Київський національний торговельно-економічний університет

Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка програмного забезпечення прогнозування погодних умов із застосуванням нейронної мережі»

Студента 2 курсу, 6м групи, спеціальності 121 «Інженерія програмного забезпечення», спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Маркевича Богдана Степановича

Науковий керівник кандидат педагогічних наук, старший викладач кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Жирова Тетяна Олександрівна

Гарант освітньої програми доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Криворучко Олена Володимирівна

КИЇВ – 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Спеціалізація Інженерія програмного забезпечення

Затверджую
Зав. кафедри інженерії програмного
програмного забезпечення та
кібербезпеки
Криворучко О. В.
8 листопада 2019 р.

Завдання
на випускний кваліфікаційний проект студентові

Маркевичу Богдану Степановичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Розробка програмного забезпечення прогнозування погодних умов із застосуванням нейронної мережі»
   Затверджена наказом ректора від "13" грудня 2019 р. № 4304

2. Строк здачі студентом закінченого проекту 01 грудня 2020 р.

3. Цільова установка та вихідні дані до проекту

Мета проекту дослідження алгоритмів та засобів для створення нейронної мережі.

Об'єкт дослідження процес моделювання, розробки та навчання нейронних мереж.

Предмет дослідження створення мобільного додатку за допомогою фреймворку Flutter.

4. Консультанти проекту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ НЕЙРОННИХ МЕРЕЖ ТА МАШИННОГО НАВЧАННЯ

1.1. Поняття нейронної мережі

1.2. Штучні нейрони як основна складова мережі

1.3. Парадигми навчання нейронних мереж

1.4. Модель нейронної мережі

1.5. Використання нейронних мереж

1.6. Висновки до розділу 1

РОЗДІЛ 2. ПЕРСЕПТРОН

2.1. Зв'язки між нейронами. Синапс.

2.2. Правило корекції за помилкою

2.3. Функція активації. Сигмоїд.

2.4. Висновки до розділу 2

РОЗДІЛ 3. ІНФРАСТРУКТУРА ДЛЯ РОЗРОБКИ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Операційні системи Android та IOS

3.2. Мова програмування Dart та фреймворк Flutter

3.3. REST API та SQLite як способи контролю даних

3.4. Висновки до розділу 3

РОЗДІЛ 4. ОПИС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Загальна характеристика

4.2. Головний екран та його складові

4.3. Екран прогнозування погоди

4.4. Висновки до розділу 4

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ТЕХНІЧНЕ ЗАВДАННЯ

КЕРІВНИЦТВО КОРИСТУВАЧА

ПРОГРАМА ТА МЕТОДИКА ТЕСТУВАННЯ .....

ДОДАТКИ

6. Календарний план виконання проекту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1.	Вибір теми випускної кваліфікаційної роботи	20.09.2019	20.09.2019
2.	Розробка та затвердження завдання на проект магістра	08.11.2019	08.11.2019
3.	Вступ та перелік літературних джерел	24.01.2019	24.01.2019
4.	Наукова стаття	01.09.2020	01.09.2020
5.	Технічне завдання (ТЗ)	28.02.2020	28.02.2020
6.	Розділ 1. Відомості щодо інструментів для розробки освітньої системи	25.06.2020	25.06.2020
7.	Розділ 2. Архітектура програмного забезпечення «STUDUCK» на стадії проекту	07.09.2020	07.09.2020
8.	Розділ 3. Програмна реалізація додатку	19.10.2020	19.10.2020
9.	Програма та методика тестування	21.10.2020	21.10.2020
10.	Керівництво користувача	23.10.2020	23.10.2020
11.	Висновки та пропозиції	30.10.2020	30.10.2020
12.	Здача випускного кваліфікаційного проекту на кафедру	03.11.2020	03.11.2020
13.	Підготовка автореферату та презентації доповіді	03.11.2020	03.11.2020
14.	Попередній захист випускного кваліфікаційного проекту	25.11.2020-27.11.2020	25.11.2020-27.11.2020
15.	Зовнішнє рецензування випускного кваліфікаційного проекту	01.12.2020	01.12.2020
16.	Підготовка до публічного захисту випускного кваліфікаційного проекту	08.12.2020-09.12.2020	

7. Дата видачі завдання «8» листопада 2019 р.

8. Науковий керівник випускного кваліфікаційного проекту Жирова Т. О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Криворучко О. В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Маркевич Б. С.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

Науковий керівник випускного кваліфікаційного проекту _____
Жирова Т. О.
(ПІБ, підпис, дата)

Відмітка про попередній захист Жирова Т. О.
(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента Маркевича Б. С.
(прізвище, ініціали)
може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Криворучко О. В.
(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.
(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

Анотація

Маркевич Б. С. Розробка програмного забезпечення прогнозування погодних умов із застосуванням нейронної мережі.

У випускній кваліфікаційній роботі розглядається розробка, моделювання та процес навчання нейронної мережі, також створення мобільного програмного забезпечення на основі фреймворку Flutter, мови програмування Dart, використання REST API технологій.

Ключові слова: нейронна мережа, нейрон, персептрон, синапс, функція активації, функція активації, Flutter, Dart.

Annotation

Markevych B. S. Development of software for weather forecasting using a neural network.

The final qualification work deals with the development, modeling and learning process of the neural network, as well as the creation of mobile software based on the Flutter framework, Dart programming language, the use of REST API technologies.

Key words: neural network, neuron, perceptron, synapse, activation function, activation function, Flutter, Dart.

ЗМІСТ

ВСТУП

Машинне навчання – надзвичайно гаряча сфера в галузі штучного інтелекту та науки про дані. Немає сумнівів, що нейронні мережі є найбільш розгляданими та широко використовуваними техніками машинного навчання.

Багато вчених з питань даних використовують нейронні мережі, не розуміючи їх внутрішньої структури. Однак розуміння внутрішньої структури та механізму таких технік машинного навчання дозволить їм ефективніше вирішувати проблеми. Це також дозволяє їм налаштовуватися, налаштовувати і навіть розробляти нові нейронні мережі для різних проектів.

Актуальність обраної теми полягає в тому, що машинне навчання стає популярним у всіх галузях промисловості щомісяця з основною метою підвищення доходів та зменшення витрат. Нейронні мережі - надзвичайно практичні методи машинного навчання в різних проектах. Їх можна використовувати для автоматизації та оптимізації процесу вирішення складних завдань.

Метою дослідження є дослідження алгоритмів та засобів для створення нейронної мережі.

Об'єктом дослідження процес моделювання, розробки та навчання нейронних мереж.

Предметом дослідження створення мобільного додатку за допомогою фреймворку Flutter.

					КНТЕУ 121-06-13.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	*Розробка програмного забезпечення прогнозування погодних умов із застосуванням нейронної мережі*	*Стадія*	*Аркуш*	*Аркушів*
Зав. каф.		Криворучко О.В.		*24.01.2019*		В	4	42
Керівник		Жирова Т.О.		*24.01.2019*				
Гарант		Криворучко О.В.		*24.01.2019*	*Вступ*	Факультет інформаційних технологій, 2 курс, 6м група		
Розробив		Маркевич Б.С.		*24.01.2019*				

Задачі дослідження:

- розробити нейронну мережу, яка буде здатна прогнозувати погодні умови;
- розробити мобільний додаток

Наукова новизна дослідження полягає в тому, що в процесі виконання проекту буде створено повноцінний додаток, який буде повністю адаптований для користувачів.

Виконання випускного кваліфікаційного проекту буде здійснюватися за допомогою теоретичних *методів дослідження.* Тобто, для виконання поставленої мети, буде виконано аналіз існуючих обставин та визначення вимог.

РОЗДІЛ 1.
ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ НЕЙРОННИХ МЕРЕЖ ТА МАШИННОГО НАВЧАННЯ

1.1. Поняття нейронної мережі

Штучні нейронні мережі (ANN), які зазвичай просто називають нейронними мережами (NN), – це обчислювальні системи, нечітко натхненні біологічними нейронними мережами, які складають мозок тварин.

ANN базується на колекції з'єднаних одиниць або вузлів, званих штучними нейронами, які вільно моделюють нейрони біологічного мозку. Кожне з'єднання, як синапси в біологічному мозку, може передавати сигнал іншим нейронам. Штучний нейрон, який отримує сигнал, потім обробляє його і може сигналізувати про нейрони, підключені до нього. "Сигнал" у зв'язку є дійсним числом, і вихід кожного нейрона обчислюється якоюсь нелінійною функцією суми його входів. З'єднання називаються ребрами. Нейрони та краї зазвичай мають вагу, яка регулюється в процесі навчання. Вага збільшує або зменшує силу сигналу при з'єднанні. Нейрони можуть мати такий поріг, що сигнал надсилається лише в тому випадку, якщо сукупний сигнал перетинає цей поріг. Як правило, нейрони агрегуються в шари. Різні шари можуть виконувати різні перетворення на своїх входах. Сигнали рухаються від першого шару (вхідного шару) до останнього шару (вихідного шару), можливо, після багаторазового обходу шарів.

Уоррен Маккалок і Уолтер Піттс (1943) відкрили тему, створивши обчислювальну модель для нейронних мереж. Наприкінці 40-х років Д. О. Хебб створив гіпотезу навчання, засновану на механізмі нервової пластичності, який став відомим як геббіанське навчання. Фарлі та Уеслі А. Кларк (1954) спочатку використовували обчислювальні машини, які потім називали "калькуляторами", для імітації геббіанської мережі. Розенблат (1958) створив персептрон. Перші функціональні мережі з багатьма шарами були опубліковані Івахненком та Лапою в 1965 році як груповий метод обробки даних. Основи безперервного зворотного розповсюдження були виведені в контексті теорії управління Келлі в 1960 р. І Брайсоном в 1961 р. , використовуючи принципи динамічного програмування.

У 1970 р. Сеппо Ліннайнмаа опублікував загальний метод автоматичного диференціювання (AD) дискретних підключених мереж вкладених диференційованих функцій. У 1973 році Дрейфус використовував зворотне розповсюдження для адаптації параметрів контролерів пропорційно градієнтам помилок. Алгоритм зворотного розповсюдження Вербоса (1975) дозволив практичне навчання багаторівневих мереж. У 1982 р. Він застосував метод AD Ліннайнмаа до нейронних мереж способом, який став широко застосовуватися. Потім дослідження застосовувались після Мінські та Паперта (1969), який виявив, що основні персептрони не здатні обробляти ексклюзив або схему і що комп'ютерам не вистачає потужності для обробки корисних нейронних мереж.

