоване зображення.[9]

modelDefenition()

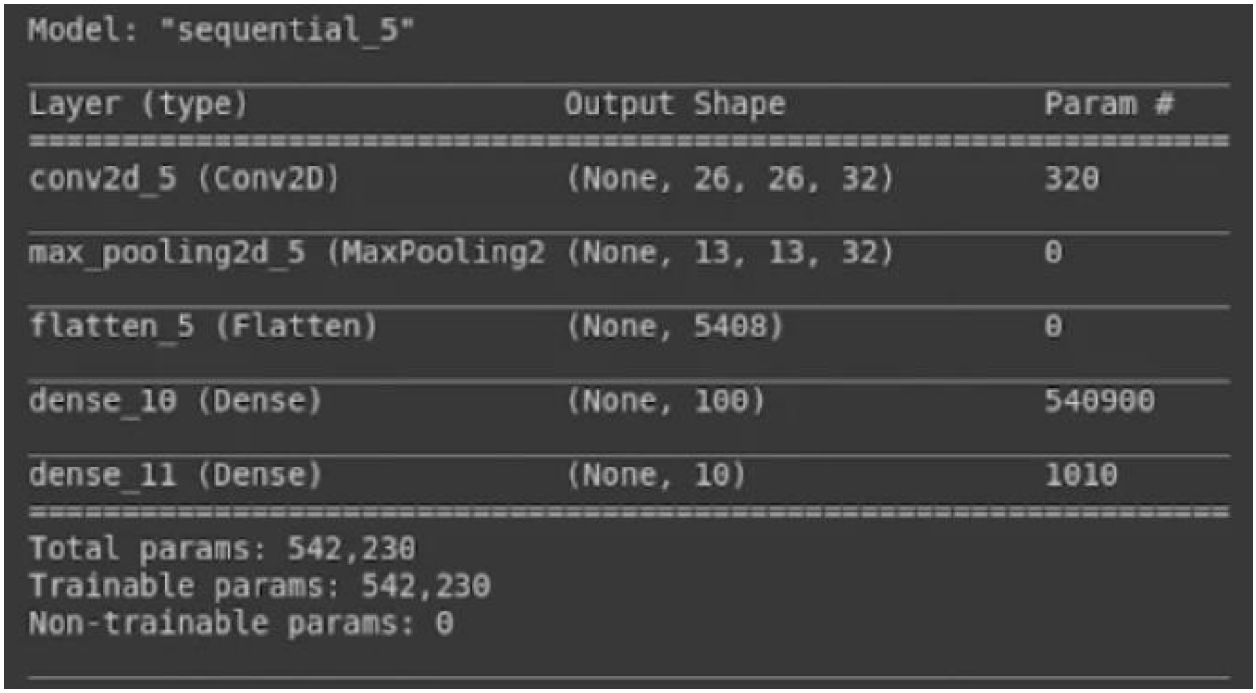
Це найважливіша функція, яку написали в цьому коді. Функція modelDefenition() використовується для створення нейронної мережі. Функції активації, які тут використали, це «relu» та «softmax».

Окрім цього, є кілька інших функцій активації, які можна використовувати в нейронних мережах, наприклад сигмоїдна функція, leakyRelu та tanh.

Оптимізатор, який використали в цій моделі, називається «opt». Оптимізатори використовуються для зміни атрибутів глибокої нейронної мережі, щоб зменшити значення втрат.

Окрім «opt», є кілька інших популярних оптимізаторів, таких як «adam optimizer» і «RMSProp». Можна використовувати їх відповідно до потреб нейронної мережі. Якщо потрібно більше дізнатися про те, як використовувати оптимізатори в Keras.

У Keras можна візуально побачити підсумок моделі за допомогою функції model.summary(). Ось результат, який отримали з моделлю(рис.1.7.).



```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 5408)              0
_________________________________________________________________
dense_10 (Dense)             (None, 100)               540900
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1010
=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 1.7. Результат моделі

modelEvaluation()

У цій функції оцінили модель за допомогою «k-кратної» перехресної перевірки. Перехресна перевірка — це процедура повторної вибірки, яка використовується для оцінки нейронної мережі на невеликій вибірці даних. Існує кілька методів оцінки моделі, таких як перехресна перевірка за винятком одного, перехресна перевірка за винятком однієї групи та вкладена перехресна перевірка.

Коли б можна використовувати нейронну мережу?

Це досить відоме питання для початківців більшу частину часу. Розуміння того, як працюють нейронні мережі, може допомогти визначити їх відповідне застосування. Нейронну мережу доцільно використовувати, коли:

1. Даних, необхідних для цілей навчання, достатньо.

2. Є необхідна обчислювальна потужність.

3. Коли не можна знайти чіткий зв’язок між даними та результатом.

Нейронні мережі для реального світу?

Насправді, нейронні мережі дуже корисні для вирішення багатьох складних проблем реального світу, таких як:

1. Біржове прогнозування

Робити прогнози на фондовій біржі за такими параметрами, як поточні тенденції, політична ситуація, громадська думка та поради економістів, можна робити за допомогою нейронних мереж. Для цього можемо використовувати нейронні мережі, такі як згорточні нейронні мережі та рекурентні нейронні мережі.[10]

2. Виявлення банківського шахрайства

Виявлення банківського шахрайства є одним із найважливіших випадків використання нейронних мереж. Там можна надати набір даних, який містить відомості про минуле банківського шахрайства. Потім можна виявляти та прогнозувати банківське шахрайство, навчаючи розроблену модель із заданим набором даних.

Інший практичний приклад нейронної мережі полягає у виявленні шахрайських транзакцій, використовуючи шаблони в даних транзакцій для позначення аномалій.

3. У системах рекомендацій

Системи рекомендацій є дуже популярним використанням нейронних мереж. RNN — це тип нейронної мережі, який переважно використовується в системах рекомендацій.

4. У пошукових платформах

Пошукові платформи, такі як Google і Yahoo, також використовують розширені типи нейронних мереж для покращення взаємодії з користувачем. Це значно покращує зручність використання пошукових систем.

1.4. Особливості застосування нейромереж у літальних апаратах

Останніми роками безпілотні літальні апарати (БПЛА), які іноді називають дронами, привернули значну увагу. Завдяки використанню пульта дистанційного керування БПЛА можна керувати без присутності пілота на відстані кількох миль. БПЛА використовуються в боях, розвідці, авіаударах, розслідуваннях та різних інших операціях. Крім того, БПЛА є корисним інструментом у різних галузях промисловості, і зараз вони використовуються для широкого спектру цілей. Наприклад, органи влади використовують БПЛА для запобігання

катастрофам, дистанційного зондування, екологічного моніторингу тощо. Їх також наймають такі компанії, як Amazon, UPS Inc. та інші для доставки продуктів. Крім того, БПЛА відіграють вирішальну роль у сільському господарстві, допомагаючи у спостереженні за посівами та застосуванні пестицидів і добрив. Крім того, персонал екстреної допомоги, а також служби екстреної медичної допомоги та ентузіасти використовують БПЛА для таких завдань, як рятувальні операції, медична допомога та рекреаційна зйомка.

Однак, замість нових застосувань БПЛА, в останні роки проблеми щодо конфіденційності та безпеки були підняті використанням систем БПЛА. Впровадження розважальних БПЛА в національний повітряний простір викликало занепокоєння щодо того, що некваліфіковані та неліцензійні пілоти потрапляють у заборонені зони та втручаються в роботу літаків. Неадекватні правила купівлі БПЛА можуть бути частиною проблеми. Наприклад, літак національної оборони був збитий приватним БПЛА трохи більше двох років тому. Найбільше занепокоєння викликають використання БПЛА для незаконного моніторингу та терористичні атаки. Щоб запобігти вищезазначеним інцидентам, необхідна технологія захисту від БПЛА, яка може ідентифікувати, класифікувати та нейтралізувати неліцензійні БПЛА, які збирають дані за допомогою різних датчиків. Нещодавно для класифікації та виявлення БПЛА проводилися численні дослідження, які досліджували способи ідентифікації БПЛА з використанням цілого ряду технологічних досягнень, таких як тепловізор, аудіо, відео, радіочастота (РЧ) і радар. За допомогою цих технологій існує багато традиційних методів ідентифікації або виявлення небажаних БПЛА, але більшість методів не змогли забезпечити адекватний рівень запобігання під час виявлення БПЛА.

Останніми роками галузі виявлення об’єктів, сегментації зображень і розпізнавання захворювань зазнали драматичних змін через нові переваги підходів машинного навчання (ML) і глибокого навчання (DL).

Отже, виявлення БПЛА набуло популярності в науковому співтоваристві після появи методів DL. Нові переваги ML і DL для виявлення БПЛА включають ефективність даних, зниження обчислювальної інтенсивності, автоматичне

навчання функцій, високу точність класифікації БПЛА та можливості наскрізного навчання. З іншого боку, є деякі недоліки ML і DL, наприклад, обмежена продуктивність у більш складних задачах виявлення БПЛА та моделі DL, які потребують великих обсягів мічених даних для навчання, що може бути обмеженням у сценаріях, коли отримання мічених даних є складним завданням або дорогим.

Враховуючи вищезазначені досягнення у виявленні та класифікації об’єктів за допомогою ML, у цьому огляді виявлення та класифікація БПЛА підлягають широкому вивченню щодо проблем, рішень і напрямків майбутніх досліджень із використанням ML та виявлення БПЛА, підкреслюючи переваги та обмеження методів, а також шляхи вдосконалення. Після цього подається ретельний критичний аналіз сучасного стану. Крім того, надається розгорнутий огляд інформації про набори даних для технологій класифікації виявлення БПЛА. Крім того, представлено виявлення та класифікацію БПЛА на основі підкріпленого навчання з детальним напрямком дослідження. Крім того, огляд стратегій виявлення БПЛА на основі гібридних датчиків містить детальні набори даних і чіткий напрямок дослідження.

В останні роки було проведено багато досліджень щодо систем БПЛА, і більшість робіт зосереджено на виявленні та класифікації об’єктів за допомогою зображень, створених за допомогою БПЛА, у застосуваннях класифікації сільськогосподарських культур, виявлення транспортних засобів, ідентифікації хвороби рослин і культур, лісове господарство, виявлення хвороб культур тощо. Навпаки, цей звіт зосереджений на виявленні БПЛА або дронів за допомогою п’яти різних технологій на основі алгоритмів ML і DL. За останні кілька років було опубліковано кілька звітів про виявлення та класифікацію БПЛА, і ключові відмінності між цими звітами та нашим звітом про дослідження зазначено в таблиці 1.

Таблиця 1. Внесок цього огляду в інші роботи з класифікації та виявлення БПЛА на основі ML

1	Огляд виявлення та класифікації безпілотних літальних апаратів за допомогою ML до 2019 року. Дослідження дає обмежене уявлення про продуктивність оригінальних підходів до виявлення та посилання.
2	Огляд зв'язку на основі БПЛА за допомогою ML, зосереджуючись на управлінні ресурсами, моделюванні каналів, місцезнаходження та безпеці до 2019 року.
3	Огляд технічної класифікації та методів реалізації виявлення та відстеження БПЛА в міському середовищі Інтернету речей і надання обмеженої кількості посилань до 2023 року.
4	Опитування щодо виявлення БПЛА на основі DL з акцентом на радіолокаційних технологіях до 2021 року.
5	Огляд мереж радарних датчиків на основі ML для виявлення та класифікації багатороторних БПЛА до 2020 року.
6	Опитування щодо виявлення несанкціонованих БПЛА до 2022 року. Однак дослідження не охоплює конкретні типи ML та сучасні підходи до виявлення.
7	Огляд стратегій виявлення безпілотників, які наголошують на використанні DL з мультисенсорними даними до 2019 року.
8	Огляд ідентифікації, нейтралізації та виявлення дронів, з меншим акцентом на методах виявлення. Дослідження в основному зосереджено на проектуванні системи з нормативної точки зору, за винятком найсучасніших методів виявлення та посилань, охоплених до 2021 року.

РОЗДІЛ 2. ТЕМАТИЧНЕ ДОСЛІДЖЕННЯ ВИБРАНИХ ХАРАКТЕРИСТИК ЛІТАЛЬНОГО АПАРАТУ

Для цілей поточного дослідження моделювання дистанції зльоту (TOD) вибрано як тематичне дослідження, щоб визначити, чи можна ШНМ використовувати для заміни існуючих детермінованих моделей. Цей розділ починається з визначення TOD, який використовується в цій роботі, після чого йде огляд того, як розробляються поточні детерміновані моделі TOD без використання ШНМ.[11-13]

2.1 Визначення дистанції зльоту

Розділ 525 (Літаки транспортної категорії) частини V (Керівництво з льотної придатності) Канадських авіаційних правил (CAR) описує юридично визнане визначення дистанції зльоту як:

(a) Дистанція зльоту на сухій злітно-посадковій смузі є більшою з:

(1) Горизонтальна відстань уздовж траєкторії зльоту від початку зльоту до точки, в якій літак знаходиться на 35 футів над злітною поверхнею, визначена відповідно до 525.111; для сухої злітно-посадкової смуги;

(2) 115 відсотків горизонтальної відстані вздовж траєкторії зльоту з усіма працюючими двигунами від початку зльоту до точки, в якій літак знаходиться на висоті 35 футів над злітною поверхнею, як визначено процедурою відповідно до 525.111.

(b) Дистанція зльоту на мокрій злітно-посадковій смузі є більшою з:

(1) Дистанція зльоту на сухій злітно-посадковій смузі, визначена відповідно до пункту (a) цього розділу;

(2) Горизонтальна відстань уздовж траєкторії зльоту від початку зльоту до точки, в якій літак знаходиться на висоті 15 футів над злітною поверхнею, досягнута таким чином, що відповідає досягненню V_2 до досягнення 35 футів над злітною поверхнею, визначеної відповідно до 525.111 для мокрої злітно-посадкової смуги.

(c) Якщо злітна дистанція не включає чисту дорогу, розбіг дорівнює злітній дистанції. Якщо дистанція зльоту включає чисту смугу:

(1) Розбіг при зльоті на сухій злітно-посадковій смузі є більшим із:

(i) Горизонтальна відстань уздовж траєкторії зльоту від початку зльоту до точки, рівновіддаленої між точкою, в якій досягається V_{LOF}, і точкою, в якій літак знаходиться на 35 футів над злітною поверхнею, як визначається відповідно до 525.111; для сухої злітно-посадкової смуги;

(ii) 115 відсотків горизонтальної відстані вздовж траєкторії зльоту, з усіма працюючими двигунами, від початку зльоту до точки, рівновіддаленої між точкою, в якій досягнуто V_{LOF}, і точкою, в якій літак знаходиться на 35 футів над поверхнею зльоту, визначається за процедурою, що відповідає 525.111.

(2) Розбіг при зльоті на мокрій злітно-посадковій смузі є більшим із:

(i) Горизонтальна відстань уздовж траєкторії зльоту від початку зльоту до точки, в якій літак знаходиться на 15 футів над злітною поверхнею, досягнута таким чином, що відповідає досягненню V_2 до досягнення 35 футів над злітною поверхнею, як визначено відповідно до 525.111 для мокрої злітно-посадкової смуги;

(ii) 115 відсотків горизонтальної відстані вздовж траєкторії зльоту з усіма працюючими двигунами від початку зльоту до точки, рівновіддаленої між точкою, в якій досягнуто V_{LOF}, і точкою, в якій літак знаходиться на 35 футів над поверхнею зльоту, визначається процедурою, що відповідає 525.111. “

З цього визначення розуміється, що TOD може відрізнятися залежно від стану злітно-посадкової смуги (суха чи мокра), наявності вільної смуги та кінцевого результату зльоту (OEI, AEO, ASD). Траєкторія зльоту, визначена стандартом CAR 525.111, на якому базується визначення TOD, представлена на рисунку 2.1. Поточне дослідження стосується лише сценарію зльоту до 35 футів над рівнем землі. Існує чотири можливі сценарії: зліт з усіма працюючими двигунами (AEO), зліт з одним непрацюючим двигуном (OEI), прискорена зупинка з AEO (ASDAEO) і прискорена зупинка з OEI (ASDOEI). OEI та AEO призводять до зльоту літака зі злітно-посадкової смуги, тоді як ASDAEO та ASDOEI обидва призводять до відхиленого зльоту. На рисунку 2.2.A та рисунку 2.2.B показано представлення TOD для сценаріїв OEI та AEO за вологих та сухих

умов, позначених як TODN-1 та TODN відповідно. На рисунку 2.2.C і рисунку 2.2.D показані однакові сценарії, якщо присутні вільної дороги. Свободна смуга – це зона за межами асфальтованої злітно-посадкової смуги, вільна від перешкод і під контролем керівництва аеропорту. Довжина вільної смуги може бути включена в довжину доступної злітної дистанції. Відстані ASDAEO та ASDOEI представлені на рисунках 2.3.A та 2.3.B для відхилених злетів, позначених як ASDN-1 та ASDN відповідно. Стандарти CAR 525.109 і 525.113 визначають сертифіковані дистанції зльоту як функції відстаней, показаних на рисунках 2.2. і 2.3. Таблиця 2 підсумовує їх визначення.

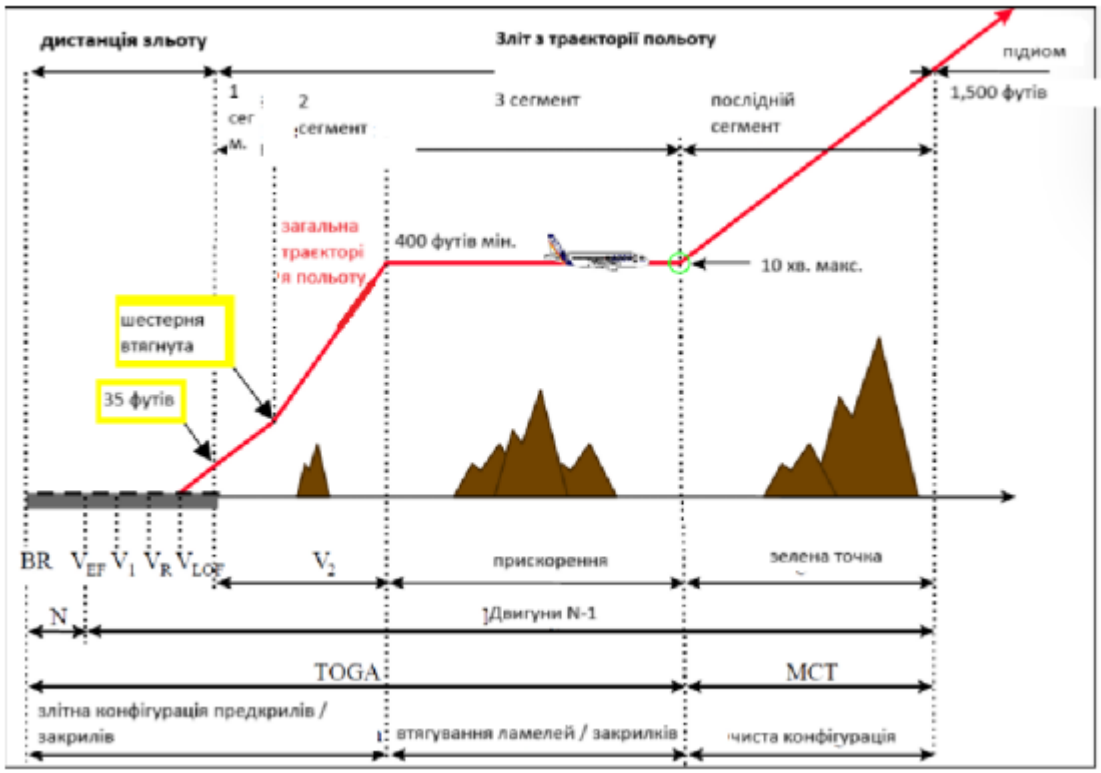


Рисунок 2.1. Ілюстрація траєкторії зльоту

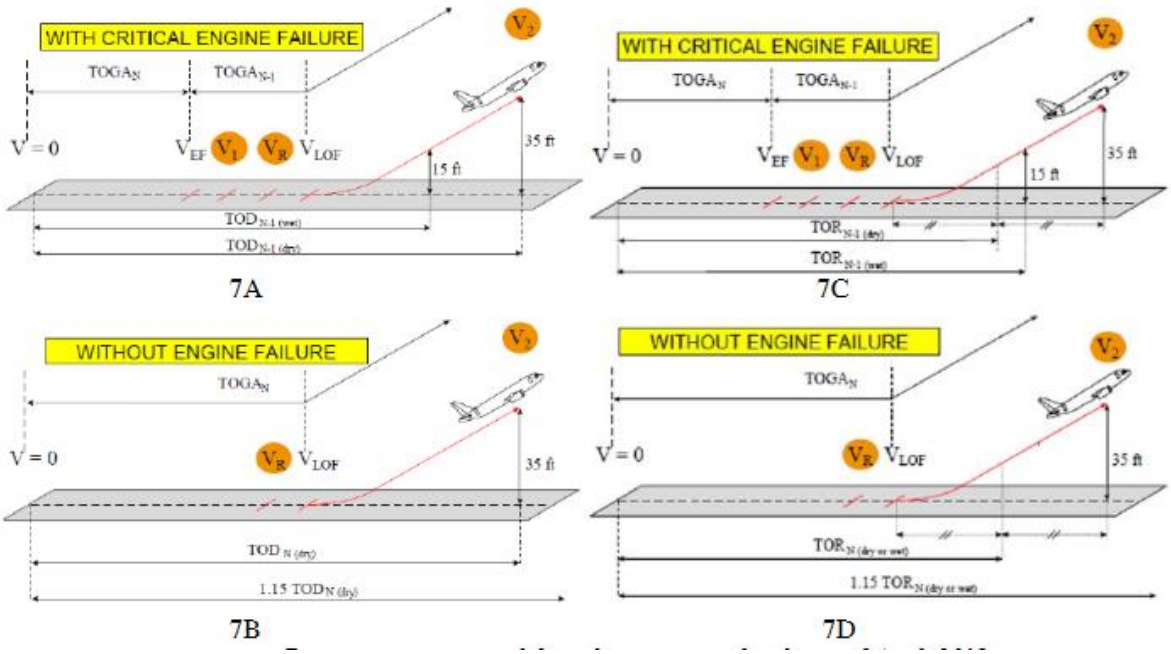


Рисунок 2.2. Відображення сценарію TOD для OEI та AEO

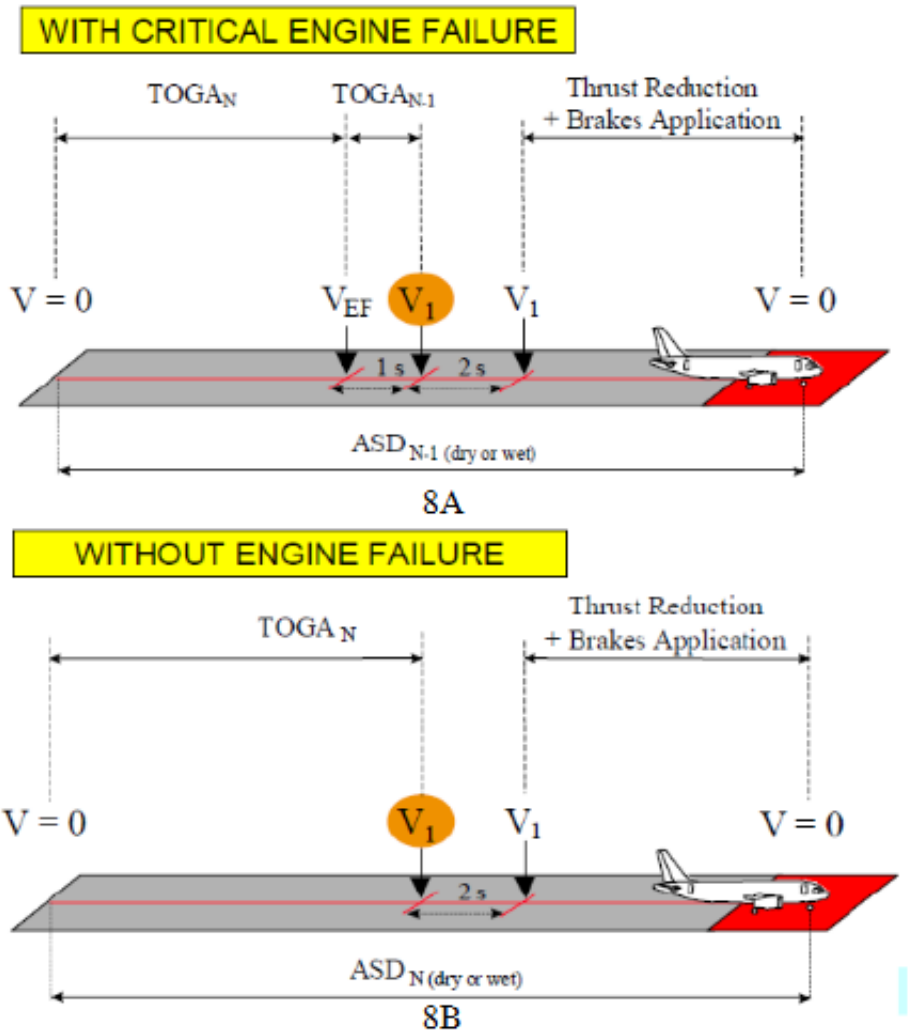


Рисунок 2.3. Відображення сценарію TOD для ASDAEO та ASDOEI

Таблиця 2. Визначення дистанції зльоту

Вільність дороги	Визначення TOD	Регулювання
Немає вільної дороги	TOD_{dry}=max($TOD_{N-1,dry}$,1.15$TOD_{N,dry}$)	CAR 525.113 / FAR 25.113 / CS 25.113
	TOD_{wet} = max (TOD_{dry},$TOD_{N-1,wet}$)	
	ASD_{dry}=max($ASD_{N-1,dry}$,$ASD_{N,dry}$)	CAR 525.109 / FAR 25.109 / CS 25. 109