Розвиток дуже масштабної інтеграції (ОМС) метал-оксид-напівпровідник (МОП) у формі додаткової технології МОП (КМОП) дозволило збільшити кількість транзисторів МОП в цифровій електроніці. Це забезпечило більшу обробну потужність для розвитку практичні штучні нейронні мережі у 1980-х рр.

У 1992 році було введено макс-пулінг, щоб допомогти з мінімальним зсувом незмінності та толерантності до деформації, щоб допомогти 3D-розпізнаванню об'єктів. Шмідхубер прийняв багаторівневу ієрархію мереж (1992), попередньо підготовлену по одному рівню за допомогою безконтрольного навчання та доопрацьованої шляхом зворотного поширення.

Джеффрі Хінтон та співавт. (2006) запропонував вивчити подання високого рівня з використанням послідовних шарів двійкових або дійсних прихованих змінних із обмеженою машиною Больцмана для моделювання кожного шару. У 2012 році Нг та Дін створили мережу, яка навчилася розпізнавати концепції вищого рівня, такі як коти, лише переглядаючи зображення без міток. Безконтрольна попередня підготовка та збільшення обчислювальної потужності від графічних процесорів та розподілених обчислень дозволили використовувати більші мережі, особливо у проблемах розпізнавання зображень та зору, що стало відомим як "глибоке навчання".

Ciresan and colleagues (2010) показали, що, незважаючи на проблему зникнення градієнта, графічні процесори роблять зворотне розповсюдження можливим для багатошарових нейронних мереж прямого зв'язку. У період з 2009 по 2012 рр. АНН почали вигравати призи в конкурсах АНН, наближаючись до виконання людським рівнем різноманітних завдань, спочатку в розпізнаванні шаблонів та машинному навчанні. Наприклад, двонаправлена і багатовимірна тривала короткочасна пам’ять (LSTM) від Graves et al. виграв три конкурси з розпізнавання зв’язаного рукописного тексту в 2009 році, не маючи жодних попередніх знань про три мови, які слід вивчити.

Ciresan and colleagues створили перші розпізнавачі шаблонів для досягнення конкурентоспроможності людини / над людської діяльності на таких орієнтирах, як розпізнавання дорожніх знаків (IJCNN 2012).

Нейронні мережі навчаються, обробляючи приклади, кожен з яких містить відомі "вхідні дані" та "результат", утворюючи зважені за ймовірністю асоціації між ними, які зберігаються в структурі даних самої мережі. Навчання нейронної мережі з поданого прикладу, як правило, проводиться шляхом визначення різниці між оброблюваним виходом мережі (часто передбачуваним) та цільовим результатом. У цьому полягає помилка. Потім мережа коригує свої зважені асоціації відповідно до правила навчання та використовуючи це значення помилки. Послідовне коригування призведе до того, що нейронна мережа видаватиме результат, який все більше схожий на цільовий результат. Після достатньої кількості цих коригувань навчання може бути припинено на основі певних критеріїв. Це відоме як контрольоване навчання.

Такі системи "вчаться" виконувати завдання, розглядаючи приклади, як правило, без програмування певних правил для конкретних завдань. Наприклад, при розпізнаванні зображень вони можуть навчитися ідентифікувати зображення, що містять котів, аналізуючи приклади зображень, які вручну позначені як "кішка" або "немає кота", і використовуючи результати для ідентифікації котів на інших зображеннях. Вони роблять це без будь-якого попереднього знання котів, наприклад, що у них є шерсть, хвости, вуса та котячі обличчя. Натомість вони автоматично генерують ідентифікаційні характеристики на прикладах, які вони обробляють.

1.2. Штучні нейрони як основна складова мережі

ANN складаються із штучних нейронів, які концептуально походять від біологічних нейронів. Кожен штучний нейрон має вхідні дані і виробляє єдиний вихід, який може бути надісланий кільком іншим нейронам. Вхідними даними можуть бути значення характеристик вибірки зовнішніх даних, таких як зображення або документи, або вони можуть бути виходами інших

нейронів. Виходи кінцевих вихідних нейронів нейронної мережі виконують таке завдання, як розпізнавання об’єкта на зображенні.

Щоб знайти вихід нейрону, спочатку беремо зважену суму всіх входів, зважену вагами зв’язків від входів до нейрона. До цієї суми додаємо термін упередженості. Цю зважену суму іноді називають активацією. Потім ця зважена сума передається через (як правило, нелінійну) функцію активації для отримання вихідних даних. Початкові дані - це зовнішні дані, такі як зображення та документи. Кінцеві результати виконують завдання, наприклад, розпізнавання об’єкта на зображенні.

1.3. Парадигми навчання нейронних мереж

Існує дві парадигми навчання нейронних мереж – з учителем і без вчителя. У першому випадку, на вхідний вектор є готова відповідь, у другому випадку нейронна мережа самонавчається. У кожного виду навчання є своя ніша завдань і за великим рахунком вони не перетинаються. На даний момент придумано і запатентовано велика кількість архітектур нейронних мереж і методів їх навчання. Але основними (вихідними) є - для навчання з учителем це «алгоритм зворотного поширення помилки», а для навчання без учителя це алгоритми Хебба і Кохонена. Ці парадигми сильно перетинаються з біологічної дійсністю, наприклад - дитина навчається з учителем або без?

Також, останнім часом, сформувалася нова, третя парадигма - навчання з підкріпленням.

Навчання з вчителем

Ця категорія навчання має справу в парах X '; і Y ';, і завдання - відобразити їх у функції f: X → Y. Тут Y дана - це вчитель, планові бажані виходи, і X дані - це дані, які генерують Y дані. Це аналогічно вчителю, який навчає всіх виконувати певне завдання.

Одна виняткова особливість цієї парадигми навчання - пряме посилання на помилку, яка просто порівнює між плановим і поточним результатом. Параметри мережі надходять в вартісну функцію, яка визначає кількість невідповідність між бажаним і поточним висновком.

Навчання з учителем – дуже придатна для задач, які заздалегідь надає патерн, мета досягнення. Приклади: класифікація картинок, розпізнавання голосу, функція апроксимації, прогнозування. Зверніть увагу, що нейромережа слід забезпечити попередніми знаннями парою вводити-незалежних значень (X) і висновком Класифікації-залежних значень (Y). Присутність залежних висновком – обов'язкова умова для навчання з учителем.

Навчання без учителя

В навчанні без учителя ми маємо справу лише з даними без міток або класифікації; замість цього наша нейронна структура намагається намалювати логічні висновки і витягти знання з цього використовуючи тільки вхідні дані X.

Це аналогічно самонавчання коли хтось навчає його / її завдяки його / її досвіду і встановлюючи побічні критерії. в навчанні без вчителя ми не визначаємо бажаний патерн для кожного спостереження але нейронна структура може навчатися без будь-якого вчителя.

Тут, функція вартості (cost function) грає важливу роль. Вона сильно впливає на значення нейрона так само добре як і зв'язок між вхідними значеннями.

Приклади завдань, де використовується навчання без вчителя: кластеризація, компресія даних, статистичні моделі і мовні моделі.

1.4. Модель нейронної мережі

Модель нейронної мережі описує її архітектуру і конфігурацію, а також використовувані алгоритми навчання.

Архітектура нейронної мережі визначає загальні принципи її побудови (плоско слоїста, повнозв'язна, слабозв'язна, прямого поширення, рекурентна і т.д.).

Конфігурація конкретизує структуру мережі в рамках даної архітектури: число нейронів, число входів і виходів мережі, які використовуються активаційні функції.

Розрізняють такі базові архітектури:

- мережі прямого поширення – всі зв'язки направлені строго від вхідних нейронів до вихідних. До таких мереж відносяться, наприклад персептрон Розенблатта і багатошаровий персептрон;
- рекурентні нейронні мережі - сигнал з вихідних нейронів або нейронів прихованого шару частково передається назад на нейрони вхідного шару мережі;
- мережі радіально-базисних функцій – мережі, що містять єдиний прихований шар, нейрони якого використовують радіально-симетричну активаційну функцію, застосовуються для вирішення задач класифікації та прогнозування;
- мережі Кохонена – клас мереж, що використовують навчання без учителя і призначених для вирішення завдань кластеризації. Вони містять всього два прошарки: вхідний (розподільний) і вихідний (кластеризуються);
- карти Кохонена або карти ознак, що самоорганізуються – різновид мереж Кохонена, в яких число вихідних нейронів вибирається багато

більше числа формованих кластерів. Використовуються для візуалізації результатів кластеризації багатовимірних даних;

- повнозв'язні мережі – нейронні мережі, в яких кожен нейрон пов'язаний з усіма іншими нейронами. Такі мережі мають найвищу щільність зв'язків;
- слабозв'язні мережі – в них нейрони з'єднані тільки зі своїми найближчими сусідами;
- нейронні мережі з плоскими шарами – в них нейрони утворюють каскади, звані шарами, при цьому нейрони кожного шару пов'язані з усіма нейронами наступного і попереднього шарів, а всередині шару зв'язків немає. Мережі з плоскими шарами можуть бути одношарові (містити один прихований шар) і багатошаровими (містити кілька прихованих шарів).

Кожна архітектура мережі призначення для вирішення певного класу задач аналізу даних (регресії, класифікації, кластеризації, прогнозування) і використовує спеціальні алгоритми навчання.

1.5. Використання нейронних мереж

До теперішнього часу найбільшого поширення набули такі види нейромереж: - згорткові нейронні мережі (CNN) імітують роботу зорової кори головного мозку і частково виконують функцію абстрактного мислення. Вони прекрасно справляються із завданням розпізнавання зображень, а їх обчислення легко распараллелить на графічних процесорах, що дозволяє створювати відносно дешеві апаратні платформи з елементами ШІ.

CNN застосовуються в системах машинного зору безпілотних автомобілів, комерційних дронів, роботів, а також в охоронному відеоспостереженні. Ви теж кожен день використовуєте CNN, якщо включили

в налаштуваннях смартфона розблокування за допомогою розпізнавання особи.

- рекурентні нейронні мережі (RNN) володіють короткочасною пам'яттю, за рахунок чого легко аналізують послідовності довільної довжини. RNN розбивають потік даних на елементарні частини і оцінюють взаємозв'язки між ними. Ці алгоритми знайшли основне застосування в розпізнаванні рукописного тексту й мови. Коли ви шукаєте мелодію на слух в Shazam, розмовляєте з Siri, Google Now або Алісою від «Яндекса», залишаєте замітки від руки для Cortana - на хмарних платформах беруться за справу рекурентні нейромережі.