	ASD_{wet}=max(ASD_{dry},$ASD_{N-1,wet}$,$ASD_{N,wet}$)	
З вільною дорогою	TOR_{dry}=max($TOR_{N-1,dry}$,1.15$TOR_{N,dry}$)	CAR 525.113 / FAR 25.113 / CS 25.113
	TOR_{wet}=max($TOR_{N-1,wet}$,1.15$TOR_{N-1,wet}$)	

Різні швидкості, що зустрічаються під час зльоту (ілюстровані на рисунку 2.2 і рисунку 2.3), визначені в таблиці 3 та їх обмеження, встановлені нормативними документами, підсумовані в таблиці 4.

Таблиця 3. Визначення швидкості зльоту.

Швидкість зльоту	Визначення	Регулювання
V_{EF}	V_{EF} — відкалібрована повітряна швидкість, на якій знаходиться критичний двигун передбачається невдача. V_{EF} має бути обраний заявником, але не може бути меншим за V_{MCG}	CAR 525.107 FAR 25.107 CS 25.107
V_1	V_1 — це максимальна швидкість, на якій екіпаж може прийняти рішення про відхилення зліт, і забезпечується зупинка повітряного судна в межах злітно-посадкова смуга. Час між V_{EF} і V_1 розпізнається як 1 секунда.	
V_R	V_R – швидкість, з якою пілот починає обертання, при відповідною швидкістю близько 3° за секунду	

V_{LOF}	V_{LOF} – це калібрована повітряна швидкість, на якій літак вперше стає повітряно-десантний. Отже, це швидкість, з якою ліфт долає вага.	
V_2	V_2 — мінімальна швидкість підйому, яку необхідно досягти на висоті 35 футів над поверхнею злітнопосадкової смуги, у разі відмови двигуна	
V_{MBE}	Максимальна швидкість поглинання шини під час екстремальних ситуацій гальмування	CAR 525.109 FAR 25.109 CS 25.109
V_{TIRE}	Максимальна швидкість руху обмежена відцентровими силами шини та теплом висота.	

Таблиця 4. Обмеження швидкості зльоту

Обмежена швидкість	Обмеження		Регулювання
V_{EF}	$V_{EF} \geq V_{MCG}$		CAR 525.107 / FAR 25.107 / CS 25.107
V_1	$V_{MCG} \leq V_{EF} \leq V_1$		CAR 525.107 / FAR 25.107 / CS 25.107
	$V_1 \leq V_{MBE}$		CAR 525.109 / FAR 25.109 / CS 25.109
V_R	$V_R \geq 1.05 V_{MCA}$		CAR 525.107 / FAR 25.107 / CS 25.107
V_{LOF}	Геометричний	$V_{LOF} \geq \mathbf{1}.05 V_{MU(N-1)}$ $V_{LOF} \geq 1.08 V_{MU(N)}$	CAR 525.107/FAR 25.107/AC 25-7A
		$V_{LOF} \geq \mathbf{1.04} V_{MU(N-1)}$ $V_{LOF} \geq 1.08 V_{MU(N)}$	CS 25.107
	Аеродинамічний	$V_{LOF} \geq 1.05 V_{MU(N-1)}$	CAR 525.107/FAR 25.107/CS 25.107

		$V_{LOF} \geq 1.10V_{MU(N)}$	
	Шина	$V_{LOF} \leq V_{TIRE}$	CAR 525.109/FAR 25.109/CS 25.109
V_2	$V_2 \geq 1.1V_{MCA}$		CAR 525.107 / FAR 25.107 / CS 25.107

2.2 Розрахунок дистанції зльоту

Розрахунок TOD розбивається на різні сегменти, які потрібно додати разом, щоб отримати остаточну відстань, як показано на рисунку 2.4.

Рисунок 2.4. Процес розрахунку TOD

Уся дистанція прискорення двигуна простягається від відпускання гальма до точки, де є швидкість обертання, VR досягнуто. Схема вільного тіла сил, що діють на літак під час зльоту, зображена на рисунку 2.5. Як сили, що діють на літак, змінюються вздовж дистанції зльоту, для розрахунку можна використовувати процес інтегрування кроків змінюється прискорення, яке потім можна використовувати для розрахунку загальної пройденої відстані[14-16].

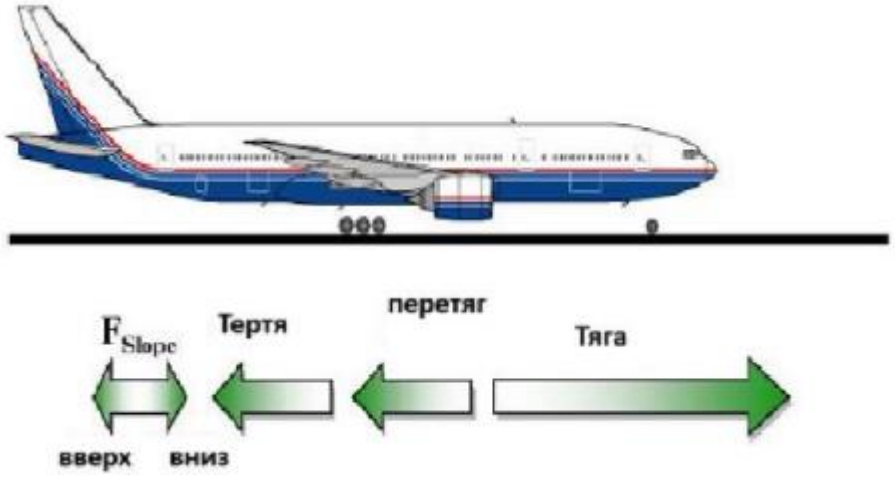


Рисунок 2.5. Сили, що діють на літак під час зльоту

Рівняння показує суму сил, що діють уздовж осі x на діаграмі вільного тіла.

$$\sum forces = T - D - \mu(W - L) - W sin\varphi$$

Де T — тяга, створювана двигунами, D — опір, μ — коефіцієнт тертя кочення злітно-посадкової смуги, W — вага літака на початку розбігу, L — підйомна сила, створювана підйомними поверхнями, а φ — нахил злітної смуги. Прискорення вздовж розбігу розраховується за допомогою наступного рівняння, де маса виражається як вага літака над постійною сили тяжіння Землі g

$$a = \frac{\sum \quad forces}{mass} \frac{g}{w} [T - D - \mu(W - L) - W \; sin \; sin \; \varphi \;]$$

Дане рівняння можна виразити через його аеродинамічні коефіцієнти, як показано в наступному рівнянні.

$$\alpha = \frac{g}{w}[T - \mu W - (CD - \mu CL)qS - W \; sin \; sin \; \varphi \;]$$

Де CD — коефіцієнт лобового опору, CL — коефіцієнт підйомної сили, q — динамічний тиск, а S — еталонний площа крила. Динамічний тиск виходить з наступного рівняння.

$$q = 0.5 \, pV_2$$

Середня швидкість при невеликій зміні швидкості визначається як:

$$V = \frac{\Delta s}{\Delta t}$$

Де Δs — це приріст відстані за приростом швидкості, а Δt — час приросту за приріст швидкості. Зміна швидкості пов'язана з прискоренням :

$$\propto = \frac{\Delta v}{\Delta t}$$

Де a — прискорення, а ΔV — приріст швидкості. Додаткову відстань між двома точками вздовж розбігу можна отримати шляхом поєднання рівнянь 5 і 6.

$$\Delta S = \frac{\overline{v \, \Delta V}}{\frac{g}{w}[T - \mu W - (CD - \mu CL)(0.5pV2)S - W \sin \varphi]}$$

Повна дистанція прискорення двигуна обчислюється як сума всіх додаткових відстаней Δs між V_0 і V_R.

Обчислення дистанції прискорення двигуна[17]

Відстань прискорення виходу двигуна поширюється від відмови двигуна (VEF) до швидкості обертання V_R. Одразу після відбувається відмова двигуна, двигун поступово втрачає тягу, що впливає на розраховану дистанцію. Це називається обертанням двигуна. Коефіцієнт спіндауну - це співвідношення між фактичною тягою, яку створює двигуни під час обертання, поділені на налаштування тяги двигуна, вибраного пілотом у кабіні. Графіки продуктивності, що показують співвідношення залишкової тяги двигуна до тяги при зльоті та часу від двигуна використовуються для визначення залишкової тяги в будь-який час від події відмови двигуна. Спиндаун коефіцієнт множиться на значення тяги, щоб отримати фактичну тягу під час відмови двигуна. По порядку використовувати правильне значення коефіцієнта обертання двигуна, який є функцією часу, ступінчастою інтеграцією. Розрахунок відстані повинен проводитися з урахуванням часу, а не швидкості. Використання з коефіцієнтом обертання 1 і швидкістю при t = 0 (V_1), перше початкове припущення прискорення може знайдено, що відповідає миттєвому прискоренню. Швидкість через 1 секунду визначається за допомогою початкове припущення прискорення. Отримавши перші два миттєві прискорення, можна отримати перше середнє прискорення за 1 секунду розрахований. Нове значення швидкості за 1 секунду можна перерахувати за допомогою середнього прискорення, і це процес можна повторювати, доки значення повітряної швидкості не наблизиться до кінцевого значення.

Розрахунок відстані спалаху

Відстань спалаху поширюється від початку обертання (VR) до висоти 35 футів над рівнем землі. Для сценарію AEO висота 35 футів над рівнем землі відповідає V35, тоді як для сценарію OEI висота 35 футів над рівнем землі відповідає V2 з несправним двигуном. Під час льотних випробувань експериментальні значення повітряної швидкості V35 і V2 отримані для різних сценаріїв і умов (тобто для різне співвідношення тяги до ваги). Відстань спалаху для сценарію AEO можна розрахувати за допомогою рівняння наведеного нижче,

а відстань спалаху для сценарію OEI можна розрахувати за допомогою наступного рівняння, де ΔtR-35 — це час спалаху від обертання до 35 футів.

$$s_{flare}AEO = \frac{VR + V35}{2}\Delta_{tR-35}$$

$$Ss_{flare}, OEI = \frac{VR + V2}{2}\Delta_{tR-35}$$

Розрахунок відстані гальмування

Відстань уповільнення тягнеться від V1 і закінчується в момент, коли швидкість руху дорівнює нулю. Правила передбачають, що для швидкості V1 необхідно враховувати час розпізнавання 2 секунди. Дросель налаштування та конструкція гальма створюють додаткові сили, які слід враховувати в цьому сценарії, які впливають на значення гальмівного коефіцієнта тертя μB. Сценарій уповільнення зображено на діаграмі вільного тіла на рисунку 2.6.

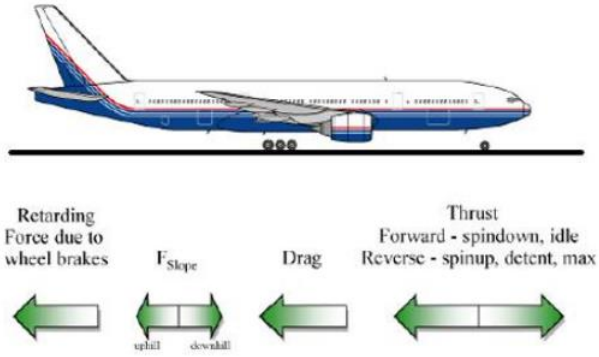


Рисунок 2.6. Сили, що діють на літак під час уповільнення при зльоті

Рівняння показує суму сил, що діють уздовж осі x на діаграмі вільного тіла.

$$\sum \quad forces = T - D - \mu B(W - L) - W\ sin\ sin\ \varphi$$

Де T — тяга, створювана двигунами, D — опір, μB — коефіцієнт тертя гальмування, W — вага літака на початку розбігу, L — підйомна сила, створювана підйомними поверхнями, а φ — нахил злітної смуги. Значення μB отримані експериментально під час льотних випробувань або надані виробником гальм. Уповільнення вздовж розбігу розраховується за допомогою рівняння, де маса виражається як вага літака над постійною сили тяжіння Землі g.[18]

$$\propto = \frac{\sum \quad forces}{mass} = \frac{g}{w}[T - D - \mu B(W - L) - W\ sin\ sin\ \varphi\ ]$$

Дане рівняння можна виразити через його аеродинамічні коефіцієнти, як показано в рівнянні нижче.

$$\alpha = \frac{g}{w}[T - \mu BW - (CD - \mu BCL)qS - W\ sin\ sin\ \varphi\ ]$$

Де CD — коефіцієнт опору, CL — коефіцієнт підйомної сили, q — динамічний тиск, а S — базова площа крила. Динамічний тиск виходить з рівняння 4. Уповільнення від V1 до Vfull stop можна розрахувати за допомогою рівняння наведеного вище. Додаткову відстань між двома точками вздовж сегмента уповільнення можна отримати за допомогою рівняння нижче.

$$\Delta S = \frac{\nabla \Delta V}{\alpha} \frac{\nabla \Delta V}{\frac{g}{w}[T - \mu W - (CD - \mu BCL)(0.5V2)S - W\ sin\ sin\ \varphi\ ]}$$

Відстань уповільнення розраховується як сума всіх додаткових відстаней Δs між V1 до Vfull stop.

Підсумовування всіх дистанцій зльоту Остаточні відстані для кожного з результатів зльоту розраховуються наступним чином:

T$ODAEOorTODN$ = двигун прискорена відстань

T$ODOEIorTODN-1$ = Усі відстані прискорення двигуна + відстань виходу двигуна

A$SDAEOorASDN$ = відстань прискорення двигуна + відстань уповільнення

A$SDOEIorASDN-1$ = Двигун відстань прискорення + відстань уповільнення

2.3 Оцінка детермінованої моделі TOD

Процес розробки детермінованої моделі TOD є складним, оскільки він об’єднує численні моделі з різних аерокосмічних дисциплін, працює з великими обсягами даних, вимагає експертних знань і залежить від багатьох зовнішніх факторів, які включають:

1. Атмосферні умови та висота в аеропорту (температура, тиск, щільність повітря, швидкість і напрям вітру)

2. Умови злітно-посадкової смуги аеропорту (накопичення відкладень на злітно-посадковій смузі, нахил злітно-посадкової смуги)
3. Експлуатаційні можливості ПС (доступна тяга, структурно обмежені швидкості зльоту, VR, CL тощо)
4. Обмеження щодо зльоту літака з правил (VMBE, VTIRE)
5. Конфігурація літака (ECS увімкнено/вимкнено, антиожеледна система увімкнено/вимкнено, налаштування закрилків, конфігурація двигуна)
6. Вага та баланс
7. Час розпізнавання пілота (час на V_1)

Враховуючи прогалини, виявлені в літературі, та оцінку існуючої детермінованої моделі TOD, обрано як тематичне дослідження для цього дослідження, оскільки це високоскладна нелінійна регресійна оптимізаційна задача, яка є функцією різноманітного набору параметрів; вимагає багато експертних знань для розвитку; передбачає обробку великих обсягів даних; і потребує значного розвитку часу та зусиль під час використання детермінованих моделей. Базуючись на огляді літератури, традиційна нейронна мережа прямого зворотного поширення з двома або більше прихованими шарами буде перевірена з використанням великих наборів даних, враховуючи, що цей тип моделі показав найбільшу перспективу при застосуванні до цього типу проблеми. Було проаналізовано два типи наборів даних:

1) детермінований набір даних, згенерований на основі існуючої моделі дистанції зльоту характеристик літака;

2) недетермінований набір даних, що складається з емпіричних даних польоту від датчиків на борту літака.

2.4. Особливості маршрутизації малих літальних апаратів

Запланований маршрут польоту повітряного судна містить послідовність маршрутних точок. Диспетчер може дозволити літальному апарату літати таким чином, щоб пропустити деякі з них. Ця техніка називається «пряма маршрутизація». Це часто економить час і пальне, і тому зазвичай вітається (і іноді вимагається) пілотами. Ось чому в документі ICAO Doc 4444 зазначено: «З

урахуванням обмежень повітряного простору, робочого навантаження диспетчера та щільності трафіку, а також за умови своєчасної координації, літаку, коли це можливо, слід пропонувати найпряміший маршрут». Хоча використання прямої маршрутизації має багато переваг, контролери повинні завжди ретельно вивчати наслідки цього варіанту, оскільки він також може спричинити проблеми.

Як засіб вирішення конфліктів пряма маршрутизація подібна до векторизації, оскільки обидва методи використовуються для досягнення горизонтального ешелонування шляхом зміни траєкторії літака. Однак є суттєва різниця з навігаційної точки зору – літак, що летить по прямому маршруту, підтримує власну навігацію, а векторний – ні[19].

Безперечно, пряма маршрутизація є технікою, якій найбільше віддають перевагу як пілоти, так і диспетчери. Перші вважають, що це дозволить їм дістатися місця призначення раніше, а другі вважають, що вони надають більш ефективні послуги. Переваги прямої маршрутизації включають:

- Компенсація затримки. Якщо рейс затримався з будь-якої причини (пізній виліт, уникнення несприятливих погодних умов тощо), дозвіл на політ за прямим маршрутом може компенсувати це, принаймні частково.
- Підвищення ефективності польоту за рахунок зменшення витрати палива.
- Скорочення часу польоту. Це особливо корисно в надзвичайних ситуаціях, коли раннє досягнення вибраного аеродрому посадки може бути вирішальним.
- Вирішення конфліктів. Подібно до векторування, прямий маршрут може змінити траєкторію польоту, що призведе до досягнення необхідного горизонтального ешелонування або забезпечить додатковий час для досягнення вертикального ешелонування. Однак, на відміну від векторизації, пряма маршрутизація скорочує траєкторію польоту, таким чином досягаючи безпеки та ефективності.

- Нейтралітет. Якщо використовується після векторизації, пряма маршрутизація може компенсувати подовжену траєкторію польоту (і час), щоб досягти нульового чистого ефекту.
- Пом'якшення турбулентності в сліді (як альтернатива паралельному зсуву)
- Простіший розрахунок конфлікту. Якщо літак летить за прямим маршрутом, його напрямок залишатиметься незмінним, а коливання швидкості в більшості випадків будуть зведені до мінімуму. Природно, що для літальних апаратів, що набирають/знижують, або за певних погодних умов це твердження не буде вірним.
- Літак продовжує політ за допомогою власної навігації. Це особливо корисно у разі втрати радіозв'язку, оскільки траєкторія польоту буде більш передбачуваною.

2.5. Маршрутизація літального апарату з використанням Google map

Маршрутизація літака передбачає планування найбільш ефективного та безпечного маршруту польоту від місця відправлення до пункту призначення з урахуванням таких факторів, як погода, повітряний рух, економія палива та нормативні обмеження. Google Maps, широко використовуваний геопросторовий інструмент, надає потужні функції візуалізації та картографування, які можна використовувати для планування маршруту. У цій анотації розглядається концепція використання Карт Google для допомоги у прокладці маршрутів літаків.

Інтеграція Карт Google у маршрутизацію літаків може слугувати інноваційною платформою для візуалізації маршрутів, рельєфу та даних про погоду в реальному часі. Хоча Карти Google традиційно не використовуються в авіації через спеціалізовані вимоги аеронавігації, вони можуть виступати як додатковий інструмент для створення попередніх маршрутів польоту[20]. У цьому підході використовується API Google Maps для 3D-візуалізації рельєфу, побудови маршрутних точок і супутникових зображень для визначення потенційних перешкод або обмеженого повітряного простору.

Основні міркування щодо впровадження цієї концепції включають адаптацію Карт Google для відображення даних про авіацію, таких як повітряні шляхи, заборонені для польотів зони та межі управління повітряним рухом. Поєднання цих можливостей з оновленнями в реальному часі, такими як накладення погоди в реальному часі та висоти польоту, дозволяє динамічно коригувати маршрут для оптимізації безпеки та ефективності.

РОЗДІЛ 3. МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ І ПРОЕКТУВАННЯ НЕЙРОМЕРЕЖ

3.1 Процес розробки нейронної мережі

Рисунок 3.1 ілюструє процедуру, яка була використана для розробки програми нейронної мережі. Перший крок полягає у визначенні бажаних цілей і вимог до НМ. Вони використовуються, щоб визначити, коли зупинитися оптимізація мережі. Наступним кроком є аналіз властивостей набору даних (тобто розподіл набору даних, розмір, формат файлу тощо), щоб правильно вибрати оптимальні властивості набору даних. Тоді набір даних попередньо оброблені, щоб зберегти лише потрібні властивості даних і полегшити інтеграцію даних із кодом Python середовище. На основі характеристик набору даних можна визначити попередню архітектуру, яка є згодом оптимізовано за допомогою топологічного дослідження. Мережа навчається за допомогою навчального набору даних і перевірено за допомогою набору даних тестування. Якщо цілі мережі не досягнуті, архітектура мережі повинна бути досягнута оновлюється, і цей процес повторюється до отримання бажаних результатів навчання та тестування[21]. Одного разу мережеві вимоги виконуються, матриця ваг навченої мережі зберігається в локальному файлі, який використовуватиметься додатком інструменту прогнозування.

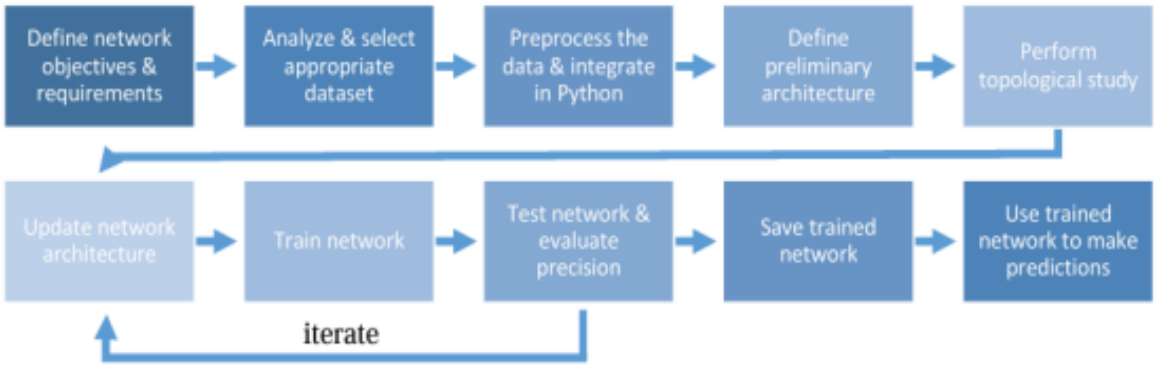


Рисунок 3.1. Процес розробки нейронної мережі

3.2 Мета створення нейромережі та визначення вимог

Цілі та вимоги нейронної мережі залежать від того, для чого вона буде використовуватися. Для розрахунку TOD дані, які містяться в посібниках з експлуатації літака (AFM), як правило, прийнятні, якщо помилки менші або рівні до 1% даних льотних випробувань. Для цілей цього дослідження бажано мінімально можливу похибку. 20 годин було довільно обрано як максимальний час навчання та тестування. Час було обрано на основі зручного достатньо тривалого часу для NN для вилучення складних моделей прийнятної точності для демонстраційних цілей цього дослідження. Оскільки інструмент, який використовує навчену NN, потенційно може використовуватися на борту літака або бути частиною розробки версій програмного забезпечення AFM, цей інструмент повинен мати можливість дуже швидко обчислювати прогнози TOD. Було встановлено вимогу, щоб цей час був рівним або меншим за 1 секунду[22]. Це дослідження використовується як доказ концепції, але його було б вигідно розвивати у спосіб, який також можна було б адаптувати для роботи з іншими сценаріями характеристик повітряних суден, ніж TOD (тобто прогнози посадкової відстані). Він також повинен мати можливість легко інтегруватися з існуючою технологією (тобто з програмним AFM або електронною польотною сумкою). Таблиця 5 підсумовує вимоги до NN.

Таблиця 5. Резюме вимог і цілей NN

Параметр нейронної мережі	Вимоги та цілі
Максимальний час навчання та тестування	20 годин
Максимальний час роботи інструменту прогнозування	1 сек
Можливість повторного використання мережі	Можливість повторного використання для альтернативних застосувань
Інтеграція з існуючою технологією	Можливість легкої інтеграції з існуючою технологією
Викидні точки даних	Здатний мати справу з викидами

3.3 Процес вибору набору даних

Перший із двох наборів даних, використаних у поточному дослідницькому проекті, було згенеровано з детермінованої моделі дистанції зльоту на основі принципів, і відповідно до частини 25 Федеральних авіаційних правил (FARs) і частини V Канадських авіаційних правил (CARs) (Стандарти льотної придатності для літаків транспортної категорії). Другий набір даних було отримано з ініціативи NASA DASHlink (Discovery in Aeronautics Systems Health), веб-інструмент для спільних досліджень інтелектуального аналізу даних і працездатності систем. Основною метою DASHlink є поширення інформації про найновіші алгоритми інтелектуального аналізу даних і справності систем, дані та дослідження. Цей набір даних надається у формі даних польоту, зібраних із датчиків на борту невпізнаного літака (для юридичних цілей).

3.3.1 Набір даних детермінованої дистанції зльоту

Параметри першого набору даних наведено в таблиці 6. На основі детермінованих зв’язків, що описують обчислення TOD, було вирішено, що для розробки NN будуть змінюватися лише найвпливовіші фактори, що впливають на продуктивність TOD. Отже, усі інші параметри, задіяні в детермінованій моделі TOD, залишалися постійними та перераховані в першому стовпці таблиці 6. Параметри, що представляють інтерес, перераховані у другому стовпці таблиці 6,

і це вага літака, барометрична висота, температура, швидкість вітру та ухил злітно-посадкової смуги.[23]

На рисунку 3.2 показані параметри як вхідні та вихідні значення нейронної мережі, де дистанція зльоту відповідає довжині поля зльоту (TOFL), яка є найбільш обмежувальною з TOD_{OEI}, TOD_{AEO} та TOD_{ASD}.

На рисунку 3.3 показано зразок необробленої детермінованої моделі TOD, яка використовується для створення набору даних. Це текстовий файл (.txt), який полегшує налаштування набору даних, а також оптимізує розмір файлу. Це важливий вибір, оскільки налаштування дозволить вибрати лише бажані параметри для NN або змінити набір даних, щоб краще відповідати оптимізації архітектури NN. Розмір файлу також важливий, оскільки цей набір даних може бути дуже великим (від 70 МБ до 61 ГБ залежно від параметрів і кількості вибраних тестів).

Таблиця 6. Параметри, що впливають на TOD, які можна отримати з детермінованої моделі TOD

Постійні вхідні параметри	Зміна вхідних параметрів	Вихідні параметри
Конфігурація літака та двигуна	Вага	TODOEI
Конфігурація клапана	Барометрична висота	TODAEO
Збалансований V1	Температура	TODASD
V2	Швидкість вітру	TOFL
Стан злітно-посадкової смуги (сухий)	Нахил ЗПС	
Налаштування тяги		
ECS двигуна (вимк.)		
Anti-Ice (Вимк.)		
BTMS		
Ефекти MMEL/CDL		

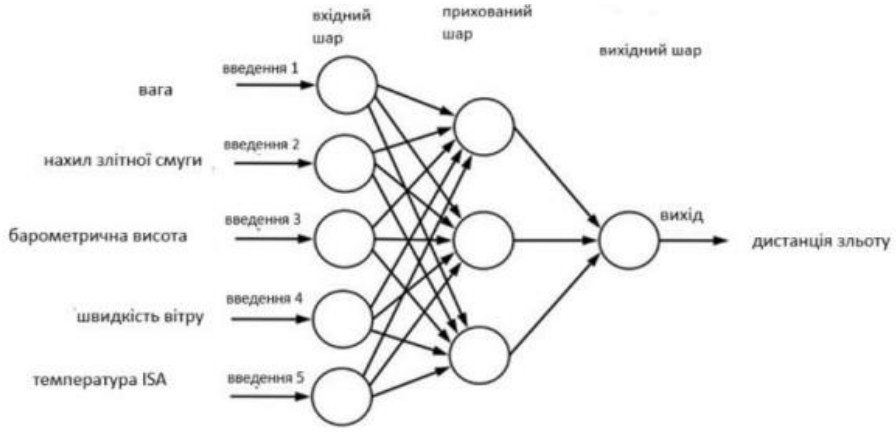


Рисунок 3.2. Вибрані вхідні та вихідні значення для нейронної мережі

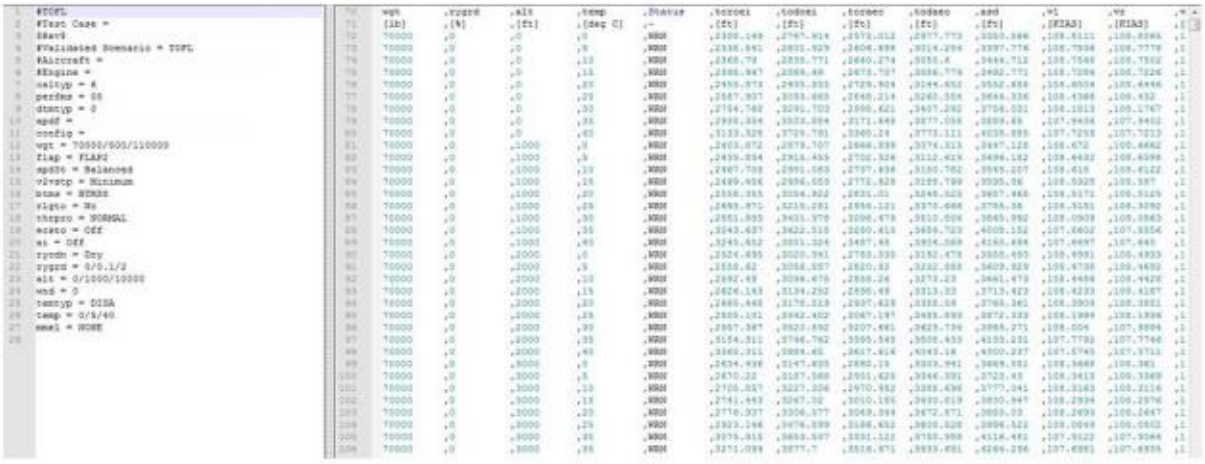


Рисунок 3.3. Зразок детермінованого набору даних

На рисунку 3.4 показано, як детермінований набір даних можна попередньо обробити та стиснути для оптимізації часу виконання та локального зберігання. Усі непотрібні символи рядка видаляються (тобто пробіли, повернення символів, повернення абзацу), і зберігаються лише п’ять вибраних вхідних параметрів і вихідне значення TOFL.

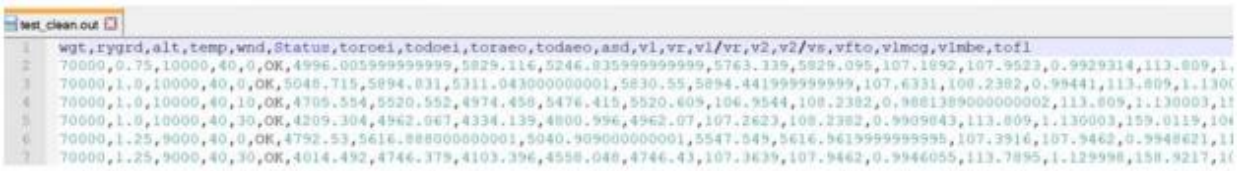


Рисунок 3.4. Стиснутий детермінований набір даних

На рисунку 3.5 показано розподіл кожного з параметрів із набору даних. Розподіл для всіх вхідних параметрів є лінійним, а розподіл для вихідного

параметра TOFL є поліноміальним. Вісь ординат показує кількість тестових прикладів, згенерованих на основі детермінованої моделі TOD, а вісь абсцис — значення кожного згенерованого тестового прикладу. Значення на осі у масштабуються для кращого порівняння. Для деяких випадків тестування 28 детерміністична модель TOD не може створити жодних значень, оскільки вони фізично неможливі. Наприклад, літак не може працювати в деяких тестових випадках із температурою DISA нижчою за -50 C, і це призводить до зменшення кількості створених тестових випадків.

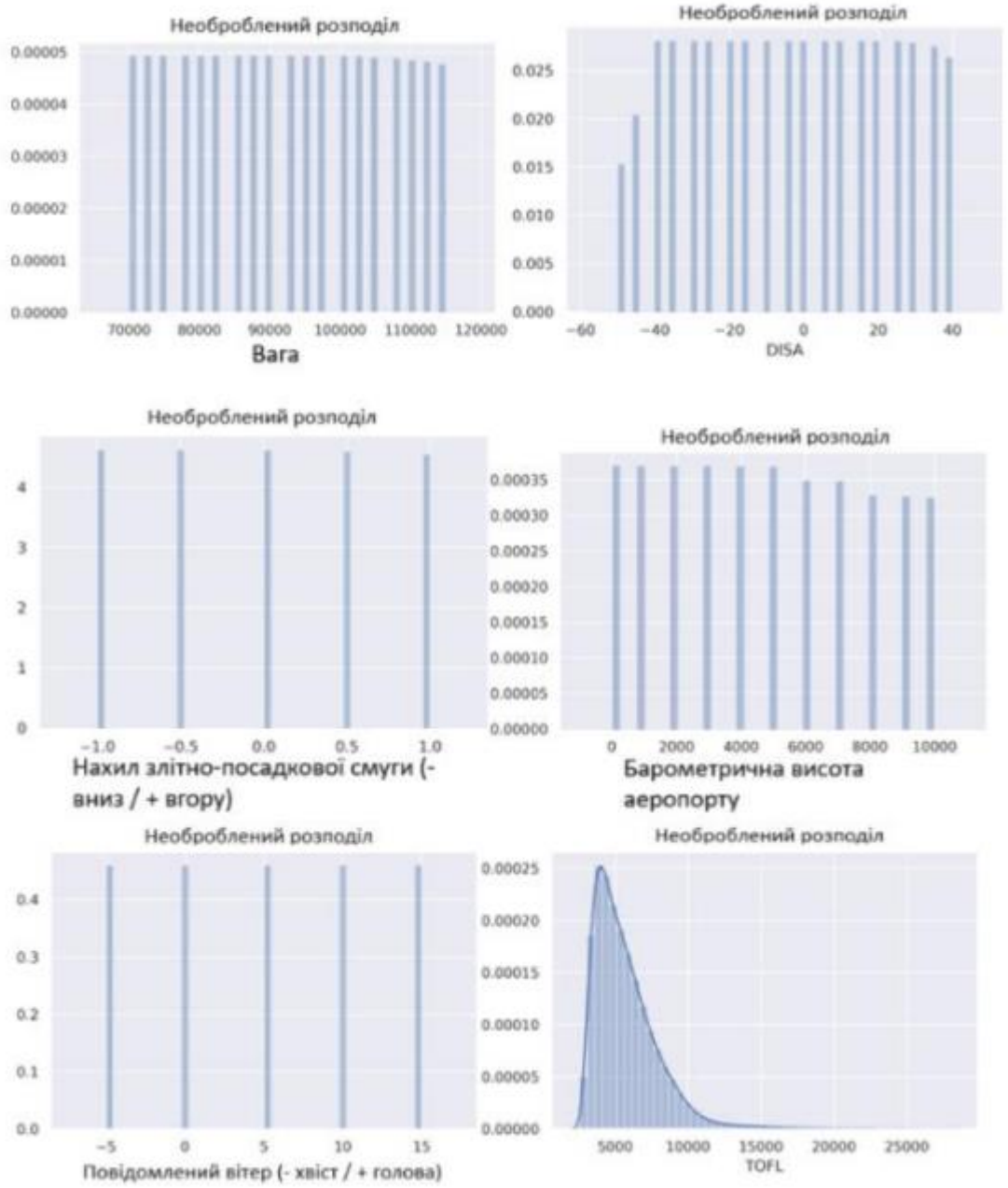


Рисунок 3.5. Розподіл набору даних для окремих вхідних і вихідних параметрів

Було протестовано три різні операції попередньої обробки:

1) необроблений розподіл без будь-яких змін;

2) нормалізований розподіл;

3) стандартизований розподіл.

Необроблений розподіл показує, що значення кожного параметра можуть сильно відрізнятися, що вплине на точність передбачення NN. Для вирішення цієї проблеми було створено нормалізований і стандартизований розподіл. Методом проб і помилок як найкращий варіант обрано нормалізацію.

3.3.2 Недетермінований набір даних NASA DASHlink

Набір даних NASA DASHlink складається із сукупних даних записів польотів, які складаються з фактичних даних, записаних на борту одного типу регіонального реактивного літака, який експлуатувався в комерційних цілях протягом трьох років (2001-2004). NASA заявляє наступне про дані про польоти: «Хоча файли містять детальну динаміку літака, продуктивність системи та інші інженерні параметри, вони не надають жодної інформації, яку можна відстежити до конкретної авіакомпанії чи виробника. […] [24]Відповідні сторони дозволили NASA надати дані широкому загалу з метою оцінки та вдосконалення можливостей аналізу даних, які можна використовувати для сприяння авіаційній безпеці». Дані польоту містять вичерпний перелік параметрів, які не потрібні для цього дослідження. Параметри, які використовуються для поточної роботи, можна знайти на рисунку 3.6 включають барометричну висоту, кількість палива, вагу літака, загальну температуру повітря, швидкість вітру, швидкість руху, висоту та середній час за Гринвічем. На рисунку також показано, як кожне значення датчика можна використовувати для розрахунку загальної ваги літака, пройденої відстані, нормалізованої швидкості вітру, часу, що минув, і дистанції зльоту; які є значеннями, необхідними для розрахунку TOD. Вибрані вхідні та вихідні параметри такі ж, як і для попереднього детермінованого набору даних, показаного на рисунку 3.6, за винятком нахилу злітно-посадкової смуги, який неможливо було обчислити через недостатню точність датчика.

Рисунок 3.6. Процес генерації вторинних даних з даних польоту

Після подальшого аналізу кожного вхідного параметра розподілу вихідного набору даних, як показано на рисунку 3.7, було виявлено чіткі зони розподілу даних, для яких недостатньо доступних точок даних для побудови мережевої мережі, здатної адекватно узагальнювати шаблони. Це спостереження було підтверджено шляхом емпіричних проб і помилок. Ці проблемні зони розподілу виділені червоним кольором на рисунку 3.7 і були вилучені з наборів даних, які потрібно перевірити в дослідженні[25]. Цей підхід до попередньої обробки даних базується на таких міркуваннях:

1. Деякі точки даних були фізично нереалістичними. Це пов’язано з помилками або неточностями датчиків. Приклади включають значення TOFL, нижчі за нуль, або вагу літака, яка більш ніж у десять разів перевищує інші значення, враховуючи, що всі значення мають бути віднесені до одного типу регіонального реактивного літака.
2. Цінності, які не пропорційно розподілені. Розподіл опору закрилків і нормований розподіл швидкості вітру демонструють непропорційні значення, оскільки майже всі точки даних зосереджені в дуже невеликій

кількості тестових випадків. Це значно ускладнює навчання NN, яка добре узагальнює всі точки даних.

3. Зони підвищеної ймовірності. Зони з найвищими розподілами, виділені зеленим кольором на рисунку 3.7, були пріоритетними для полегшення узагальнення шляхом зменшення кількості викидних точок даних.

Початковий набір даних було розділено на два окремі набори даних: один об'єднує дані для одного літака, а інший – для всього парку з дванадцяти літаків. Обговорений вище підхід попередньої обробки використовувався для зменшення набору даних із використанням обмежень, наведених у таблиці 7 і таблиці 8 відповідно. На рисунках 3.8 і 3.9 показано розподіли двох завершених наборів даних, використаних у цьому дослідженні.

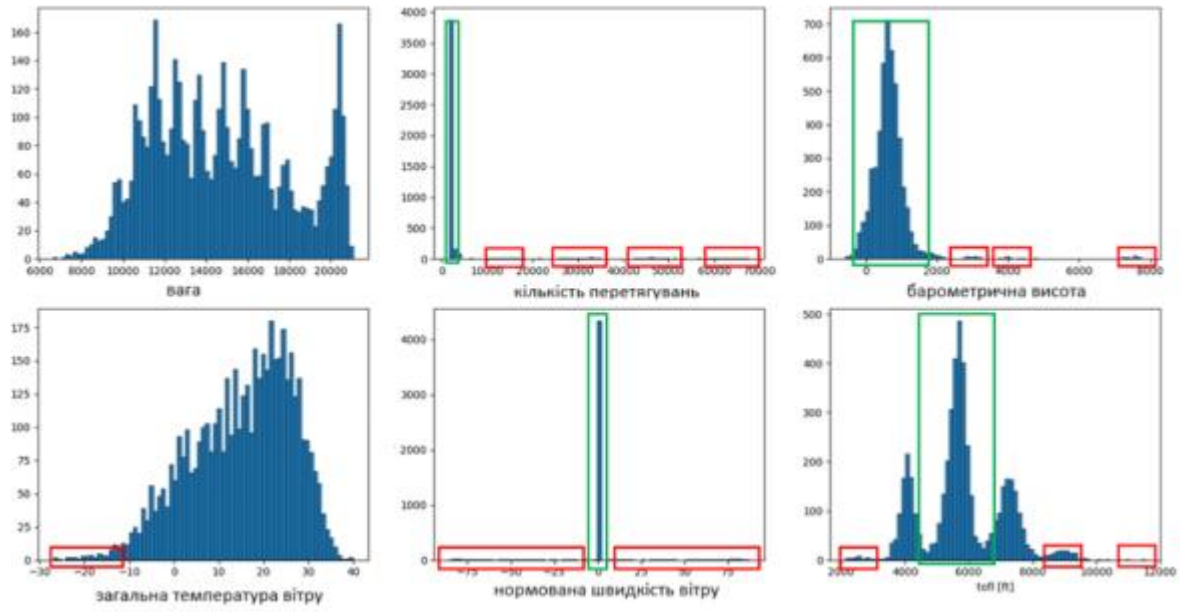


Рисунок 3.7. Розподіл вихідного набору даних для флоту з 12 літаків

Таблиця 7. Обмеження, накладені на вихідний набір даних для 1 літака

Параметр	Ліміти
Вага	> 0 & < 23000
Кількість перетягувань	> 2400 & < 2470
Барометрична висота [фути]	> -800 & < 2400
Загальна температура повітря [градус С]	> -10
Нормована швидкість вітру [фути/с]	= 0
TOFL	> 0 & < 10000

Таблиця 8. Обмеження, накладені на вихідний набір даних для флоту з 12 літаків

Параметр	Ліміти
Вага	> 0 & < 23000
Кількість перетягувань	> 1800 & < 2600
Барометрична висота [фути]	> -800 & < 2400
Загальна температура повітря [градус С]	Немає
Нормована швидкість вітру [фути/с]	= 0
TOFL	> 4700 & < 6500

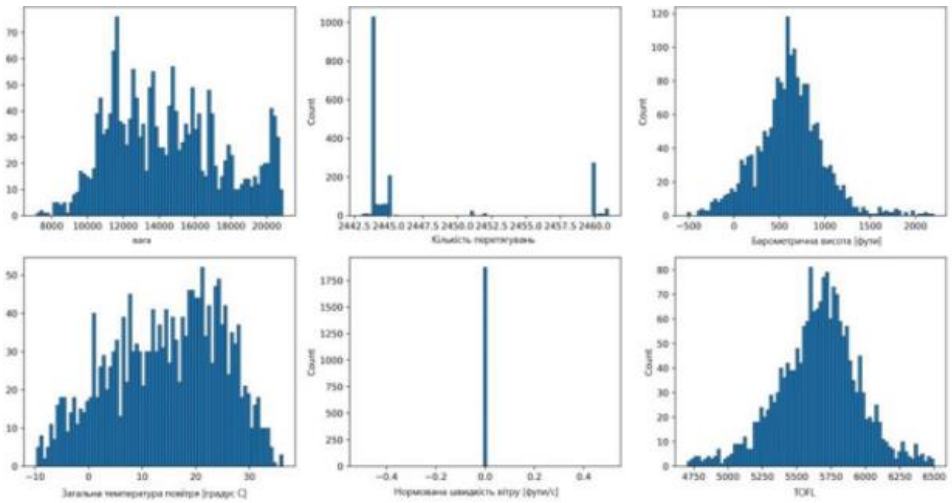


Рисунок 3.8. Розподіл вибраного набору даних для 1 літака