- мережі з довготривалою і короткочасною пам'яттю (LSTM) стали подальшим розвитком RNN. Вони гарні для прогнозування змін будь-якої величини (наприклад, біржових курсів або купівельного попиту) шляхом екстраполяції. Також їх застосовують для глибокого аналізу природної мови. Наприклад, Google використовує LSTM в персональному помічника і системі машинного перекладу Google Translate. Без LSTM якість перекладів так і залишалося б на рівні програм з дев'яностих, з якими часом було простіше перевести текст самому, ніж виправляти численні помилки.

- керовані рекурентні блоки (GRU) - порівняно недавня модифікація RNN, що з'явилася тільки в 2014 році. Їх використовують для синтезу мови, яка володіє емоційним забарвленням і звучить як справжня. Наприклад, в тестах сервісів Google Duplex і Microsoft XiaoIce люди не змогли відрізнити говорять ботів від живих співрозмовників. Примітно, що XiaoIce дозволила Microsoft зміцнитися на азіатському ринку, де розвиток компанії завжди стримувався мовним бар'єром.

- глибокі нейронні мережі (DNN) - будь-яка мережа більш ніж з трьома шарами. Вони лежать в основі механізмів глибокого машинного навчання,

знаходячи неявні взаємозв'язку між різнорідними даними. Яскравий приклад - пошук кореляцій між розвитком захворювань і різними потенційними факторами у величезних масивах наукових статей за допомогою IBM Watson.

- генеративно-змагальні мережі (GAN). Це комбінація нейромереж, одна з яких генерує варіанти, а інша відсіває їх (виступає в якості арбітра). Таке поєднання дозволяє реалізувати машинне навчання без учителя, що підвищує автономність II. Наприклад, PixelDTGAN генерує окремі зображення одягу, взуття та аксесуарів для каталогів онлайн-магазинів. В якості вхідних даних використовуються фотографії, на яких ці предмети гардероба демонструють фотомоделі. Поки зйомка для каталогів одягу вважається досить витратною частиною електронної комерції, але цілком можливо, що в найближчі роки нейромережі дозволяють швидко проілюструвати каталог, навіть не залучаючи фотографа. Обробка фотографій теж забирає багато часу. Ви можете спробувати зробити це набагато швидше, додаючи і прибираючи графічні об'єкти за допомогою іншої нейромережі -IBM GANPaint. Подібна їй нейросеть DRAGAN вже застосовується для автоматичної відтворення персонажів аніме і мультфільмів. Вона дозволяє прискорити вихід нових серій і утримати аудиторію розважальних каналів, не перевантажуючи аніматорів колосальним об'ємом роботи. Так що там мультфільми! GAN дозволяють анімувати тривимірну модель людини, переносячи на неї руху іншого в реальному часі. Перші результати виглядають не дуже переконливо, однак особливість будь нейромережі - в тому, що вона покращується з кожною новою порцією даних.

1.6. Висновки до розділу 1

На сьогоднішній день відомо вже більше 200 різних парадигм нейронних мереж (не лише детермінованих, але і імовірнісних), десятки нейронних мереж реалізовані в спеціалізованих кристалах і платах, на їх основі створені потужні робочі станції і навіть суперкомп’ютери. Сучасні

технології досягли того рубежу, коли стало можливим виготовлення технічної системи з 3.4 млрд. нейронів (саме така кількість їх в мозку людини). Проте їх з’єднання продовжує залишатися проблемою.

РОЗДІЛ 2
ПЕРСЕПТРОН

2.1. Зв’язки між нейронами. Синапс

В основі перцептрона лежить математична модель відтворення інформації мозку. Різні дослідники по-різному його визначають. У самому загальному своєму відео (як його описував Розенблатт) він представляє систему з елементів трьох різних типів: сенсорів, асоціативних елементів та реагуючих елементів.

Принцип роботи персептрона наступний:

a. Перші в роботу включають S-елементи. Вони можуть знаходитись або в стані спокою (сигнал становить 0), або в стані збудження (сигнал становить 1);
b. Далі сигнали від S-елементів передаються A-елементам за так званим S-A зв'язком. Ці зв'язки можуть мати ваги, лише лише -1, 0 або 1;
c. Далі сигнали від чутливих елементів, які пройшли за S-A зв'язком, потрапляють в A-елементи, які ще називають асоціативними елементами;
    - Одному A-елементу може відповідати кілька S-елементів;
    - Якщо сигнали, що поступають на A-елемент, у сукупності підвищуються деякий його поріг θ, до цього A-елемент повертається і видає сигнал, становить 1;

- У протилежному випадку (сигнал від S-елементів не перевищує порогу A-елемента), генерується нульовим сигналом;

d. Далі сигнали, які виробляються вбудованими A-елементами, направляються до сумматору (R-елемента), дію якого вже відомо. Однак, щоб потрапити до R-елемента, вони проходять по A-R зв'язку, у яких теж є ваги (які вже можуть приймати будь-які значення, від відливу від S-A зв'язку);

e. R-елемент складає один з іншим зважені сигнали від A-елементів, а потім

- якщо перевищений визначений порог, генерує вихідний сигнал, становить 1;
- якщо поріг не перевищено, то вихід персептрона становить -1.

Синапс це зв'язок між двома нейронами. У синапсів є 1 параметр - вага. Завдяки йому, вхідна інформація змінюється, коли передається від одного нейрона до іншого. Допустимо, є 3 нейрони, які передають інформацію наступному. Тоді у нас є 3 ваги, що відповідають кожному з цих нейронів. У той час нейрона, у якого вага буде більше, та інформація і буде домінуючою в наступному нейроні (приклад - змішування квітів). Насправді, сукупність вагів нейронної мережі або матриці вагів - це мозок всієї системи. Завдяки цьому ми вдячні, попередня інформація обробляється та перетворюється в результат.

2.2. Правило корекції за помилкою

Метод корекції помилок - метод навчання перцептрона, запропонований Френком Розенблаттом. Представляє собою такий метод навчання, при якому вага зв'язку не змінюється до тих пір, доки поточна реакція перцептрона залишається правильною. При появі неправильної реакції вага змінюється на одиницю, знак (+/-) визначає протилежний від знаку помилки.

У теоремі збіжності перцептрона розрізняються різні види цього методу, доведено, що будь-який з них дозволяє отримати сходження при вирішенні будь-якої задачі класифікації.

Метод корекції помилок без квантування

Якщо реакція на стимул S_i правильна, то ніякого підкріплення не вводиться, але при появі помилок до ваги кожного активного А-елемента додається величина $n = p_i \Delta x_i$, де Δx_i - число одиниць підкріплення, вибирається так, щоб величина сигналу перевищувала поріг θ, а

$$\rho_i = \begin{cases} +1, i \int S_i^+; \\ -1, i \int S_i^-. \end{cases} \quad (2.1)$$

при цьому S_i^+ - стимул, що належить позитивному класу, а S_i^- стимул, що належить негативному класу.

Метод корекції помилок з квантуванням

Відрізняється від методу корекції помилок без квантування тільки тим, що $\Delta x_i = 1$, Тобто дорівнює одній одиниці підкріплення.

Цей метод і метод корекції помилок без квантування є однаковими за швидкістю досягнення рішення в загальному випадку, і більш ефективними в порівнянні з методами корекції помилок з випадковим знаком або випадковими збуреннями.

Метод корекції помилок з випадковим знаком підкріплення

Відрізняється тим, що знак підкріплення η вибирається випадково незалежно від реакції перцептрона і з однаковою ймовірністю може бути позитивним або негативним. Але так само як і в базовому методі - якщо перцептрон дає правильну реакцію, то підкріплення дорівнює нулю.

Метод корекції помилок з випадковими збуреннями

Відрізняється тим, що величина і знак η для кожної зв'язку в системі вибираються окремо і незалежно відповідно до деякого розподілом ймовірностей. Це метод призводить до самої повільної збіжності, в порівнянні з вище описаними модифікаціями.

2.3. Функція активації. Сигмоїд.

У штучних нейронних мережах функція активації вузла визначає вихід цього вузла з урахуванням вхідних даних або набору входів. Стандартну інтегральну схему можна розглядати як цифрову мережу функцій активації, яка може бути "ON" (1) або "OFF" (0), залежно від вводу. Це схоже на поведінку лінійного персептрона в нейронних мережах. Однак лише нелінійні функції активації дозволяють таким мережам обчислювати нетривіальні проблеми, використовуючи лише невелику кількість вузлів, і такі функції активації називаються нелінійностями.

Найпоширеніші функції активації можна розділити на три категорії: функції хребта, радіальні функції та функції згину.

Функції хребта

Функції хребта - це одновимірні функції, що діють на лінійну комбінацію вхідних змінних. Часто використовувані приклади включають:

Лінійна активація:

$$\varphi(v) = a + v` \qquad (2.2)$$

Активація ReLU:

$$\varphi(v) = \max(0, a + v`b) \tag{2.3}$$

Активація Heaviside:

$$\varphi(v) = 1_{a+v`b>0} \tag{2.4}$$

Логістична активація:

$$\varphi(v) = (1 + \exp(-\mathrm{a} - \mathrm{v`b}))^{-1} \tag{2.5}$$

У біологічно натхненних нейронних мережах функцією активації, як правило, є абстракція, що представляє швидкість дії потенціалу дії в клітині. У найпростішій формі ця функція є двійковою - тобто або нейрон спрацьовує, або ні. Функція виглядає як

$$\varphi(v) = U(a + v`b) \tag{2.6}$$

, де U є функцією кроку Heaviside.

Лінія позитивного нахилу може бути використана для відображення збільшення швидкості стрільби, яке відбувається із збільшенням вхідного струму. Така функція мала б вигляд

$$\varphi(v) = a + v` \tag{2.7}$$

Оскільки біологічні нейрони не можуть знизити швидкість стрільби нижче нуля, використовуються випрямлені лінійні функції активації:

$$\varphi(v) = \max(0, a + v`b) \tag{2.8}$$

Вони вводять нелінійність при нулі, яка може бути використана для прийняття рішень. Нейрони також не можуть спрацьовувати швидше, ніж певна швидкість, мотивуючи функції активації сигмоподібної форми, область яких є кінцевим інтервалом.

Всі проблеми, згадані вище, можна вирішити за допомогою нормалізуемих сігмоідной функції активації. Одна з реалістичних моделей залишається в нульовому стані, поки не прийде вхідний сигнал, в цей момент

швидкість збудження спочатку швидко зростає, але поступово досягає асимптоти в 100% швидкості збудження. Математично, це виглядає як

$$\varphi(v_i) = U(v_i)th(v_i) \quad (2.9)$$

, де гіперболічний тангенс можна замінити будь-який сигмоид. Така поведінка реально відбивається в нейроні, оскільки нейрони не можуть фізично порушуватися швидше деякої певної швидкості. Ця модель, проте, має кілька проблем в обчислювальних мережах, оскільки функції не диференційована, що потрібно для обчислення зворотної передачі помилки навчання.