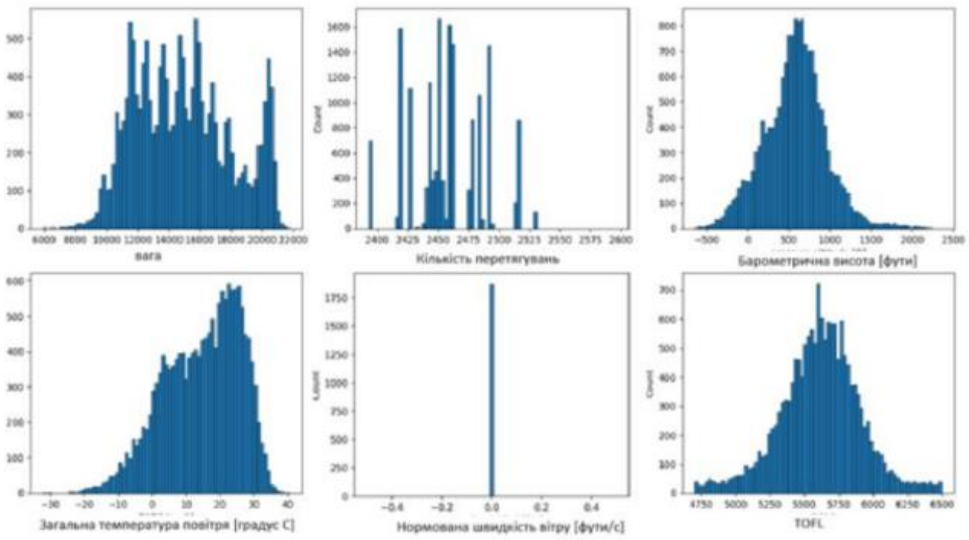


Рисунок 3.9. Розподіл вибраного набору даних для флоту з 12 літаків

3.4 Розробка архітектури нейронної мережі

Розробка архітектури мережі передбачає визначення компонентів архітектури, перелічених нижче та зображених на рисунку 3.10.

1. функція оптимізації;
2. функція втрат;
3. функція активації;
4. кількість прихованих шарів;
5. кількість нейронів на прихований шар;
6. максимально допустима кількість епох.

Рівень 1 є вхідним, і його кількість нейронів має дорівнювати кількості вхідних значень у наборі даних[26]. Останній шар, рівень 4, є вихідним шаром, і його кількість нейронів має дорівнювати числу вихідних значень у наборі даних. Рівні 2 і 3 називаються «прихованими шарами» і використовуються для паралельного поширення інформації до вихідного рішення. Чим більше прихованих шарів має мережа, тим «глибшою» вважається мережа. Термін «Глибоке навчання» зазвичай використовується для опису NN, яка має два або більше прихованих рівнів. Кожен нейрон складається з функції активації, яка вирішує, коли та чи використовується нейрон у обчисленні, функції оптимізації, яка використовується, щоб вирішити, як поширювати інформацію до наступного нейрона, і функції втрати, яка є функція, яку намагається оптимізувати функція оптимізації. Загальна мета цих компонентів полягає в тому, щоб знайти зважений скалярний добуток значення, приписуваного кожному нейрону, що дає значення, найближче до бажаного вихідного значення. Це робиться шляхом оновлення ваг і зміщень кожного мережевого підключення. Цей тип нейронної мережі із «прямим зворотним розповсюдженням» використовує алгоритм зворотного розповсюдження для оновлення вагових коефіцієнтів і зміщень, що дасть найкраще кінцеве рішення. Кількість разів, коли вагові коефіцієнти та зміщення оновлюються та зворотно поширюються мережею, називають «епохами», які можна розглядати як зворотну ітерацію. Вибір різних типів функцій і значень для

цих параметрів архітектури значно впливає на продуктивність мережі. Щоб звузити пошук, можна використовувати загальні правила з літератури:

1. Як загальне спостереження, чим глибша мережа, тим вища здатність мережі до узагальнення. Це пов'язано з підвищеною потребою в обчислювальній потужності, більшим часом роботи та не обов'язково завжди дає кращі результати у випадках, коли недостатньо даних для вилучення шаблонів або дані надто шумні.
2. Певні типи функцій оптимізації, втрати та активації більш ефективні для конкретних програм і наборів даних. Вони будуть пояснені більш детально далі в поточному розділі.
3. Кількість нейронів на прихований шар і максимально допустима кількість епох є функцією розміру набору даних.

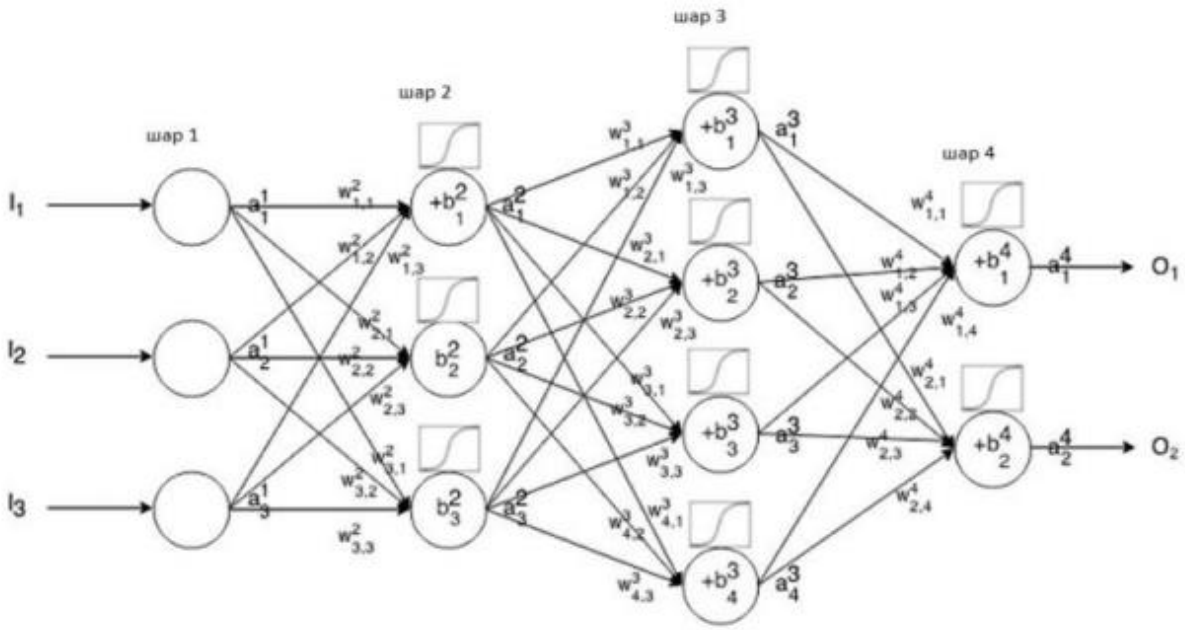


Рисунок 3.10. Процес розробки архітектури нейронної мережі

Для розробки нейронної мережі доступний ряд різних інструментів. NN можна програмувати безпосередньо на мовах програмування з низьким рівнем абстракції, як-от C, але пов'язане з цим робоче навантаження та знання програмування потрібні високі. Щоб вирішити ці проблеми, було створено низку різних програмних основ. Фреймворки програмування мають різні цілі та орієнтації, включаючи промисловий розвиток, академічні дослідження, швидке

створення прототипів і простоту моделювання впровадження. Фреймворки машинного навчання включають TensorFlow, Keras, PyTorch, Theano, Matlab і Caffee, серед інших. На рисунку 3.11 показано рейтинг найбільш часто використовуваних фреймворків ML на основі дослідження, проведеного Джеффом Хейлом, який розглядав рейтинг різних фреймворків глибокого навчання за різними категоріями. Keras API було обрано для поточного дослідження на основі наступного переваги:

1. Ефективність у зменшенні когнітивного навантаження (тобто послідовні та прості API, мінімізує кількість дій користувача, необхідних для звичайних випадків використання, і забезпечує чіткий і ефективний зворотний зв'язок у разі помилки користувача).
2. Корисно для швидкого створення прототипів і експериментів.
3. Простота впровадження в широкий спектр продуктів (iOS, android, браузер, Google Cloud, Raspberry Pi тощо).
4. Широке впровадження та легкість доступу до супровідної документації. Він повністю визнаний переднім кінцем TensorFlow, який можна використовувати в майбутніх дослідженнях для подальшої оптимізації розробленої мережі.

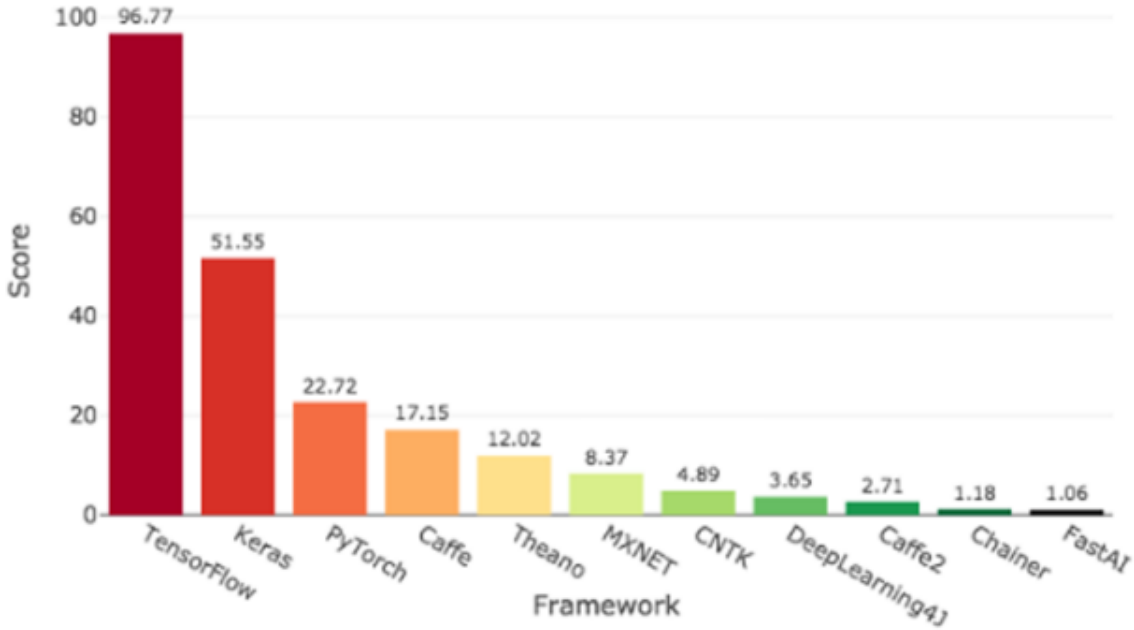


Рисунок 3.11. Рейтинг фреймворків глибокого навчання

На рисунку 3.12 показано класифікацію параметрів, які зазвичай використовуються в ML, які можна вибрати під час визначення архітектури мережі. Усі функції, представлені на малюнку, доступні в Keras, а також є можливість створювати власні функції.

Вибір відповідних функцій втрат поточна робота спрямована на вирішення задачі регресійної оптимізації, а не проблеми класифікації, яка звужує процес вибору архітектури. З функцій втрат, зображених на рисунку 3.12, функції logcsoh, MAPE, MAE, MSE і MSLE актуальні для проблеми регресії. У таблиці 9 наведено визначення для кожної функції втрат, а також їх найбільш використовувані застосування. Гровер пояснює, що медіана є більш стійкою до викидів, ніж середнє, що, отже, робить MAE більш стійкою до викидів, ніж MSE. Крім того, De Myttenaere et al. пояснюють, що пошук найкращої моделі за MAPE еквівалентний регресії зваженої середньої абсолютної помилки (MAE). З цих причин MSLE і MAPE використовуються для тестування NN.

Через міркування безпеки, пов’язані з аерокосмічними застосуваннями, прогнози, створені NN, повинні бути консервативними. Незважаючи на те, що бажано мати архітектуру NN, яка дає найнижчі значення MSLE або MAPE (які є мірою загальної похибки для всіх тестових випадків), слід зазначити, що керівна метрика для визначення найкращої моделі є найгіршою. Case error для всіх можливих тестових випадків.[27]

Рисунок 3.12. Зазвичай використовувані параметри архітектури NN

Таблиця 9. Визначення та застосування функції втрати

Визначення	Типове застосування
$mse = \frac{1}{n}\sum_{i=1}^{n}(actual_i - predicted_i)^2$	Добре підходить для наборів даних, які мають гауссове значення розподілу
$msle = \frac{1}{n}\sum_{i=1}^{n}[\log(actual_i) - \log(predicted_i)]^2$	Добре підходить для наборів даних із високою дисперсією. Він страждає від проблеми градієнта і Гессе для дуже великих нецільових прогнозів

$mae = \frac{1}{n}\sum_{i=1}^{n}(\lvert actual_i\rvert - \lvert predicted_i\rvert)$	Добре підходить для наборів даних, які мають переважно гауссовий розподіл, але містять викиди
$mape = \frac{100\%}{n}\sum_{i=1}^{n}\left\lvert\frac{actual_i - predicted_i}{actual_i}\right\rvert$	Те саме, що MAE, але як функція відносного відсоткова помилка

Вибір відповідних функцій оптимізації

Для задач регресії ML використовувані оптимізаційні функції, є варіантами алгоритмів градієнтного спуску, які називаються стохастичними градієнтними спусками (SGD).

Градієнтний спуск — це алгоритм оптимізації, який використовується для мінімізації деякої функції втрат шляхом ітераційного переміщення в напрямку найкрутішого спуску, визначеного негативним градієнтом. У ML градієнтний спуск використовується для оновлення вагових коефіцієнтів NN. Алгоритм SGD може зіткнутися з двома відомими проблемами:

1. Місцевий мінімум: SGD мають проблеми з регресійними кривими, які є дуже крутими. Це може призвести до локальних мінімальних проблем, які захоплюють алгоритм SGD у неправильному локальному оптимальному розв’язку. Momentum було розроблено для вирішення цієї проблеми шляхом прискорення рішення SGD у відповідному напрямку.
2. Вибір швидкості навчання. Вибір оптимальної швидкості навчання є складним, оскільки надто мала швидкість може призвести до дуже повільної збіжності, тоді як надто велика може перешкоджати збіжності та викликати коливання функції втрат навколо мінімуму або навіть до розходяться. Деякі варіанти SGD використовують адаптивні швидкості навчання, які оновлюються залежно від інших параметрів.

Таблиця 10 показує зведення відповідних функцій оптимізації SGD.

Функцію оптимізації Nadam (прискорена оцінка адаптивного моменту за Нестеровим) було обрано, оскільки вона забезпечує найбільше переваг у роботі з локальним мінімумом і для пошуку оптимальної швидкості навчання.

Таблиця 10 Функції оптимізації та їх застосування

Функція оптимізації	Типове застосування
Прискорений градієнт Нестерова (NAG)	Допомагає прискорити SGD з проблемного локального мінімуму рішення.
Adagrad	Адаптує швидкість навчання до частоти появи параметри. Він добре підходить для роботи з розрідженими даними.
Adadelta and RMSprop	Розширення Adagrad, монотонно знижена швидкість навчання.
Адам (Оцінка адаптивного моменту) [71]	Адаптує швидкість навчання шляхом збереження експоненціального значення спадне середнє минулих квадратичних градієнтів (наприклад, Adadelta і RMSprop), також зберігаючи експоненціальний спадне середнє минулих градієнтів (як імпульс).
AdaMax	Варіант Адама, корисний для роботи з нестабільними норми навчання минулого градієнта. Практичний при веденні справ з більшими нормальними значеннями для минулих градієнтів.
Nadam (прискорена адаптація Нестерова Оцінка моменту)	Поєднує в собі можливості Адама і NAG

Вибір відповідних функцій активації

Вибір відповідної функції активації залежить від застосування NN. Деякі розроблені для надання двійкових вхідних даних (0 або 1), інші для надання значень від 0 до 1, а треті для надання точних значення. Гудфеллоу та ін. рекомендують використовувати функцію активації ReLU (Rectified Linear Units) для вхідного та прихованого шарів та лінійні або сигмоїдні функції для вихідного шару. Якщо функція ReLU не дає прийнятних результатів, можна емпірично протестувати похідні від ReLU (eLU, SeLU) або Softmax, Softplus і Softsign.

Для поточного дослідження максимальна кількість дозволених епох не вважається критичним параметром для визначення архітектури. Гудфеллоу та ін. пояснюють, що часто корисніше визначити «критерій ранньої зупинки», який, коли його спрацьовує, зупинить навчання незалежно від кількості досягнутих епох.

Використовується такий критерій: «якщо останні десять епох не призвели до покращення середньої відсоткової помилки, навчання буде припинено». Цей критерій має додаткову перевагу в економії часу обчислення, оскільки NN не буде навчатися для подальших епох, які в будь-якому випадку не дадуть кращих результатів.

Таблиця 11 надає матрицю різних комбінацій архітектур NN, які були перевірені з використанням різних наборів даних.

Таблиця 11. Вибрані параметри архітектури для тестування за допомогою наборів даних

Оптимізація функція	Втрата функція	активація функція	Вузли/вхідний рівень	Приховані шари	Вузли/прихований шар
Adadelta	MAPE	ReLU	1000	1	10
Adamax	MSLE		3000	2	100
Nadam			5000	3	1000

3.5 Навчання та тестування нейронної мережі

Навчання NN передбачає використання набору даних для оновлення вагових коефіцієнтів NN, щоб знайти значення, найближчі до вихідного рішення, тоді як тестування NN використовує навчальні вагові коефіцієнти для прогнозування з новим набором даних для перевірки результатів. Тестування також є хорошим методом перевірки для експериментів у випадках, які не були охоплені в наборі навчальних даних. Навчання та тестування NN можна проводити за допомогою одного набору даних і розділити його на дві частини. У більшості задач регресії використовується розподіл перевірки 70 % для навчання та 30 % для тестування. Це співвідношення буде використано для всіх результатів, представлених у цій

дипломній роботі. Програма Keras має можливість генерувати графіки, які можна використовувати для оцінки продуктивності під час навчання та тестування, як показано на рисунку 3.13. Для кожної протестованої архітектури в цьому дослідженні ці графіки використовувалися для оцінки продуктивності NN. На осі Y відкладено вибрану метрику продуктивності, а на осі X – кількість епох і, таким чином, є мірою часу навчання. Наприклад, на рисунку 3.13 вибраний показник MAPE і порівнюється для значень навчання та тестування. Графік показує:

1) наскільки швидко метрика може наблизитися до значення в стаціонарному стані;

2) точність значення в стаціонарному стані.

Відомою проблемою, яка може виникнути під час порівняння результатів навчання та тестування, є надмірне або недостатнє оснащення. Недостатньо підігнані значення не можуть виділити базовий шаблон, знайдений у наборі даних, у той час як надмірно пристосовані значення можуть виділити цей шаблон надто добре, як показано на рисунку 3.14. Недооблаштовані значення, як правило, не можуть збігатися з прийнятним рішенням, тоді як перепідібрані значення не дозволяють пристосуватися до проміжків між точки даних. Це може призвести до низької продуктивності моделі під час тестування значень із іншого набору даних, ніж під час навчання. Ці проблеми можна пом’якшити, використовуючи коефіцієнт перевірки, близький до 30/70, і надаючи достатньо даних.

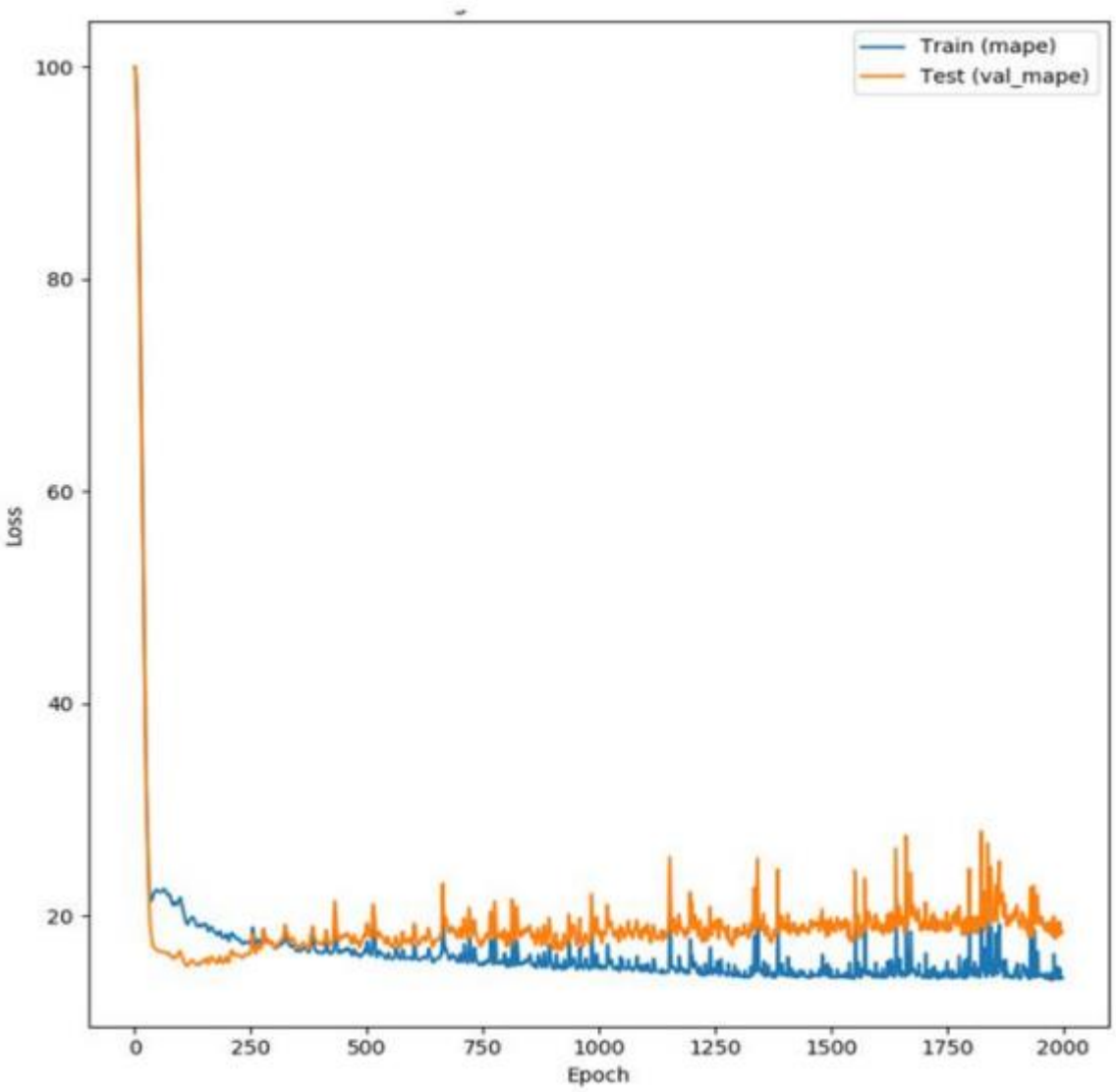


Рисунок 3.13. Навчання та тестування MAPE проти епохи

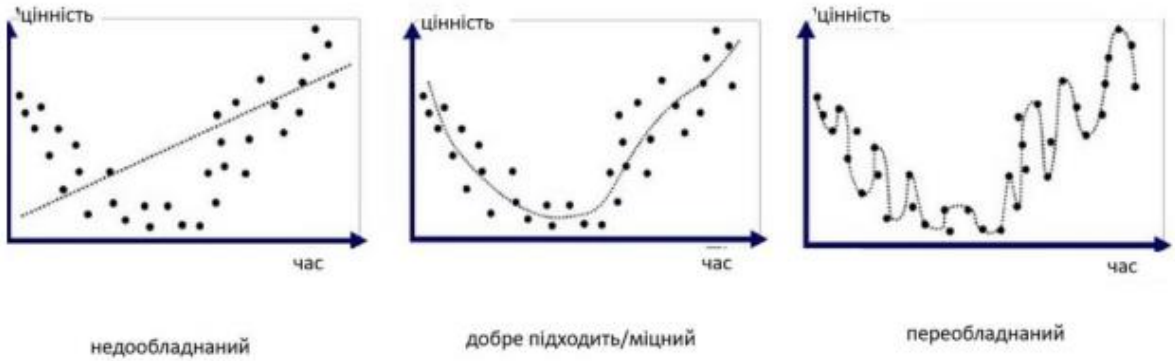


Рисунок 3.14. Порівняння недостатнього та надмірного оснащення [72]

3.6 Використання навченої мережі для прогнозування

Після навчання моделі NN створюється файл, який містить лише навчені ваги мережі. Можна розробити просту програму інструменту прогнозування, щоб отримати вихід, подібний до того, що показано на рисунку 3.15. Користувач

передає вхідні дані інструменту, для яких він бажає, щоб інструмент передбачав TOD, і інструмент прогнозує TOD і порівнює значення NN із фактичним значенням з набору даних, на якому він навчався. Час роботи інструменту прогнозування дуже швидкий, оскільки він шукає лише ваги та обчислює один вихід на основі набору заданих вхідних параметрів. Формат навчених вагових коефіцієнтів NN також полегшує інтеграцію з існуючими інструментами (FMS, EFB, програмне забезпечення AFM тощо).

```
Weight [lb] = 93000
Runway Slope (-Down/+Up) [%] = 1.25
Airport Pressure Altitude [ft] = 5000
DISA [deg C] = -20
Reported Wind (-Tail/+Head) [kts] = 15
     Takeoff Field Length [ft] = 4889.172

data=
        wgt  rygrd     alt  temp   wnd      tofl
0  93000.0   1.25  5000.0 -20.0  15.0  4889.172

Actual value          = 4889.1720
Predicted by neural net = 4889.7251
Percentage error = 0.0113%
Mean difference [ft] = 0.5531
```

Рисунок 3.15. Вихід інструменту, що використовує навчену NN

На рисунку 3.16 зображено файлову структуру програм на Python, які використовувалися в цьому дослідженні та описувалися в цьому розділі. Набір даних аналізується за допомогою файлу dataset_properties.py. Це робиться для полегшення етапу попередньої обробки даних при роботі з дуже великими наборами даних. Більшість наборів даних не будуть у належному форматі або потребуватимуть попередньої обробки, перш ніж їх можна буде використовувати для розробки мережевих мереж шляхом видалення точок даних або зміни розподілу набору даних. Це робиться за допомогою файлу “clean_out_file.py”. Далі файл “neural_network_architecture_selection.py” використовується для навчання та тестування мережі та оцінки її продуктивності. Цей код полегшує експерименти з вибору оптимальної архітектури NN для кожного набору даних. Кожна навчена мережа зберігається у файлах типів “saved_net.hdf5”, які дуже

ефективні для пам’яті, і потім можуть використовуватися “neural_network_tool.py” для прогнозування на основі введення користувача.

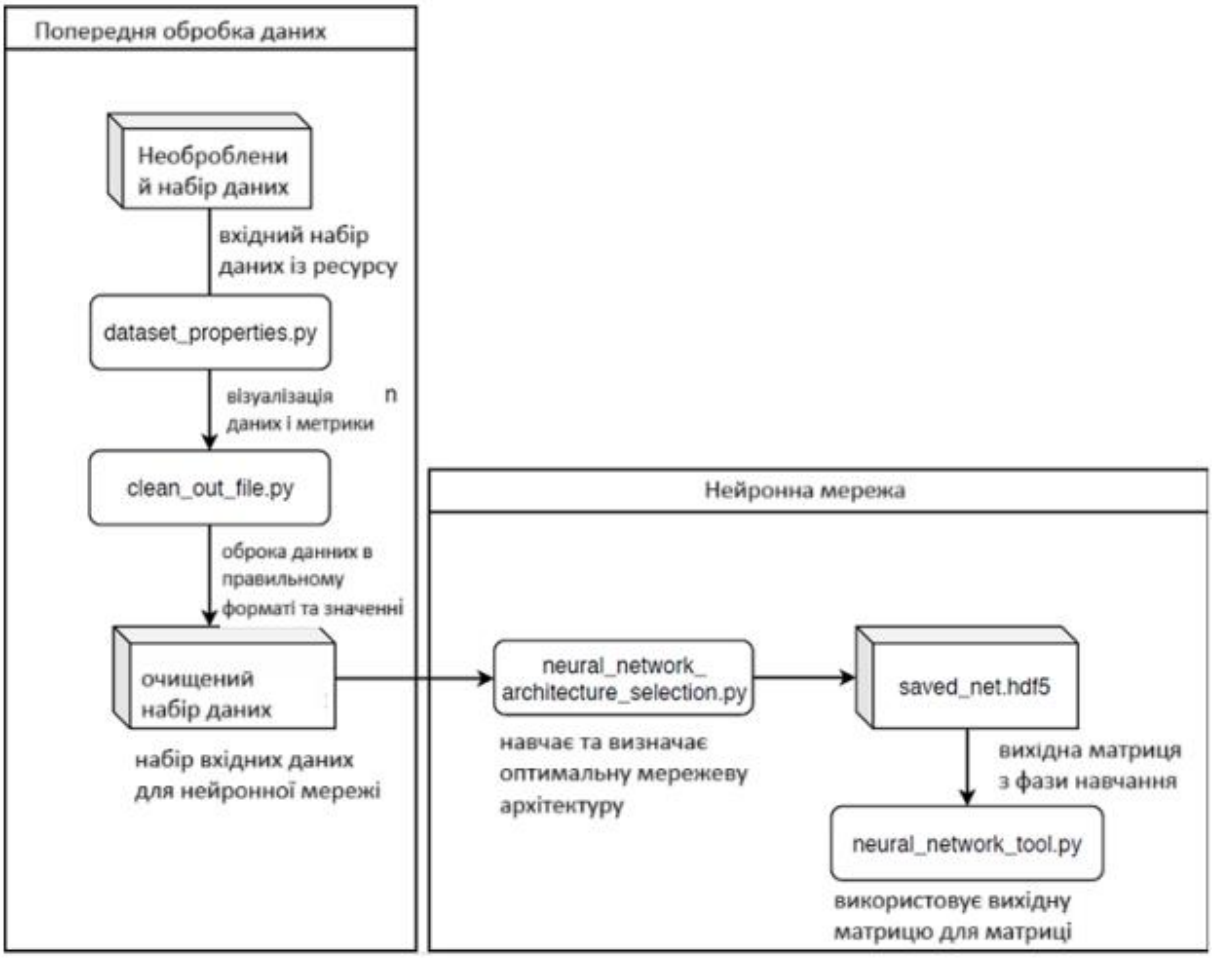


Рисунок 3.16. Схема структури коду

РОЗДІЛ 4. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ В СЕРЕДОВИЩІ PYTORCH

4.1. Створення і завантаження наборів даних

Для розробки нейронної мережі, яка може передбачити дистанцію зльоту літака, ключовим кроком є підготовка та завантаження наборів даних. Якість моделі багато в чому залежить від якості даних, на яких вона навчається. У цьому розділі розглядаються основні етапи створення, обробки та завантаження навчальних даних[28].

1. Джерела даних

Для дослідження використовуються наступні джерела даних:

Детермінований набір даних дистанції зльоту: складається з чітко визначених параметрів, таких як маса літального апарата, довжина злітної смуги, швидкість вітру, температура повітря, висота над рівнем моря тощо.

Недетермінований набір даних NASA DASHlink: включає реальні експериментальні дані про польоти, які можуть містити певні похибки чи відсутні значення.

Обидва типи даних є важливими для створення універсальної моделі, здатної працювати в різних умовах.

2. Формат даних

Дані зазвичай представлені у форматах CSV (Comma-Separated Values) або JSON, які зручно обробляти за допомогою бібліотек Python. Кожен запис у наборі даних містить вхідні параметри (features) та цільову змінну (target), тобто дистанцію зльоту.

Приклад структури даних у форматі CSV наведені у таблиці 12:

Таблиця 12. Приклад структури даних у форматі CSV

Маса (кг)	Швидкість вітру (м/с)	Температура (°C)	Висота (м)	Дистанція зльоту (м)
2000	5	20	300	1200
1500	3	15	150	1000

3. Завантаження даних у PyTorch

У середовищі PyTorch для роботи з даними використовуються спеціальні класи Dataset та DataLoader, які спрощують завантаження, передобробку та ітерацію над даними.

3.1. Завантаження даних з файлу

Для завантаження даних із CSV-файлу використовується бібліотека pandas.