Остання модель, яка використовується в багатошарових перцептронах - сігмоідной функція активації в формі гіперболічного тангенса. Зазвичай використовуються два види цієї функції:

$$\varphi(v_i) = th(v_i) \quad (2.10)$$

, образ якої нормалізований до інтервалу [-1, 1], і

$$\varphi(v_i) = (1 + \exp(-v_i))^{-1} \quad (2.11)$$

, зрушена по вертикалі для нормалізації від 0 до 1. Остання модель вважається більш біологічно реалістичною, але має теоретичні і експериментальні труднощі з обчислювальними помилками деяких типів.

2.4. Висновки до розділу 2

Моделі персептронів - це більш спрощений вид нейронної мережі, в якій вони несуть вхід, вагу кожного входу, беруть суму зваженого входу і застосовується функція активації.

Вони приймають і конструюють лише двійкові значення, тобто перцептрони реалізуються лише для двійкової класифікації з одним обмеженням, що вони застосовуються лише для лінійно відокремлених об'єктів.

Перцептрони є основою нейронної мережі, слід добре розуміти модель персептрону, яка надає переваги при вивченні глибоких нейронних мереж.

РОЗДІЛ 3
ІНФРАСТРУКТУРА ДЛЯ РОЗРОБКИ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Операційні системи Android та IOS

Подібно до того, як операційна система Linux або Windows керує вашим настільним або портативним комп'ютером, мобільна операційна система - це програмна платформа, поверх якої інші програми можуть працювати на мобільних пристроях.

Операційна система відповідає за визначення функцій та функцій, доступних на вашому пристрої, таких як колесо великого пальця, клавіатури, WAP, синхронізація з програмами, електронна пошта, обмін текстовими повідомленнями тощо. Мобільна ОС також визначатиме, які сторонні програми (мобільні програми) можна використовувати на вашому пристрої.

Архітектура iOS - це багатошарова архітектура. На верхньому рівні iOS виступає посередником між базовим обладнанням та додатками, що створюються. Однак програми не взаємодіють безпосередньо з базовим обладнанням. Подібним чином, програми розмовляють з апаратним забезпеченням за допомогою набору чітко визначених систем інтерфейсів.

Ці інтерфейси дозволяють легко писати програми, які послідовно працюють на пристроях з різними апаратними можливостями.

Нижні рівні забезпечують базові послуги для всіх програм. Аналогічним чином, він базується на рівні верхнього рівня і надає складний графічний інтерфейс та супутні послуги.

Apple забезпечує більшість своїх системних інтерфейсів у спеціальних пакетах, званих фреймами. Фрейм - це каталог, що містить динамічну спільну бібліотеку, яка подається, наприклад, пов'язані із файлами заголовків зображення та допоміжні програми, необхідні для підтримки ресурсів цієї бібліотеки. Таким чином, кожен шар має набір фреймворку, який розробник використовує для побудови додатків.

Android – це потужна операційна система, яка підтримує велику кількість програм у смартфонах. Ці програми є більш зручними та удосконаленими для користувачів. Апаратне забезпечення, що підтримує програмне забезпечення Android, базується на архітектурній платформі ARM. Android – це операційна система з відкритим кодом, що означає, що вона безкоштовна, і кожен може нею користуватися.

Android використовує потужне ядро Linux і підтримує широкий спектр драйверів обладнання. Це забезпечує основні функціональні можливості системи, такі як управління процесами, управління пам’яттю, управління пристроями, такими як камера, клавіатура, дисплей тощо, ядро обробляє всі речі.

У верхній частині розплідника Linux є набір бібліотек, включаючи веб-браузери з відкритим кодом, такі як WebKit, бібліотека libc.

Рівень фреймворку додатків надає багато служб вищого рівня таким додаткам, як диспетчер Windows, система перегляду, менеджер пакетів, менеджер ресурсів тощо. Розробникам програм дозволяється використовувати ці служби у своєму додатку.

Всі програми можна найти у верхньому шарі. Прикладами таких програм є контакти, книги, браузери, послуги тощо. Кожна програма виконує різну роль у загальних додатках.

3.2. Мова програмування Dart та фреймворк Flutter

Dart - це мова з відкритим кодом, розроблена в Google з метою дозволити розробникам використовувати об'єктно-орієнтовану мову зі статичним аналізом типу. З моменту першого стабільного випуску в 2011 році, Dart досить сильно змінився як у самій мові, так і в основних цілях. У версії 2.0 система типу Dart перейшла з необов'язкової в статичну, і з моменту її появи Flutter став основною ціллю мови.

На відміну від багатьох мов, Dart був розроблений з метою зробити процес розробки максимально комфортним і швидким для розробників. Тож він постачається з досить широким набором вбудованих інструментів, таких як власний менеджер пакетів, різні компілятори / компілятори, парсер та форматор. Крім того, віртуальна машина Dart та збірка Just-in-Time роблять зміни коду негайно виконуваними.

Отримавши код, його можна скомпілювати рідною мовою, тому для запуску не потрібно особливого середовища. У разі веб-розробки, Dart переводиться на JavaScript.

Що стосується синтаксису, то Dart дуже схожий на такі мови, як JavaScript, Java та C ++, тому вивчення Dart, знаючи одну з цих мов, - це питання годин.

Крім того, Dart має велику підтримку асинхронності, а робота з генераторами та ітерабельними файлами надзвичайно проста.

Дарт – це мова загального призначення, і її можна використовувати майже для всього:

- У веб-додатках за допомогою бібліотеки art:html та транслятора для перетворення коду Dart у JavaScript або за допомогою таких фреймворків, як AngularDart
- На серверах, використовуючи бібліотеки art:http і art:io. Існує також декілька фреймворків, якими можна скористатися, наприклад, Aqueduct.
- У консольних додатках.
- У мобільних додатках завдяки Flutter.

Flutter – це міжплатформний набір інтерфейсів, призначений для повторного використання коду в таких операційних системах, як iOS та Android, а також дозволяє додаткам взаємодіяти безпосередньо з базовими службами платформи. Мета полягає в тому, щоб дозволити розробникам доставляти високопродуктивні додатки, які почуваються природно на різних платформах, враховуючи відмінності там, де вони існують, спільно використовуючи якомога більше коду.

Під час розробки програми Flutter працюють у віртуальній машині, яка пропонує гаряче перезавантаження змін без необхідності повної перекомпіляції. Для випуску програми Flutter компілюються безпосередньо до машинного коду, будь то інструкції Intel x64 чи ARM, або до JavaScript, якщо націлено на Інтернет. Фреймворк є відкритим кодом, має ліцензію BSD і має процвітаючу екосистему сторонніх пакетів, які доповнюють його основні функціональні можливості.

Flutter розроблений як розширювана багатошарова система. Він існує як серія незалежних бібліотек, кожна з яких залежить від базового рівня. Жоден шар не має привілейованого доступу до шару нижче, і кожна частина рівня фреймворку спроектована як не обов'язкова та замінна.

Для основної операційної системи програми Flutter упаковуються так само, як і будь-яка інша рідна програма. Специфічний для платформи

вбудовувач забезпечує точку входу; узгоджується з базовою операційною системою для доступу до таких послуг, як рендеринг поверхонь, доступність та введення; та керує циклом подій повідомлення. Embedder написаний мовою, яка підходить для платформи: в даний час Java та C ++ для Android, Objective-C / Objective-C ++ для iOS та macOS та C ++ для Windows та Linux. За допомогою embedder код Flutter може бути інтегрований до існуючої програми як модуль, або код може представляти собою весь вміст програми. Flutter включає в себе кілька вбудованих програм для загальних цільових платформ, але існують і інші вбудовувальні програми.

В основі Flutter лежить ядро, яке здебільшого написане на C++ і підтримує примітиви, необхідні для підтримки всіх додатків Flutter. Воно відповідає за растеризацію складених сцен, коли потрібно намалювати новий кадр. Також забезпечує низькорівневу реалізацію основного API Flutter, включаючи графіку (через Skia), макет тексту, файлові та мережеві вводи-виводи, підтримку доступності, архітектуру плагінів, а також середовище виконання та компіляції Dart.

Ядро піддається дії Flutter через dart:ui, який огортає базовий код C++ у класах Dart. Ця бібліотека містить примітиви найнижчого рівня, такі як класи керування підсистемами введення, графіки та тексту.

Flutter порівняно невеликий; багато функцій вищого рівня, які можуть використовувати розробники, реалізовані у вигляді пакетів, включаючи плагіни платформи, такі як камера та веб-перегляд, а також специфічні функції платформи, такі як символи, http та анімації, що базуються на основних бібліотеках Dart та Flutter. Деякі з цих пакетів походять із ширшої екосистеми, що охоплює такі послуги, як оплата в додатках, аутентифікація Apple та анімація.

3.3. REST API та SQLite як способи контролю даних

REST – це скорочення від Representational State Transfer. Це архітектурний стиль для розподілених гіпермедійних систем і вперше був представлений Роєм Філдінгом у 2000 році у своїй дисертації.

Як і будь-який інший архітектурний стиль, REST також має свої власні 6 керівних обмежень, які необхідно задовольнити, якщо інтерфейс потрібно називати RESTful. Ці принципи наведені нижче:

- Клієнт-сервер – відокремлюючи проблеми користувальницького інтерфейсу від проблем зберігання даних, ми покращуємо переносимість користувальницького інтерфейсу на декількох платформах та покращуємо масштабованість, спрощуючи компоненти сервера.
- Незалежність – кожен запит від клієнта до сервера повинен містити всю інформацію, необхідну для розуміння запиту, і не може використовувати будь-який збережений контекст на сервері. Тому стан сеансу повністю тримається на клієнті.
- Кешування – Обмеження кешу вимагають, щоб дані у відповіді на запит неявно або явно позначалися як кешовані чи некешовані. Якщо відповідь можна кешувати, тоді кеш-пам’ять клієнта отримує право повторно використовувати ці дані відповідей для подальших, еквівалентних запитів.
- Уніфікований інтерфейс – Застосовуючи принцип загальності до інтерфейсу компонента, спрощується загальна архітектура системи та покращується видимість взаємодій. Для того, щоб отримати єдиний інтерфейс, для керування поведінкою компонентів необхідні численні архітектурні обмеження. REST визначається чотирма обмеженнями інтерфейсу: ідентифікація ресурсів; маніпулювання ресурсами через

представництва; самоописові повідомлення; і, гіпермедіа як двигун стану застосування.