```
import pandas as pd

# Завантаження даних із файлу
data = pd.read_csv('flight_data.csv')

# Перевірка структури даних
print(data.head())
```

3.2. Створення власного класу Dataset

PyTorch дозволяє створювати власні набори даних для гнучкого налаштування передобробки.

```
import torch
from torch.utils.data import Dataset

class FlightDataset(Dataset):
    def __init__(self, data):
        self.data = data
        self.features = self.data.iloc[:, :-1].values  # Вхідні параметри
        self.target = self.data.iloc[:, -1].values     # Цільова змінна

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        x = torch.tensor(self.features[idx], dtype=torch.float32)
        y = torch.tensor(self.target[idx], dtype=torch.float32)
        return x, y

# Ініціалізація Dataset
dataset = FlightDataset(data)
```

3.3. Використання DataLoader

DataLoader дозволяє ітерувати дані партіями (batches), що прискорює навчання моделі.

```
from torch.utils.data import DataLoader

# Створення DataLoader
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)

# Приклад ітерації
for batch in dataloader:
    features, target = batch
    print(features.shape, target.shape)
    break
```

4. Передобробка даних

Дані мають бути підготовлені перед подачею до моделі. Основні етапи передобробки:

- Нормалізація даних

Для забезпечення стабільності навчання всі числові параметри масштабуються до діапазону [0, 1] або [-1, 1].

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
data.iloc[:, :-1] = scaler.fit_transform(data.iloc[:, :-1])
```

- Обробка відсутніх значень

Заповнення відсутніх даних середнім значенням або використання алгоритмів інтерполяції:

```
data.fillna(data.mean(), inplace=True)
```

- Кодування категоріальних змінних[29]

Якщо дані містять категоріальні змінні (наприклад, тип літака), вони перетворюються у числовий формат за допомогою One-Hot Encoding.

5. Розподіл на навчальну, валідаційну та тестову вибірки

Дані поділяються на три частини:

- Навчальна вибірка (Train Set): використовується для навчання моделі (80%).
- Валідаційна вибірка (Validation Set): використовується для перевірки під час навчання (10%).
- Тестова вибірка (Test Set): використовується для оцінки точності моделі після навчання (10%).

Розподіл здійснюється за допомогою бібліотеки sklearn:

```
from sklearn.model_selection import train_test_split

train_data, test_data = train_test_split(data, test_size=0.2, r
val_data, test_data = train_test_split(test_data, test_size=0.5
```

Таким чином, створення і завантаження наборів даних — це критично важливий етап у процесі розробки нейромережі. Використання PyTorch для роботи з даними забезпечує ефективне управління навчальними вибірками, їх підготовкою та оптимізацією для подальшого використання у моделі.

4.2. Побудова моделі, компонентів та шарів нейромережі

Процес побудови нейромережі є основним етапом у розробці системи прогнозування дистанції зльоту літального апарату. У цьому розділі розглядаються принципи створення моделі архітектури, компоненти нейронної мережі та особливості налаштування шарів із використанням бібліотеки PyTorch.

1. Вибір типу нейронної мережі

Для задачі прогнозування (регресії) найбільше підходить багатошарова перцептронна мережа (Multilayer Perceptron, MLP) . Основні причини вибору:

- Вхідні дані представлені в табличному форматі.
- Мережа здатна навчатися на числових даних і виконувати регресійний аналіз.

Архітектура мережі створює кілька повнозв'язаних шарів із функціями активації для побудови нелінійних залежностей.

2. Компоненти нейромережі

2.1. Вхідний шар

- Приймає на вхід кілька характеристик літального апарату (наприклад, маса, швидкість вітру, температура повітря).
- Кількість нейронів у цій шарі дорівнює кількість ознак у вхідних даних (наприклад, 5 параметрів — 5 нейронів).

2.2. Приховані шари

- Мають зміну кількості нейронів залежно від складності задачі.
- Використовуються нелінійні функції активації, такі як ReLU (Rectified Linear Unit), які добре працюють для регресійних завдань.

2.3. Вихідний шар

- Складається з одного нейрона, який прогнозує цільове значення (дистанцію зльоту).
- У регресійних задачах функція активації на вихідному шарі також не використовується.

3. Процес побудови моделі в PyTorch

3.1. Імпортування більшості бібліотек

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

3.2. Створення класу нейромережі

У PyTorch модель знижується як клас, що успадковується від torch.nn.Module.

```python
class FlightModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(FlightModel, self).__init__()

        # Вхідний шар
        self.input_layer = nn.Linear(input_size, hidden_size)

        # Прихований шар
        self.hidden_layer = nn.Linear(hidden_size, hidden_size)

        # Вихідний шар
        self.output_layer = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        # Проходження через вхідний шар з активацією ReLU
        x = F.relu(self.input_layer(x))

        # Проходження через прихований шар з активацією ReLU
        x = F.relu(self.hidden_layer(x))

        # Вихідний шар (без активації для регресії)
        x = self.output_layer(x)

        return x
```

3.3. Ініціалізація моделі

```python
# Визначення розмірів шарів
input_size = 5      # Кількість вхідних ознак
hidden_size = 64    # Кількість нейронів у прихованому шарі
output_size = 1     # Вихідна змінна (дистанція зльоту)

# Створення моделі
model = FlightModel(input_size, hidden_size, output_size)

# Виведення архітектури моделі
print(model)
```

Приклад виведення:

```
FlightModel(
  (input_layer): Linear(in_features=5, out_features=64, bias=True)
  (hidden_layer): Linear(in_features=64, out_features=64, bias=True)
  (output_layer): Linear(in_features=64, out_features=1, bias=True)
)
```

4. Гіперпараметри моделі

4.1. Функції активації

Функція активації налаштовується, як нейрони активуються та передають інформацію далі:

- ReLU: використання для прихованих шарів через свою ефективність в унікальних проблемах зняття градієнтів.
- Без активації на вихідному шарі: проблема регресії потребує випадкового прогнозування чисельних значень.

4.2. Початкові значення ваг

Ваги ініціалізуються автоматично, але за потреби можна застосувати власні методи ініціалізації.

```
def init_weights(m):
    if isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)
        nn.init.zeros_(m.bias)

model.apply(init_weights)
```

4.3. Гіперпараметри навчання

- Кількість шарів: зазвичай 2–3 прихованих шари є достатніми для регресії завдань.
- Кількість нейронів у шарі: оптимальний параметр — 32–128 нейронів залежно від складності даних.
- Швидкість навчання: починається з 0,001.

5. Додавання регулярізації

Для запобігання перенавчанню додається шар Dropout, що випадково оновлює певний відсоток нейронів під час навчання.

```
self.dropout = nn.Dropout(p=0.5)  # 50% нейронів вимикаються
x = self.dropout(x)
```

Нормалізація

Batch Normalization стабілізувати та прискорити навчання.

```
self.batch_norm = nn.BatchNorm1d(hidden_size)
x = self.batch_norm(x)
```

6. Застосування моделі до даних

Щоб перевірити, чи правильно модель обробляє вхідні дані, можна передати тестовий вектор:

```
# Тестовий приклад
example_data = torch.rand(1, input_size)  # Один зразок із 5 ознак
output = model(example_data)
print(output)
```

Побудова моделі нейромережі включає визначення структури, налаштування компонентів (вхідних, прихованих і вихідного шарів) та врахування гіперпараметрів. PyTorch надає інструментарій для створення моделей, які можуть бути адаптовані до конкретних завдань, таких як прогнозування дистанції зльоту літальних апаратів.

4.3. Автоматичне диференціювання

Автоматичне диференціювання — це ключовий етап у процесі навчання нейронних мереж, який забезпечує ефективне обчислення похідних (градієнтів) функції втрат відносно параметрів моделі. Завдяки градієнтам здійснюється корекція параметрів під час оптимізації. У середовищі PyTorch автоматичне диференціювання реалізується за допомогою модуля autograd, який дозволяє обчислювати похідні для складних функцій без ручного розрахунку.

Основні принципи автоматичного диференціювання

1. Граф обчислень (комп’ютерне представлення функцій):
    - PyTorch автоматично будує динамічний граф обчислень, де кожна операція представлена як вузол.
    - Вхідні тензори та виконувані над ними операції зберігаються у графі, який дозволяє ефективно обчислювати градієнти у зворотному проході (backward pass).
2. Ланцюгове правило:

- Для обчислення градієнтів використовується правило ланцюга, що дозволяє розбивати складні похідні на послідовність простих.

3. Forward і backward passes:
   - Під час прямого проходу (forward pass) обчислюється значення функції.
   - У зворотному проході (backward pass) обчислюються похідні.

Реалізація автоматичного диференціювання в PyTorch

Модуль autograd автоматично обчислює градієнти для тензорів, які мають параметр requires_grad=True.

Приклад обчислення градієнтів простої функції

Розглянемо функцію:

$$y=x^3+2x^2+5xy$$

Обчислимо похідну dy/dx для значення x=2.

```python
import torch

# Оголошення тензора із можливістю обчислення градієнтів
x = torch.tensor(2.0, requires_grad=True)

# Обчислення функції
y = x**3 + 2*x**2 + 5*x

# Обчислення градієнта
y.backward()

# Виведення градієнта
print(f"Градієнт функції y по x: {x.grad}")
```

Результат:

Градієнт функції y по x буде:

$$Dy/dx=3x^2+4x+5$$

При x=2, градієнт дорівнює 25.

Використання автоматичного диференціювання в нейронній мережі

У процесі навчання нейронної мережі автоматичне диференціювання використовується для оновлення параметрів (ваг і зміщень) на основі функції

втрат. PyTorch обчислює градієнти для всіх параметрів моделі за допомогою одного виклику методу .backward().

```python
import torch
import torch.nn as nn

# Створення простої моделі
model = nn.Linear(1, 1)  # Лінійний шар: y = Wx + b

# Вхідні дані та цільові значення
x = torch.tensor([[2.0], [3.0]], requires_grad=True)
y_true = torch.tensor([[4.0], [6.0]])

# Функція втрат
criterion = nn.MSELoss()

# Прямий прохід (forward pass)
y_pred = model(x)

# Обчислення функції втрат
loss = criterion(y_pred, y_true)

# Зворотний прохід (backward pass)
loss.backward()

# Виведення градієнтів
print("Градієнт ваг:", model.weight.grad)
print("Градієнт зміщення:", model.bias.grad)
```

- Прямий прохід обчислює прогнозовані значення y_pred.
- Функція втрат criterion порівнює прогнозовані значення з реальними y_true.
- Зворотний прохід викликається методом loss.backward(), що обчислює градієнти функції втрат для параметрів моделі (weight, bias).

PyTorch підтримує обчислення похідних вищого порядку через параметр create_graph=True.

Приклад[30]:

Обчислимо першу та другу похідні для функції $y=x^3$.

```python
x = torch.tensor(2.0, requires_grad=True)
y = x**3

# Перша похідна
dy_dx = torch.autograd.grad(y, x, create_graph=True)[0]

# Друга похідна
d2y_dx2 = torch.autograd.grad(dy_dx, x)[0]

print(f"Перша похідна: {dy_dx}")
print(f"Друга похідна: {d2y_dx2}")
```

Результат:

- Перша похідна: $3x^2=12$.
- Друга похідна: $6x=12$.

Обмеження автоматичного диференціювання

1. Ресурсозатратність:

   Обчислення градієнтів може бути обчислювально дорогим для великих моделей.

2. Необхідність обнулення градієнтів:

Перед виконанням нового зворотного проходу необхідно обнулювати градієнти, використовуючи:

```python
optimizer.zero_grad()
```

Відключення градієнтів:

У випадках, коли обчислення градієнтів не потрібне (наприклад, під час інференсу), їх слід відключити:

```python
with torch.no_grad():
    y_pred = model(x)
```

Автоматичне диференціювання в PyTorch є потужним інструментом, який спрощує процес навчання нейронних мереж, забезпечуючи автоматичне обчислення градієнтів для всіх параметрів моделі. Завдяки модулю autograd, розробники можуть ефективно реалізувати складні алгоритми оптимізації, не витрачаючи час на ручне розрахування похідних.

4.4. Оптимізація параметрів моделі

Оптимізація параметрів моделі є ключовим етапом навчання нейронних мереж, який спрямований на мінімізацію функцій шляхом оновлення параметрів

(ваг та зміщень) моделі. У PyTorch цей процес реалізується за допомогою різних алгоритмів оптимізації, які вибирають градієнти, обчислені за допомогою автоматичного диференціювання.

Оптимізація базується на методах градієнтного випуску, які оновлюють параметри моделі у напрямку, що зменшує значення функції втрат.

Процес оновлення параметрів виконується за формулою:

$$\theta_{t+1}=\theta_{t-\eta}\cdot\nabla L(\theta_t)$$

де:

- θt— параметри моделі на ітераціїttt;
- η— коефіцієнт навчання (raterning rate);
- $\nabla L(\theta_t)$— градієнт функції втрат L щодо параметрівθt.

PyTorch надає декілька популярних алгоритмів оптимізації через модуль torch.optim, серед яких:

- SGD (Stochastic Gradient Descent): класичний метод стохастичного градієнтного спуску.
- Адам (Adaptive Moment Estimation): вдосконалений метод, що адаптує швидкість навчання для кожного параметра.
- RMSprop, Adagrad, Adadelta: методи, які задають адаптивні коефіцієнти навчання.

Для налаштування оптимізатора необхідно:

1. Вказати параметри моделі.
2. Вибрати алгоритм оптимізації.
3. Задати гіперпараметри (наприклад, коефіцієнт навчання).

```
import torch.optim as optim

# Припустимо, що model — це нейронна мережа
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Процес оптимізації

1. Прямий прохід:

   Обчислюється прогноз моделі на основі вхідних даних.

2. Обчислення функцій втрат:

   Використовується функція втрат для оцінки помилок між прогнозом і реальними значеннями.

3. Зворотний прохід:

   Методом backward() обчислення градієнтів параметрів.

4. Оновлення параметрів:

   Оптимізатор оновлює значення параметрів на основі обчислених градієнтів.

5. Обновлення градієнтів:

   Перед наступною ітерацією потрібно змінити значення градієнтів, після чого PyTorch додає їх до попередніх значень.