- Багатошарова система – Стиль багатошарової системи дозволяє архітектурі складатися з ієрархічних шарів, обмежуючи поведінку компонентів, так що кожен компонент не може "бачити" далі безпосереднього шару, з яким вони взаємодіють.
- Код на вимогу (необов'язково) – REST дозволяє розширити функціональність клієнта, завантажуючи та виконуючи код у формі аплетів або сценаріїв. Це спрощує клієнтів, зменшуючи кількість функцій, необхідних для попередньої реалізації.

Ключовим абстрагуванням інформації в REST є ресурс. Будь-яка інформація, яку можна назвати, може бути ресурсом: документ або зображення, тимчасова служба, колекція інших ресурсів, невіртуальний об'єкт (наприклад, особа) тощо. REST використовує ідентифікатор ресурсу для ідентифікації конкретного ресурсу, що бере участь у взаємодії між компонентами.

Інша важлива річ, пов'язана з REST, – це методи ресурсів, які слід використовувати для здійснення бажаного переходу. Велика кількість людей помилково відносять методи ресурсів до методів HTTP GET / PUT / POST / DELETE.

Рой Філдінг ніколи не згадував жодної рекомендації щодо того, який метод застосовувати в якому стані. Все, що він наголошує, це те, що це повинен бути єдиний інтерфейс. Якщо ви вирішите, що для оновлення ресурсу буде використовуватися HTTP POST – замість того, щоб більшість людей рекомендували HTTP PUT - це нормально, і інтерфейс програми буде RESTful.

SQLite – це бібліотека програмного забезпечення, яка забезпечує реляційну систему управління базами даних. Lite в SQLite означає легкий з точки зору налаштування, адміністрування бази даних та необхідних ресурсів.

SQLite має такі помітні особливості: автономний, безсерверний, з нульовою конфігурацією, транзакційний.

Зазвичай RDBMS, такі як MySQL, PostgreSQL тощо, вимагає окремого серверного процесу для роботи. Додатки, які хочуть отримати доступ до сервера баз даних, використовують протокол TCP / IP для надсилання та отримання запитів. Це називається архітектурою клієнт / сервер. Наступна схема ілюструє дану архітектуру:

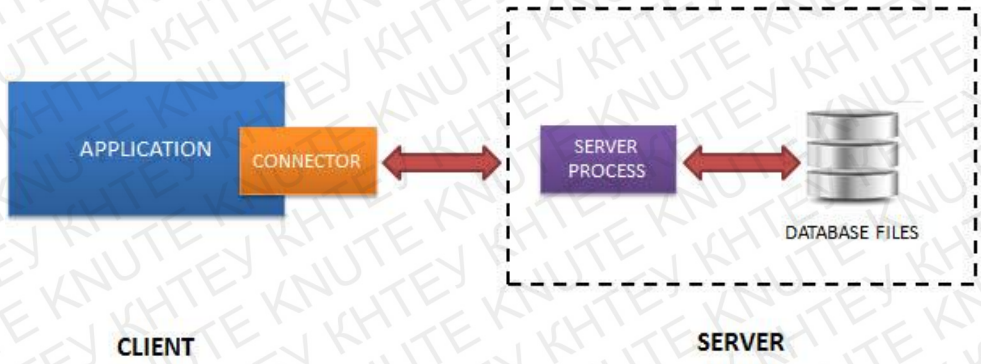


Рис. 3.1. Архітектура клієнт / сервер

База даних SQLite інтегрована з програмою, яка здійснює доступ до бази даних. Додатки взаємодіють із базою даних SQLite, що читає та записує безпосередньо з файлів бази даних, що зберігаються на диску.

Наступна схема ілюструє безсерверну архітектуру SQLite:

Рис. 3.2. Безсерверна архітектура

SQLite є автономним засобом, що вимагає мінімальної підтримки з боку операційної системи або зовнішньої бібліотеки. Це робить SQLite додатковим для використання в будь-якому середовищі, особливо на вбудованих пристроях, таких як iPhone, телефонах Android, ігрових консолях, портативних медіаплеєрах.

SQLite розроблений за допомогою ANSI-C. Його код доступний як великий sqlite3.c та його загальний файл sqlite3.h. Якщо ви хочете розробити програму, яка використовує SQLite, вам просто потрібно залишити ці файли у своєму проекті та скомпілювати його разом із кодом.

Через безсерверну архітектуру не потрібно «встановлювати» SQLite перед його використанням. Немає жодного серверного процесу, який потрібно конфігурувати, запускати та зупиняти. Крім того, SQLite не використовує жодних файлів конфігурації.

Усі транзакції в SQLite повністю сумісні з ACID. Це означає, що всі запити та зміни є атомними, послідовними, ізольованими та довговічними.

Іншими словами, всі зміни в транзакції відбуваються повністю або зовсім не відбуваються, навіть коли виникає така несподівана ситуація, як збій програми, збій живлення або збій операційної системи.

SQLite використовує динамічні типи для таблиць. Це означає, що ви можете зберігати будь-яке значення в будь-якому стовпці, незалежно від типу даних.

SQLite дозволяє єдиному підключенню до бази даних одночасно отримувати доступ до декількох файлів бази даних. Це приносить багато приємних функцій, таких як об’єднання таблиць у різних базах даних або копіювання даних між базами даних за допомогою однієї команди.

SQLite здатний створювати бази даних в пам'яті, з якими дуже швидко працювати.

3.4. Висновки до розділу 3

Dart – це об'єктно-орієнтована мова програмування, яка вперше була представлена Google в 2011 році. З тих пір Dart неухильно розвивався, випускаючи різні функції. На даний момент являється основною мовою фреймворку Flutter.

Flutter - це платформа для розробки програмного забезпечення, яка була представлена Google у 2015 році та отримала свій перший випуск у травні 2017 року. Сьогодні Flutter стабільно зростає та надає можливості не лише для мобільних розробок iOS та Android, а й для веб- та настільних додатків.

REST сервіси використовуються для безпечного та швидкого комунікування в мережі за допомогою CRUD методів.

SQLite – це вбудована реляційна система управління базами даних, сумісна з ACID. Використовується для кешування даних на клієнтах в клієнт-серверних архітектурах.

РОЗДІЛ 4
ОПИС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Загальна характеристика

Weather App пропонує яскравий та інтуїтивно зрозумілий інтерфейс, легкий доступ до прогнозів погоди, в будь-якому місці, та достатньо інших функцій, щоб випереджати більшість погодних програм, доступних для вашого телефону.

Починаючи від даних про те, як довго триватиме поточний дощовий злив, до прогнозування швидкості вітру у вашому місті наступного тижня, Weather App багатий корисними даними та функціями. Сюди входять радіолокаційні прогнози, щоденні та щотижневі прогнози, рівні температури та вітру та інші метеорологічні дані.

Однією з найкращих функцій програми є інструмент "розумний прогноз" який базується на нейронній мережі. Дана функція основана на завантаженні погодних даних з інтернету за попередній рік, їх аналізу, навчанні нейронної мережі, та видачі результатів.

Додаток доступний на iOS та Android операційних системах, за рахунок того, що використовує один “code base” та написаний на кросплатформенному фреймворку.

4.2. Головний екран та його складові

Головний екран складається із трьох основних складових:

- Поточне місцезнаходження

Рис. 4.1. Поточне місцезнаходження

- Нижнє меню

Рис. 4.2. Нижнє меню

- Контейнер прогнозу погоди

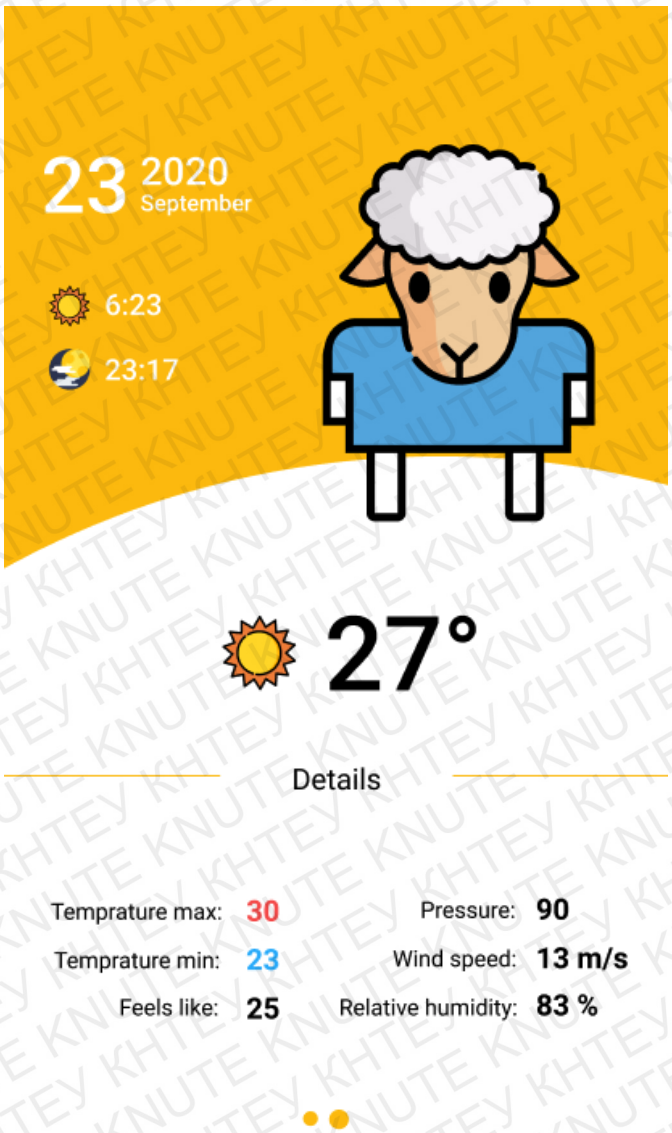


Рис. 4.3. Контейнер прогнозу погоди

За допомогою кліку на текст поточного місцезнаходження можна потрапити на екран з мапою, за допомогою якої можна вибрати будь-яку локацію, для якої потрібно отримати прогноз погоди.

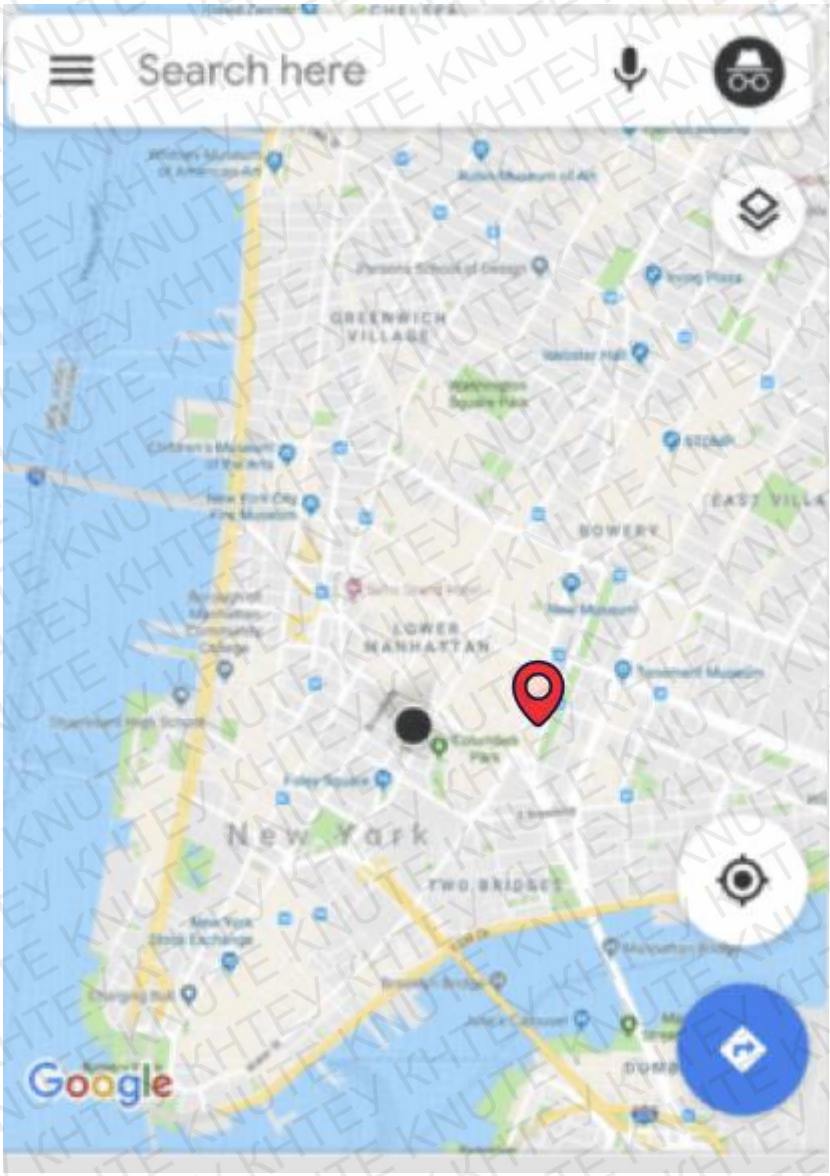
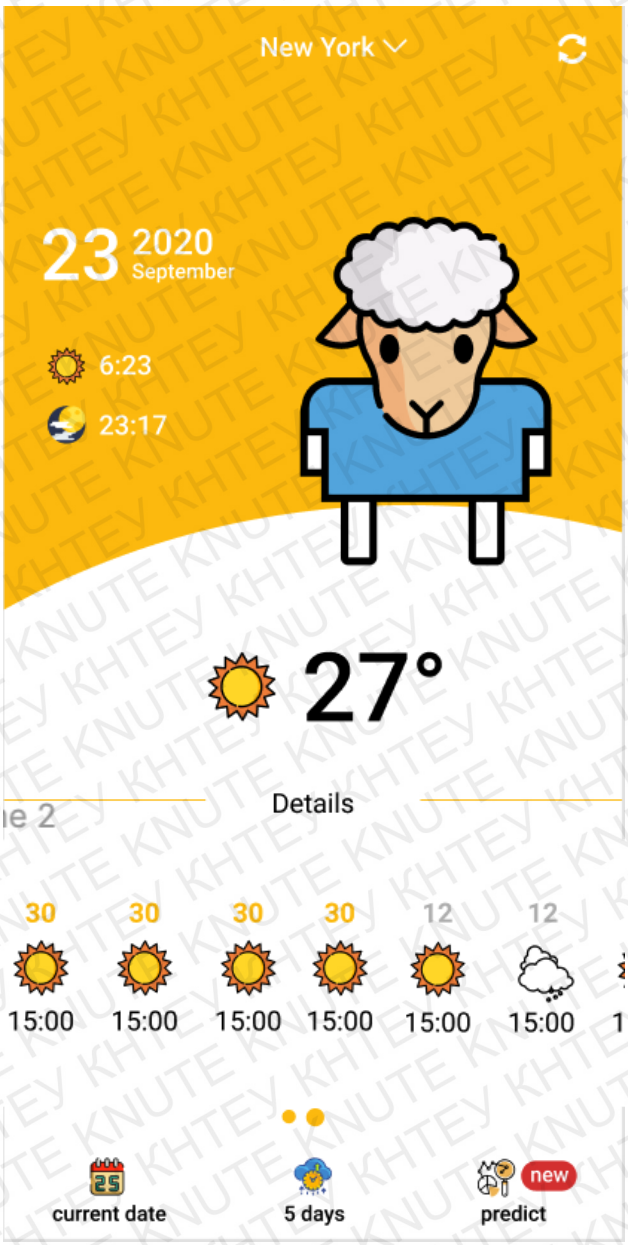


Рис. 4.4. Екран мапи

При свайпі на головному екрані по безпосередньо контейнеру із метеорологічними даними додаток перейде на погодинний прогноз для поточного дня / ночі.



Ри. 4.5. Погодинний прогноз

В нижньому меню присутні три категорії

- Прогноз для поточного дня / ночі
- Прогноз на декілька днів
- Прогноз на основі нейронної мережі

При переході а режим прогнозу на декілька днів, процес зміни днів / ночей виражений в свайпах вгору / вниз.

4.3. Екран прогнозування погоди

При переході на екран прогнозування за допомогою нейронної мережі, запускається процес завантаження та обробки даних, який ділиться на декілька етапів:

1. Завантаження даних погодних умов за останній рік відносно поточних дня та локації за допомогою REST API.
2. Кешування в базі даних SQLite
3. Ініціалізація персептрону
4. Запуск процесу навчання нейронної мережі на основі завантажених та кешованих даних
5. Вибір будь-якої дати в майбутньому та передача її у персептрон
6. Видача результату у вигляді 1 чи 0 (1 - волога погода; 0 - суха).

4.4. Висновки до розділу 4

Отже, згідно з Технічним завданням, реалізація програмного продукту не можлива без створення нейронної мережі. Тому першим етапом розробки програмного забезпечення було її створення. Наступним етапом розробки програмного забезпечення було проектування користувацького інтерфейсу.

На основі спроектованого інтерфейсу програмного продукту було написано саме програмний код з використанням об’єктно-орієнтованої мови програмування Dart. Розробку програмного продукту було розподілено на етапи згідно з визначеними класами, їх функціями та зв’язками один з одним. Всі етапи розробки було виконано, програмний продукт протестовано.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Метою роботи була розробка мобільного додатку, який використовує нейронну мережу для прогнозування погоди.

У ході роботи було проаналізовано мову програмування Dart. Позитивні та негативні сторони для подальшого використання при розробці мобільного додатку.

Було обрано архітектурний паттерн BLoCK для створення архітектури додатку.

В роботі наведено аналіз технології обміну даними REST API та їх кешування у вигляді бази даних SQLite. Розглянуті сильні та слабкі сторони технологій для визначення пріоритетів при розробці мобільного додатку.

Тому реалізований мобільний додаток є унікальним серед існуючих аналогів. Визначена платформа розробки -Android та iOS, середовище розробки Xcode та Android Studio, архітектура мобільного додатку - практичний BLoCK.

Розроблено призначений для користувача інтерфейс програми, максимальне число переходів між екранами для виконання тестових сценаріїв. Базова функціональність програмного забезпечення, реалізована в рамках магістерської роботи, і використана архітектура передбачає подальше нарощування його функціональних можливостей.

					КНТЕУ 121-06-13.МР		
					Розробка програмного забезпечення прогнозування погодних умов із застосуванням нейронної мережі	*Стадія*	*Аркуш*	*Аркушів*
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*		ВП	40	42
Зав. каф.		Криворучко О.В.		*30.10.2020*				
Керівник		Жирова Т.О.		*30.10.2020*				
Гарант		Криворучко О.В.		*30.10.2020*	*Висновки та пропозиції*	Факультет інформаційних технологій, 2 курс, 6м група		
Розробив		Маркевич Б.С.		*30.10.2020*				

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chen, Yung-Yao; Lin, Yu-Hsiu; Kung, Chia-Ching; Chung, Ming-Han; Yen, I.-Hsuan , "Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes". 2019 – 50 с.
2. McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics.
3. Kleene, S.C. "Representation of Events in Nerve Nets and Finite Automata". Annals of Mathematics Studies. Princeton University Press. 2017 - 41 с.
4. Hebb, Donald . The Organization of Behavior. New York: Wiley. 1949 – 37с.
5. Farley, B.G.; W.A. Clark (1954). "Simulation of Self-Organizing Systems by Digital Computer". IRE Transactions on Information Theory. 1954 – 56с.
6. Rosenblatt, F. "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain". Psychological Review., 1958 – 389 с.
7. Werbos, P.J. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, 1975 - 546 с.
8. Schmidhuber, J. "Deep Learning in Neural Networks: An Overview". Neural Networks. , 2015 - 786 с.
9. Mark Murphy, The Busy Coder's Guide to Advanced Android Development, 2009 - 456 с.
10. Ian Lake and Reto Meyer , Professional Android, 2018 - 487 с.