```python
import torch
import torch.nn as nn
import torch.optim as optim

# Створення простої моделі
model = nn.Linear(1, 1)

# Функція втрат
criterion = nn.MSELoss()

# Оптимізатор
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Приклад даних
x = torch.tensor([[1.0], [2.0], [3.0]])
y_true = torch.tensor([[2.0], [4.0], [6.0]])

# Навчання моделі
for epoch in range(100):  # Кількість епох
    # Прямий прохід
    y_pred = model(x)

    # Обчислення функції втрат
    loss = criterion(y_pred, y_true)

    # Обнулення градієнтів
    optimizer.zero_grad()

    # Зворотний прохід
    loss.backward()

    # Оновлення параметрів
    optimizer.step()

    if epoch % 10 == 0:
        print(f"Епоха {epoch}, Втрата: {loss.item()}")
```

Коефіцієнт навчання (η) є гіперпараметром, який впливає на швидкість і стабільність оптимізації:

- Занадто великий η може призвести до нестабільності та розбіжності.
- Занадто малий η уповнює процес навчання.

Динамічне налаштування швидкості навчання

PyTorch дозволяє адаптувати коефіцієнти навчання під час навчання за допомогою планувальників (Schedulers), наприклад:

```python
from torch.optim.lr_scheduler import StepLR

# Планувальник зменшує lr кожні 10 епох у 2 рази
scheduler = StepLR(optimizer, step_size=10, gamma=0.5)

for epoch in range(100):
    # Звичайний процес навчання
    y_pred = model(x)
    loss = criterion(y_pred, y_true)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Оновлення коефіцієнта навчання
    scheduler.step()
```

Особливості оптимізації нейронних мереж

1. Розв'язання проблеми перенавчання:

   Для уникнення перенавчання можна використовувати методи регулярізації, наприклад:

   - Вибуття.
   - L1 або L2-регуляризація.

2. Мініпакети (міні-пакети):

   Для оптимізації великих наборів даних виконуються на підмножинах даних (мініпакетах), що зменшує обчислювальну складність.

3. Зупинка навчання (раннє припинення):

   Навчання може бути припинено, якщо функція втрати на валідаційному наборі перестає зменшуватися.

Оптимізація параметрів є центральним компонентом навчання нейронних мереж. У середовищі PyTorch за допомогою потужних інструментів, таким як модуль optim, можна ефективно реалізовувати різні методи оптимізації, адаптувати коефіцієнти навчання та інтегрувати сучасні алгоритми для досягнення найкращих результатів.

ВИСНОВКИ

Мета цього дослідження полягала в тому, щоб дослідити, чи можна штучні нейронні мережі ефективно використовувати як альтернативу поточним моделям продуктивності літаків. На основі результатів, отриманих із детерміністичного набору даних, створеного за допомогою існуючої моделі TOD, можна стверджувати, що результати NN відповідали існуючій моделі. Для оптимальної моделі NN ці результати збігалися з існуючими значеннями моделі TOD в межах 0,61 % або нижче та мали загальну середню похибку моделі 0,05 %. Навчання нейронних мереж з використанням даних із існуючої детермінованої моделі виявилося дуже ефективним, головним чином завдяки його здатності генерувати різні розподіли та розміри наборів даних. Для результатів NN, заснованих на недетермінованих польотних даних, також було виявлено, що NN може давати прийнятні прогнози. Для оптимальної моделі NN, створеної з використанням даних польоту, середня помилка для всіх прогнозованих тестових випадків продемонструвала ефективність 3,94 %, але мережа була найбільше обмежена тестовими випадками, що виходять із діапазону помилок 18,59 %. Було розглянуто причини для пояснення неприйнятних результатів, і вважається, що за допомогою подібного набору даних зі збільшеним розміром і більш детальною інформацією про кожне значення даних потенційно можна отримати прийнятну продуктивність. Майбутні дослідження могли б дослідити більш глибоке обґрунтування того, чому ці конкретні тестові випадки викидів дають вищі значення помилок.

У роботі розглянуто основні аспекти розвитку та застосування нейромереж, їхні програмні та апаратні реалізації, а також особливості використання в технічних системах і літальних апаратах. Описано історичні етапи розвитку нейромереж, визначено сучасні тенденції їх впровадження у різних сферах.

Досліджено процеси маршрутизації малих літальних апаратів із використанням сучасних технологій, таких як Google Maps, а також визначено методологічні підходи до оцінки характеристик дистанції зльоту. Розроблено

архітектуру нейронної мережі для прогнозування параметрів літального апарату, включаючи роботу з детермінованими та недетермінованими наборами даних.

Описано процеси створення, навчання та тестування нейромереж у середовищі PyTorch. Розроблено та реалізовано компоненти нейромережі, включаючи автоматичне диференціювання й оптимізацію параметрів моделі.

Отримані результати дозволяють використовувати створену модель нейромережі для ефективного прогнозування характеристик літальних апаратів, що відкриває перспективи для подальших досліджень і впровадження у практичні технічні системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. K. Noor, "Nondeterministic Approaches and Their Potential for Future Aerospace Systems," NASA Langley Research Center, Hampton, 2001.
2. T. A. Cruse, "Non-Deterministic, Non-Traditional Methods (NDNTM)," NASA Glenn Research Center, Cleveland, 2001.
3. S. J. Rey, "Mathematical Models in Geography," International Encyclopedia of the Social & Behavioral Sciences (Second Edition), vol. 14, no. 2, pp. 785-790, 2015.
4. W. Schwarzacher, "Chapter 3 Deterministic models," Developments in Sedimentology, vol. 19, pp. 37-59, 2008.
5. M. Urquidi-Macdonald and D. D. Macdonald, "Performance Comparison Between a Statistical Model, a Deterministic Model, and an Artificial Neural Network Model for Predicting Damage for Pitting Corrosion," Journal of Research of the National Institute of Standards and Technology, vol. 99, no. 4, pp. 495-504, 1994.
6. D. N. Mavris and J. S. Schutte, "Application of Deterministic and Probabilistic System Design Methods and Enhancements of Conceptual Design Tools for ERA Project Final Report," Langley Research Center, Hampton, 2016.
7. D. Gonze, J. Hallow and A. Goldbeter, "Deterministic versus Stochastic Models for Circadian Rhythms," Journal of Biological Physics, vol. 28k, pp. 637-653, 2002.
8. R. M. Vogel, "Stochastic and Deterministic World Views," Journal of Water Resources Planning and Management, vol. 125, no. 6, pp. 311-313, 1999.
9. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Montreal: MIT Press, 2015.
10. S. Russell and P. Norvig, Artificial intelligence - A modern approach, Upper Saddle River, New Jersey: Pearson Education, Inc., 2010.
11. S. Herbert and A. Newell, "Human problem solving: The state of the theory in 1970," American Psychologist, vol. 26, no. 2, pp. 145-159, 1971.

12. G. W. Ernst, "GPS and Decision Making: An Overview," Theoretical Approaches to Non-Numerical Problem Solving, vol. 28, pp. 59-107, 1970.
13. L. Fausett, Fundamentals of neural networks - Architectures, algorithms, and applications, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1994.
14. Liljeqvist, "The Essence of Artificial Neural Networks," 20 March 2016. [Online]. Available: https://medium.com/@ivanliljeqvist/the-essence-of-artificial-neural-networks-5de300c995d6. [Accessed 18 September 2019].
15. W. S. McCulloch and W. H. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," Bulletin of Mathematical Biophysics, vol. 5, no. 1, pp. 115-133, 1943.
16. N. Whitehead and B. Russell, Principia Mathematica, Cambridge: Cambridge University Press, 1910.
17. Turing, "Computing Machinery and Intelligence," Mind, vol. 59, no. 236, pp. 433-460, 1950.
18. D. O. Hebb, The Organization of Behavior, Montreal: John Wiley & Sons, 1949.
19. D. E. Rumelhart and J. L. M. G. E. hinton, "A General Framework for Parallel Distributed Processing," 1986.
20. M. Baroni and A. Kilgarriff, "WebBootCaT: a web tool for instant corpora," Computational Lexicography and Lexicology, 2006.
21. D. Yarowsky, "Unsupervised Word Sense Disambiguation Rivaling Supervised Methods," in 33rd Annual Meeting of the Association for Computational Linguistics, Cambridge, 1995.
22. M. J. Pedelty, "A Review of the Field of Artificial Intelligence and its Possible Applications to NASA Objectives," School of Government and Public Administration, Washington, D.C., 1965.
23. H. J. Dunn and R. C. Montgomery, "A Moving Window Parameter Adaptive Control System for the F8-DFBW Aircraft," IEEE Transactions on Automatic Control, Vols. AC-22, no. 5, pp. 788-795, 1977.

24. Massachusetts Institute of Technology Lincoln Laboratory, "DARPA Neural Network Study Final Report," Tactical Technology Office of the U.S. Defense Advanced Research Projects Agency (DARPA/TTO), Lexington, MA, 1989.
25. L. Harrison, P. Saunders and J. Janowitz, "Artificial Intelligence with Applications for Aircraft," Federal Aviation Administration (FAA), Springfield, Virginia, 1994.
26. W. H. Cheung, Neural Network Aided Aviation Fuel Consumption Modeling, M.S. thesis, Blacksburg, Virginia: Virginia Polytechnic Institute and State University, 1997.
27. D. L. Simon and T. W. Long, "Adaptive Optimization of Aircraft Engine Performance Using Neural Networks," NASA - U.S. Army Research Laboratory, Cleveland, OH, 1995.
28. D. J. Linse and R. F. Stengel, "Identification of Aerodynamic Coefficients Using Computational Neural Networks," in Aerospace Sciences Meeting (AIAA), Reno, NV, 1992.
29. S. Kim and A. J. Calise, "Nonlinear Flight Control Using Neural Networks," Journal of Guidance, Control and Dynamics, vol. 20, no. 1, pp. 26-33, 1997.
30. J. Calise and R. Rysdyk, "Nonlinear adaptive flight control using neural networks," Control Systems, vol. 18, no. 6, pp. 14-25, 1999.

ДОДАТОК А

```
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.layers import Conv2D
from keras.models import Sequential
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.optimizers import SGD
from keras.layers import Flatten
def loadDataset():
(train_X, train_Y), (test_X, test_Y) = mnist.load_data()
train_X = train_X.reshape((train_X.shape[0], 28, 28, 1))
test_X = test_X.reshape((test_X.shape[0], 28, 28, 1))
train_Y = to_categorical(train_Y)
test_Y = to_categorical(test_Y)
return train_X, train_Y, test_X, test_Y
def preprocess(trainSet, testSet):
train_norm = trainSet.astype('float32')
test_norm = testSet.astype('float32')
train_norm = train_norm / 255.0
test_norm = test_norm / 255.0
return train_norm, test_norm
def modelDefinition():
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1),
kernel_initializer='he_uniform'))
```

```
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))
opt = SGD(lr=0.01, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
return model
def modelEvaluation(Xdata, Ydata, n_folds=5):
scores, histories = list(), list()
kfold = KFold(n_folds, random_state=1, shuffle=True)
for train_ix, test_ix in kfold.split(Xdata):
model = define_model()
train_X, train_Y, test_X, test_Y = Xdata[train_ix], Ydata[train_ix], Xdata[test_ix], Ydata[test_ix]
history = model.fit(train_X, train_Y, epochs=10, batch_size=32, validation_data=(test_X, test_Y), verbose=0)
_, acc = model.evaluate(test_X, test_Y, verbose=0)
print('> %.3f' % (acc * 100.0))
scores.append(acc)
histories.append(history)
return scores, histories
def summary(scores):
print('Accuracy: mean=%.3f' % (mean(scores)*100))
pyplot.boxplot(scores)
pyplot.show()
def runTest():
train_X, train_Y, test_X, test_Y = loadDataset()
train_X, test_X = preprocess(train_X, test_X)
scores, histories = modelEvaluation(train_X, train_Y)
```

```
summary(scores)
runTest()
```

ДОДАТОК Б

Розроблені засоби нейронної мережі

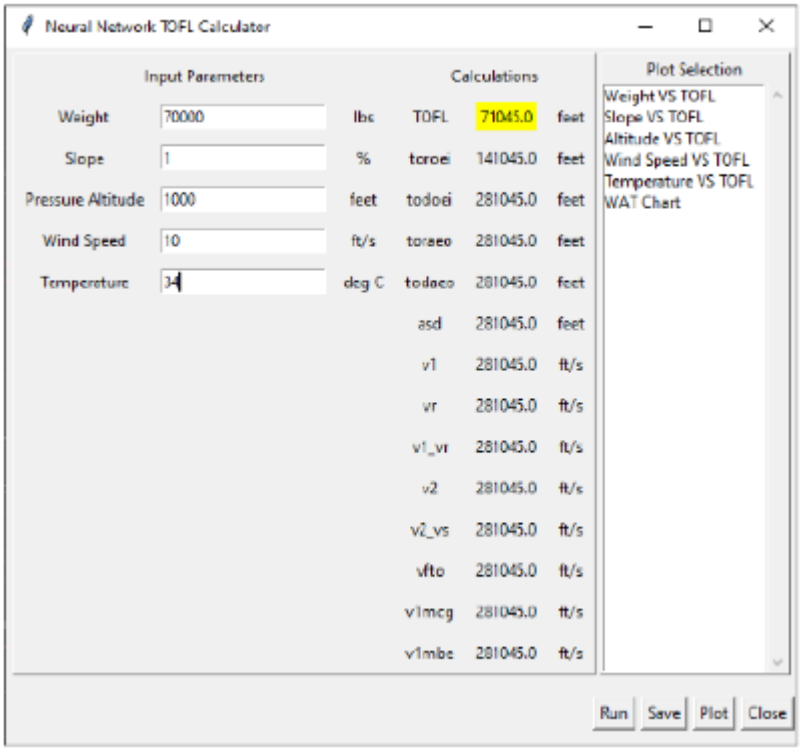


Рисунок 1.Інструмент прогнозування TOFL

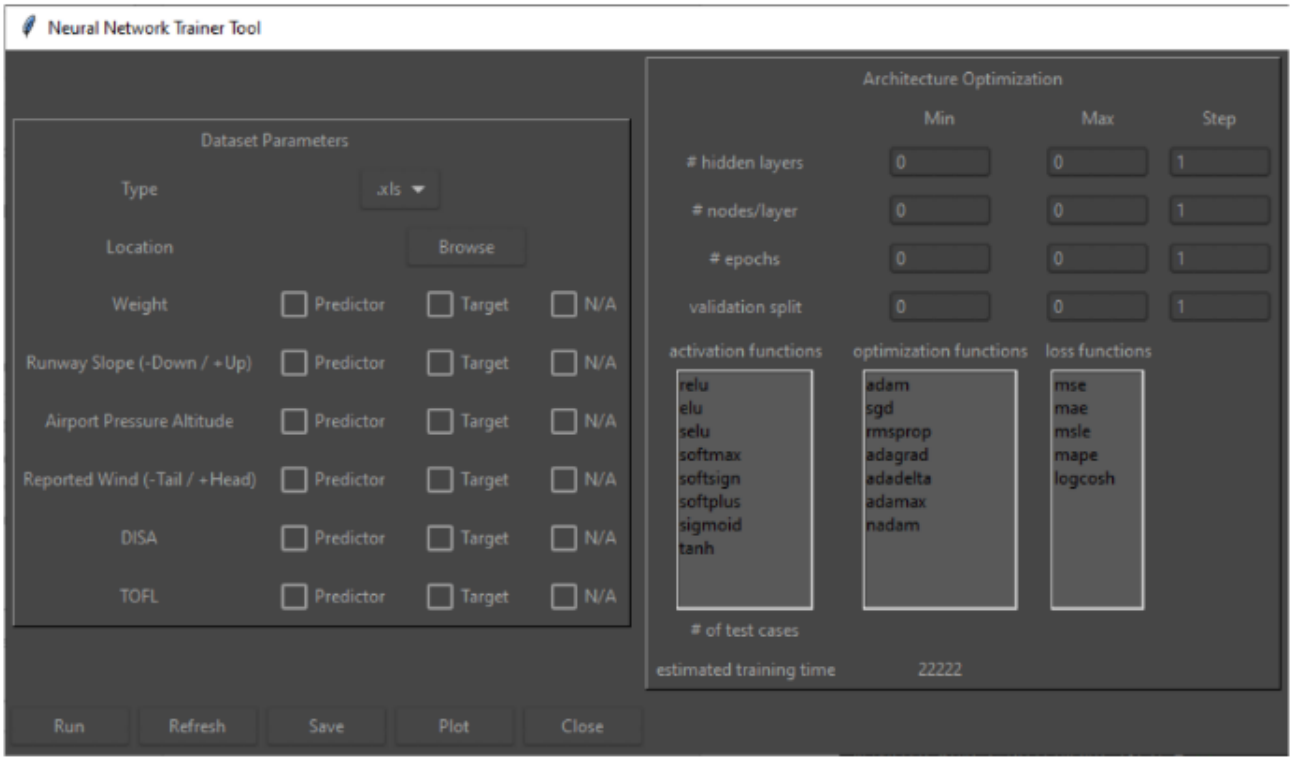


Рисунок 2. Інструмент оптимізації архітектури NN