Інтернет-ресурси

					КНТЕУ 121-06-13.МР		
					Розробка програмного забезпечення прогнозування погодних умов із застосуванням нейронної мережі	*Стадія*	*Аркуш*	*Аркушів*
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*		СВД	41	42
Зав. каф.		Криворучко О.В.		*24.01.2019*				
Керівник		Жирова Т.О.		*24.01.2019*				
Гарант		Криворучко О.В.		*24.01.2019*	*Список використаних джерел*	Факультет інформаційних технологій, 2 курс, 6м група		
Розробив		Маркевич Б.С.		*24.01.2019*				

11. Flutter [Електронний ресурс] – Режим доступу: https://flutter.dev/.
12. Dart [Електронний ресурс] – Режим доступу: https://dart.dev/.
13. iOS Dev Documantation [Електронний ресурс] – Режим доступу: https://developer.apple.com/

ТЕХНІЧНЕ ЗАВДАННЯ

1. Загальні відомості

1. Найменування системи

   Weather App

2. Планові терміни початку та закінчення робіт

   Початок: 20.09.2019

   Закінчення: 25.11.2020

3. Порядок оформлення і пред'явлення результатів робіт

   1. Структура робіт проекту
   2. Детальний план робіт
   3. Матриця розподілу відповідальності в проекті
   4. Реєстр ризиків проекту
   5. Проектна документація
   6. Виконання робіт по етапам проекту та проекту в цілому

1.3.1. Виконавець формує докладний перелік робіт щодо впровадження системи за основними напрямками для кожної функціональної області згідно даного переліку вимог:

- Управління проектом
- Документація
- Користувачі
- Процеси
- Тестування

					КНТЕУ 121-06-13.МР		
					Розробка програмного забезпечення прогнозування погодних умов із застосуванням нейронної мережі	*Стадія*	*Аркуш*	*Аркушів*
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*		ТЗ	43	42
Зав. каф.		Криворучко О.В.		*28.02.2020*				
Керівник		Жирова Т.О.		*28.02.2020*				
Гарант		Криворучко О.В.		*28.02.2020*	*Технічне завдання*	Факультет інформаційних технологій, 2 курс, 6м група		
Розробив		Маркевич Б.С.		*28.02.2020*				

1.3.2. Виконавець формує та узгоджує із Замовником детальний план робіт по проекту. Планування робіт виконується за методологією, що передбачена виробником програмної платформи та Виконавцем.

1.3.3. Виконавець формує та узгоджує матрицю відповідальності (RASCI), що описує розподіл ключових робіт проекту між ролями/учасниками проектної команди.

1.3.4. Виконавець готує та узгоджує реєстр ризиків проекту, у якому надається:

• опис можливих ризиків

• вірогідність виникнення ризиків

• стратегія запобігання, мінімізації та нівелювання ризиків.

1.3.5. Виконавець готує перелік та шаблони проектної документації, що містять описи всіх необхідних налаштувань та інформацію, необхідну для впровадження системи, а саме:

• Детальний опис бізнес-вимог

• Опис реалізації бізнес-вимог

• Функціональну та технічну архітектуру

• Опис налаштувань

• Опис доопрацювань системи (функціональний дизайн)

• Опис доопрацювань системи (технічний дизайн)

• Опис конвертації даних

• Сценарії тестування функціональності

• Сценарії тестування потужності

• Документацію для навчання адміністраторів та користувачів

• Опис структури таблиць БД та їх взаємозв’язків

• Інструкції для користувачів

1.3.6. Виконання робіт по етапах проекту затверджується актами виконаних робіт.

Прийманню Замовником підлягають результати робіт виконання етапів. Про свою готовність до здачі робіт Виконавець письмово повідомляє Замовника.

Головним бенефіціаром виступаю КНТЕУ.

2. Мета та призначення створення системи

Система призначена для перегляду прогнозу погоди та прогнозуванню для днів, які не входять в перелік до основних в додатку за допомогою нейронної мережі.

Метою створення системи є можливість прогнозування погодних умов за допомогою нейронної мережі.

3. Вимоги до системи

Вимоги до системи в цілому

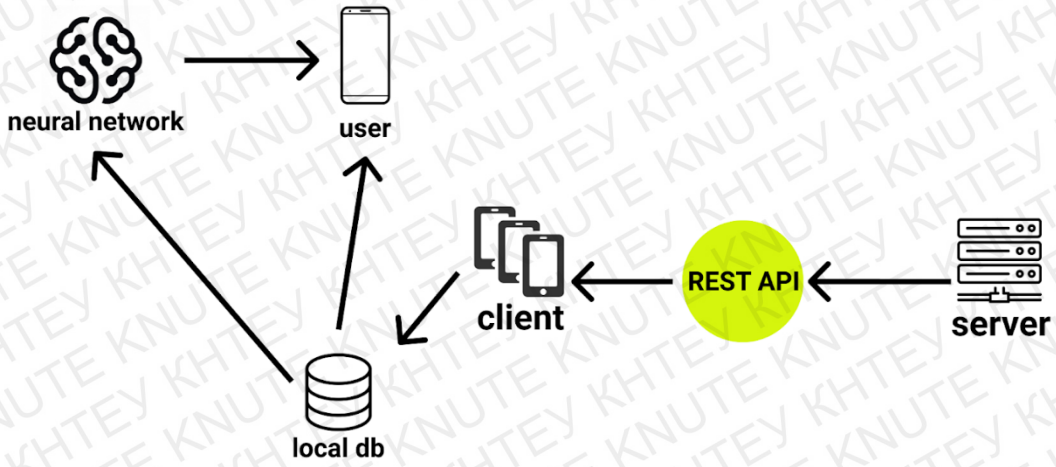
• Повинна розроблятися на базі сучасних інформаційних систем різних класів як блоків функціональності, кожен з яких відповідає затвердженим функціональним вимогам та успішно інтегрується з іншими блоками функціональності для ефективного використання із застосуванням найкращих світових практик.

• Повинна бути побудована за клієнт-серверною архітектурою.

• Повинна надавати доступ користувачам до інтерфейсу системи за допомогою існуючої internet-мережі без використання додаткових програмних комплексів та сервісів.

1. Вимоги до структури та функціонування системи, перелік підсистем
2. Вимоги до способів і засобів інформаційного обміну мі ж компонентами системи

Система повинна отримувати дані засобами REST API із сторонніх веб сервісів (за вибором Виконавця) та зберігатися локально в базі даних для подальшого використання цих даних безпосередньо системою.



Вимоги до режимів функціонування системи

Система повинна функціонувати в режимі онлайн та офлайн.

Вимоги до пристосованості системи до змін

Забезпечення пристосованості системи повинно виконуватися за рахунок автоматизованого управління:

• своєчасності адміністрування;

• модернізації процесів збору, обробки і завантаження даних відповідно до нових вимог;

• модифікації процедур доступу і представлення даних кінцевим користувачам;

Вимоги до надійності

Вимоги до надійності технічних засобів і програмного забезпечення визначаються їх розробниками.

Оцінка надійності та стабільності роботи буде оцінюватися за наступними параметрами: Uptime/Availability – безперервний час роботи або доступності системи (визначається у відсотках, наприклад 99.99%)

Основні показники надійності роботи системи будуть вимірюватись на етапі тестування та дослідної експлуатації системи. Метод дослідження та

оцінки – експериментальний, тобто в тестовому режимі мають бути відтворені основні аварійні ситуації, за яких Опис вимог мають бути підтверджені показники надійності та відновлення працездатності системи.

Вимоги до ергономіки та технічної естетики

Комунікаційне, серверне та інше обладнання в ході виконання робіт не виготовляється і вимоги щодо ергономіки та технічної естетики забезпечуються відповідними Виробниками обладнання, що використовується.

Вимоги до інформаційної безпеки

1. Можливість встановлення захищеного з'єднання між клієнтом та сервером.
2. Наявність механізмів контролю за остаточним видаленням даних.
3. Рольова модель доступу користувачів до ресурсів системи.
4. Запобігання несанкціонованого внесення змін або знищення баз даних.

Вимоги до захисту від впливу зовнішніх факторів

Для роботи системи має бути забезпечено такі режими резервування даних, система повинна їх підтримувати:

1. Живе/постійне резервування даних на рівні системи зберігання даних.

Регулярне резервування (резервування даних програмними засобами у визначені інтервали).

ПРОГРАМА ТА МЕТОДИКА ТЕСТУВАННЯ

Необхідно виконати тестування мобільного додатку. Для успішного проведення тестування, слід виконати наступні тест-кейси:

1. Робота із екраном прогнозу для поточної дати.
2. Робота із екраном прогнозу на довгостроковий період.
3. Робота прогнозом за допомогою нейронної мережі.

Кроки для виконання тест-кейсів та очікуваний результат описані в

Таблиця 7. 1

ПОРЯДОК ТЕСТУВАННЯ

Тест-кейс	**Кроки для виконання**	**Очікуваний результат**
Робота із екраном прогнозу для поточної дати	Вхід до системи.	Буде відкрито екран прогнозу для поточної дати.
	Натискання на поточну локацію.	Буде відкрито екрану вибору локації.
	Вибір нової локації та натискання кнопки «Оновити».	Відкриється екран прогнозу поточної дати для нової локації.
Робота із екраном прогнозу на довгостроковий період	Натискання кнопки «16 Days».	Буде відкрито екран довгострокового прогнозу.
	Гортання вниз / вверх	Перемикання між днями залежно від жесту.

Тест-кейс	Кроки для виконання	Очікуваний результат
Робота прогнозом за допомогою нейронної мережі	Натискання кнопки «Predict» в нижньому меню.	Відкриється екран прогнозу за допомогою нейронної мережі та запуститься процес тенування.

Якщо в результати виконання тест-кейсів будуть аналогічними очікуваному результату, тестування можна вважати успішним.

КЕРІВНИЦТВО КОРИСТУВАЧА

В додатку для роботи користувачеві можна використовувати наступні елементи:

1. Екран прогнозу погоди поточного дня із погодинним прогнозом.
2. Екран прогнозу погоди для кількох днів (перегляд відбувається за допомогою гортання елементів вниз / вверх).
3. Екран вибору локації для завантаження погоди (потрапити на нього потрібно за допомогою натискання на поточну локацію у верхній частині екрану).
4. Екран прогнозування погоди за допомогою нейронної мережі.

Для того, щоб переглянути прогнозування погоди за допомогою нейронної мережі, слід з нижнього меню перейти на відповідний екран (кнопка «Predict»).

ДОДАТКИ

Додаток А

«Лістинг програмного коду для персептрону»

```
import 'dart:async';
import 'dart:developer' as debug;
import 'dart:math';
import 'dart:core';

import 'package:meteoshipflutter/utils/weather_code_utils.dart';

import 'model/data/data_model.dart';

class Perceptron {
 List<double> _expected;
 List<List<double>> _trainInput;
 List<List<double>> _weightsInputLayer;
 List<double> _weightHiddenLayer;

 ///-----------------------------------------------
 /// Map HistoricalForecast data list into list of neutrons
 ///-----------------------------------------------
 List<List<double>> _mapDataIntoNeurons(List<HistoricalForecast> data) {
  List<List<double>> output = <List<double>>[];
  data.forEach((element) {
    output.add(element.neuron);
  });
  return output;
 }

 ///-----------------------------------------------
 /// Init training data
 ///-----------------------------------------------
 void _initTrainingData(List<HistoricalForecast> data) {
  _trainInput = _mapDataIntoNeurons(data);
  _expected = _mapExpectations(data);
  _initWeights();
 }

 List<double> _mapExpectations(List<HistoricalForecast> data) {
  var expectations = <double>[];
  data.forEach((element) {
```

```
    expectations.add(element.result);
  });
  return expectations;
 }

 ///----------------------------------------------
 /// Provide network training
 /// Need to init training data and weights before start [_initTrainingData]
 /// [learningRate] - constant of speed for training
 /// [epoch] - amount of times to run train algorithm
 ///----------------------------------------------
 void _train(int index, double learningRate) {
  var hiddenLayer =
      _transferFunctionV2(_trainInput[index], _weightsInputLayer);
  var outputLayer = _transferFunctionV2(hiddenLayer, [_weightHiddenLayer]);
  var output = outputLayer[0];
//       print(output);

  /// hidden layer
//      debug.debugger();
  var errorHiddenLayer = output - _expected[index];
  var gradientHiddenLayer = output * (1 - output);

  /// delta weight hidden layer
  var dWeightHiddenLayer = errorHiddenLayer * gradientHiddenLayer;

//    debug.debugger();
  var newWeights = <double>[];
  for (int i = 0; i < hiddenLayer.length; i++) {
    newWeights.add(_countNewWeight(_weightHiddenLayer[i], hiddenLayer[i],
        dWeightHiddenLayer, learningRate));
  }
  _weightHiddenLayer.clear();
  _weightHiddenLayer.addAll(newWeights);

  /// input layer
//       debug.debugger();
  var errorInputLayer = <double>[];
  for (int i = 0; i < _weightHiddenLayer.length; i++) {
   errorInputLayer.add(dWeightHiddenLayer * _weightHiddenLayer[i]);
  }
  var gradientInputLayer = <double>[];
  for (int i = 0; i < hiddenLayer.length; i++) {
   gradientInputLayer.add(hiddenLayer[i] * (1 - hiddenLayer[i]));
  }
```

```
  /// delta weight input layer
//    debug.debugger();
  var dWeightInputLayer = <double>[];
  for (int i = 0; i < errorInputLayer.length; i++) {
    dWeightInputLayer.add(errorInputLayer[i] * gradientInputLayer[i]);
  }
//       debug.debugger();
  var newInputWeights = <List<double>>[];
  for (int i = 0; i < _weightsInputLayer.length; i++) {
    var sublist = <double>[];
    for (int j = 0; j < _weightsInputLayer[i].length; j++) {
      sublist.add(_countNewWeight(_weightsInputLayer[i][j],
          _trainInput[index][j], dWeightInputLayer[i], learningRate));
    }
    newInputWeights.add(sublist);
  }
  _weightsInputLayer.clear();
  _weightsInputLayer.addAll(newInputWeights);
 }

 static const prodTrainingInputs = <List<double>>[
  [0.0, 0.0, 1.0],
  [1.0, 1.0, 1.0],
  [1.0, 0.0, 1.0],
  [0.0, 1.0, 1.0],
  [1.0, 1.0, 0.0],
  [1.0, 0.0, 0.0],
 ];

 static const prodTrainingResults = <double>[0.0, 1.0, 1.0, 1.0, 1.0, 0.0];

 Future<void> train(int epoch, double learningRate) async {
  return Future(() async {
    _trainInput = prodTrainingInputs;
    _expected = prodTrainingResults;
    _initWeights();
    for (int i = 0; i < epoch; i++) {
      print(i);
      for (int p = 0; p < _trainInput.length; p++) {
        _train(p, learningRate);
//         await Future.delayed(Duration(microseconds: 1));
      }
    }
  });
 }
```

```
  ///----------------------------------------------
  /// Provide network predicting
  /// Need to train or init custom weights before predicting
  /// [train] or [_initInputData]
  ///----------------------------------------------
  Future<int> predict(HistoricalForecast data) {
    return Future(() async {
      print(data.toMap());
      var hiddenLayer = _transferFunctionV2(data.neuron, _weightsInputLayer);
      var outputLayer = _transferFunctionV2(hiddenLayer, [_weightHiddenLayer]);
      print(outputLayer[0]);
//      print(dp(abs(outputLayer[0]), 5));
      return outputLayer[0].toInt();
    });
  }

  double dp(double val, int places){
    double mod = pow(10.0, places);
    return ((val * mod).round().toDouble() / mod);
  }

//  int getNearestCode(double normalizedItem) {
//    print(normalizedItem);
//    var codesTable = getNormalizedCodesTable();
//    var normalizedCodes = <double>[];
//    codesTable.keys.forEach((element) => normalizedCodes.add(element));
//    double distance = abs(normalizedCodes[0] - normalizedItem);
//    int idx = 0;
//    for (int c = 1; c < normalizedCodes.length; c++) {
//      double cdistance = abs(normalizedCodes[c] - normalizedItem);
//      if (cdistance < distance) {
//        idx = c;
//        distance = cdistance;
//      }
//    }
//    return codesTable.entries
//        .firstWhere((element) => element.key == normalizedCodes[idx])
//        .value;
//  }

  ///----------------------------------------------
  /// Return absolute value of [x]
  ///----------------------------------------------
  double abs(double x) {
    return x < 0 ? -x : x;
  }
```

```
///----------------------------------------------
/// Init synapses weights according to neurons list size
///----------------------------------------------
void _initWeights() {
 if (_weightsInputLayer != null || _weightHiddenLayer != null) {
  return;
 }
 var random = Random(1);
 var weightInputLayer = <List<double>>[];
 var weightHiddenLayer = <double>[];
 for (int i = 0; i < 2; i++) {
  var sublist = <double>[];
  for (int j = 0; j < _trainInput[0].length; j++) {
   sublist.add((random.nextDouble() + 0.1) * 0.3);
  }
  weightInputLayer.add(sublist);
 }
 for (int i = 0; i < weightInputLayer.length; i++) {
  weightHiddenLayer.add((random.nextDouble() + 0.1) * 0.3);
 }
 _weightsInputLayer = weightInputLayer;
 _weightHiddenLayer = weightHiddenLayer;
}

///----------------------------------------------
/// Function to transfer neurons and weight
/// Returns new hidden layer
/// [neurons] - matrix of neurons
/// [weights] - matrix of equivalent weights
///----------------------------------------------
List<double> _transferFunctionV2(
  List<double> neurons, List<List<double>> weights) {
 var result = <double>[];
 for (int i = 0; i < weights.length; i++) {
  double sum = 0;
  for (int j = 0; j < neurons.length; j++) {
   sum += weights[i][j] * neurons[j];
  }
  result.add(_activationFunction(sum));
 }
 return result;
}

double _countNewWeight(
  double oldWeight, double output, double dWeight, double learningRate) {
```

```
  return oldWeight - output * dWeight * learningRate;
 }

 ///-----------------------------------------------
 /// Activation function
 /// Returns new potential output
 /// [x] - value from hidden layer
 ///-----------------------------------------------
 double _activationFunction(double x) {
  return 1 / (1 - pow(e, -x));
 }

 /// Testing network ----------------------------

 static List<List<double>> _testTrainInput = [
  [0.0, 0.0, 1.0],
  [1.0, 1.0, 1.0],
  [1.0, 0.0, 1.0],
  [0.0, 1.0, 1.0],
 ];

 static List<double> _testTrainExpected = [0.0, 1.0, 1.0, 0.0];

 void trainTest() {
  var _trainInput = _testTrainInput;
  var _expected = _testTrainExpected;

  var random = Random(1);

  _weightsInputLayer = <List<double>>[
   [
    random.nextDouble(),
    random.nextDouble(),
    random.nextDouble(),
   ],
   [
    random.nextDouble(),
    random.nextDouble(),
    random.nextDouble(),
   ]
  ];
  _weightHiddenLayer = [random.nextDouble(), random.nextDouble()];

  var learningRate = 0.2;

  for (int k = 0; k < 5000; k++) {
```

```
    for (int p = 0; p < _trainInput.length; p++) {
      var hiddenLayer =
          _transferFunctionV2(_trainInput[p], _weightsInputLayer);
      var outputLayer =
          _transferFunctionV2(hiddenLayer, [_weightHiddenLayer]);
      var output = outputLayer[0];
//      print(output);

      /// hidden layer
//      debug.debugger();
      var errorHiddenLayer = output - _expected[p];
      var gradientHiddenLayer = output * (1 - output);

      /// delta weight hidden layer
      var dWeightHiddenLayer = errorHiddenLayer * gradientHiddenLayer;

//    debug.debugger();
      var newWeights = <double>[];
      for (int i = 0; i < hiddenLayer.length; i++) {
        newWeights.add(_countNewWeight(_weightHiddenLayer[i], hiddenLayer[i],
            dWeightHiddenLayer, learningRate));
      }
      _weightHiddenLayer.clear();
      _weightHiddenLayer.addAll(newWeights);

      /// input layer
//      debug.debugger();
      var errorInputLayer = <double>[];
      for (int i = 0; i < _weightHiddenLayer.length; i++) {
        errorInputLayer.add(dWeightHiddenLayer * _weightHiddenLayer[i]);
      }
      var gradientInputLayer = <double>[];
      for (int i = 0; i < hiddenLayer.length; i++) {
        gradientInputLayer.add(hiddenLayer[i] * (1 - hiddenLayer[i]));
      }

      /// delta weight input layer
//    debug.debugger();
      var dWeightInputLayer = <double>[];
      for (int i = 0; i < errorInputLayer.length; i++) {
        dWeightInputLayer.add(errorInputLayer[i] * gradientInputLayer[i]);
      }
//      debug.debugger();
      var newInputWeights = <List<double>>[];
      for (int i = 0; i < _weightsInputLayer.length; i++) {
        var sublist = <double>[];
```

```
      for (int j = 0; j < _weightsInputLayer[i].length; j++) {
        sublist.add(_countNewWeight(_weightsInputLayer[i][j],
            _trainInput[p][j], dWeightInputLayer[i], learningRate));
      }
      newInputWeights.add(sublist);
    }
    _weightsInputLayer.clear();
    _weightsInputLayer.addAll(newInputWeights);
  }
 }
//   predictTest();
}

void predictTest([List<double> input = const [0.0, 0.0, 0.0]]) {
  var hiddenLayer = _transferFunctionV2(input, _weightsInputLayer);
  var outputLayer = _transferFunctionV2(hiddenLayer, [_weightHiddenLayer]);
  print(outputLayer[0]);
  print(outputLayer[0].toInt() == 0 || outputLayer[0] == double.infinity);
}
}
```