Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _комп'ютерної інженерії та управління_____
(повна назва)

Кафедра автоматизації проектування обчислювальної техніки_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _другий (магістерський)_____

_____ГЮІК.501330.035_____

Розпізнавання людських малюнків на основі моделі рекурентної нейронної мережі

(тема)

Виконав: студент 6 курсу, групи СКСм 18-2 ___Хаханов І.В._____
(прізвище, ініціали)

Спеціальність 123_ Комп'ютерна інженерія_____
(код і повна назва спеціальності)
Тип програми _освітньо-професійна__
(освітньо-професійна або освітньо-наукова)
Освітня програма Спеціалізовані комп'ютерні системи_____
(повна назва освітньої програми)

Керівник _професор кафедри АПОТ__ _Литвинова Є.І._____

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління Кафедра автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)
другий (магістерський)

Спеціальність 123_ Комп'ютерна інженерія
(код і повна назва)

Тип програми __освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма _Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____
(підпис)
«____»_____ 2019 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Хаханову Івану Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розпізнавання людських малюнків на основі моделі рекурентної нейронної мережі

затверджена наказом університету від _____ _____20__ р. № _____ 2. Термін подання студентом роботи до екзаменаційної комісії 01 грудня 2019 р.
3. Вихідні дані до роботи: Використовуване програмне забезпечення – платформа Google Cloud Platform; операційна система Ubuntu 16.04_

4. Перелік питань, що потрібно опрацювати в роботі
Вступ; 1. Формулювання мети і задач роботи; 2. Огляд літератури за темою роботи з аналізом існуючих методів вирішення задачі; 3. Розробка кубітних структур даних для методів машинного навчання; 4. Розробка структури процесора синтезу кубітних даних; 5. Послідовний синтез булевих похідних за кубітними покриттями; 6. Опис підходів теоретичного і практичного вирішення задачі розпізнавання образів; 7. Розробка методу та алгоритму розпізнавання людських малюнків; 8. Програмна реалізація методу та алгоритму розпізнавання людських малюнків; 9. Результати практичного застосування отриманих результатів; Висновки.
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Презентація

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів	Примітка
1.	Аналіз завдання	роботи10.09.2019	
2.	Огляд літератури за темою роботи	20.09.2019	
3.	Опис підходів теоретичного і практичного вирішення задач роботи	15.10.2019	
4.	Розробка кубітних структур даних для методів машинного навчання та структури процесора синтезу кубітних даних	20.10.2019	
5.	Послідовний синтез булевих похідних за кубітними покриттями	20.10.2019	
6.	Розробка методу та алгоритму розпізнавання людських малюнків	30.10.2019	
7.	Програмна реалізація методу та алгоритму розпізнавання людських малюнків	20.11.2019	
8.	Результати практичного застосування отриманих результатів	28.11.2019	
9.	Оформлення пояснювальної записки	01.12.2019	

10.	Оформлення презентації	01.12.2019	
11.	Представлення роботи до перевірки в системі Unicheck	09.12.2019	
12.	Представлення роботи до захисту	15.12.2019	

Дата видачі завдання _____ _____20__ р.

Студент _____
(підпис)

Керівник роботи _____ Литвинова Є.І._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Записка пояснювальна: 85 с., 29 рисунків, 25 джерел посилання.

РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА, LSTM, РОЗПІЗНАВАННЯ, ПЕРЕКРЕСТНА ЕНТРОПІЯ, QUICK DRAW, КУБІТНІ СТРУКТУРИ ДАНИХ, ТАБЛИЦЯ ІСТИННОСТІ

Мета роботи полягає у підвищенні точності та зменшенні часу розпізнавання інформації за рахунок використання нейронної мережі, підвищення продуктивності методів тестування і дедуктивного моделювання несправностей за рахунок використання memory-driven архітектур і кубітних структур даних.

Отримано такі основні результати: запропоновано модель відношень порядку в структурах даних і компьютингу, яка формує ефективність паралельного управління алгоритмами обробки великих даних за рахунок суперпозиціонування кінцевої множини дискретних станів; розроблено структурну модель взаємодії кубітних покриттів логічних функцій і похідних компонентів, орієнтованих на синтез і аналіз цифрових систем;

запропоновано tree-driven ATPG процесор і структури даних для обчислення кубітних булевих похідних, представлений двійковим деревом-графом xor-елементів для паралельної обробки частин кубітного покриття; розроблено модель рекурентної нейронної мережі з довгою короткостроковою пам'яттю; розроблено алгоритм роботи нейронної мережі для розпізнавання образів на прикладі розпізнавання людських малюнків; здійснено програмну реалізацію алгоритму роботи нейронної мережі для розпізнавання людських малюнків.

ABSTRACT

Explanatory note: 85 pages, 29 figures, 25 sources.

RECURRENT NEURAL NETWORK, LSTM, PATTERN RECOGNITION, CROSS ENTROPY, QUICK DRAW, QUBIT DATA STRUCTURES, FAULT DETECTION TABLE

The purpose of the work is to increase accuracy and reduce the time of pattern recognition through the use of a neural network, and also increase the productivity of methods for testing and deductive fault simulation through the use of memory-driven architectures and qubit data structures.

The following main results were obtained: a model of order relations in data and computer structures, which forms the efficiency of parallel control of big data processing algorithms through the superposition of a finite set of discrete states; a structural model of the interaction of qubit coverages of logical functions and derivatives of components focused on the synthesis and analysis of digital systems; a tree-driven ATPG processor and data structures for taking qubit Boolean derivatives, represented by a binary graph tree of xor elements for

parallel processing parts of a qubit coverage; a model of a recurrent neural network with a long short-term memory; a neural network algorithm for the recognition of human drawing examples; software implementation of the neural network algorithm for the recognition of human drawings.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

РНН — Рекурентна нейронна мережа ReLU

— Rectified Linear Unit

LSTM — Long Short-term Memory

ШІ — Штучний інтелект

ВСТУП

Компанія Gartner розглядає квантовий комп'ютинг та штучний інтелект як інноваційні технології, в які вкладаються мультиміліардні інвестиції від топ-компаній планети. Квантовий комп'ютинг базується на обчислювачах, що використовують субатомні частинки при створенні структур даних і організації їх фізичної взаємодії для реалізації логічних операцій під час паралельного розв’язання комбінаторних задач [1,2]. Квантовий емуляційний комп'ютинг потребує створення структур даних, моделей, методів і алгоритмів для паралельного розв’язання комбінаторних задач на класичних комп'ютерах шляхом використання додаткової пам'яті. Стратегія створення квантового комп'ютингу полягає в одночасній розробці квантової апаратури і квантових програмних додатків на основі паралельних алгоритмів.

Ефективність квантових обчислень на класичному комп'ютері визначається метрикою паралельного розв'язання комбінаторних задач на кубітних структурах даних за рахунок ускладнення алгоритмів і збільшення пам'яті для зберігання унітарних кодів.

Моделювання на класичних комп'ютерах паралельних квантових алгоритмів дає істотний приріст продуктивності за рахунок використання надлишкової пам'яті для унітарного кодування кубітних структур даних. Моделювання квантових алгоритмів дозволить створити банк паралельних програмних додатків для майбутнього парку ринково доступних квантових комп'ютерів [1,2].

Квантовий комп'ютинг оперує двома базовими технологічними операціями: суперпозиція і переплутування, яким ставляться у

відповідність логічні операції функціонально повного базису: диз'юнкція та інверсія в класичному обчислювачі.

Розпізнавання образів є одніим з найбільш поширених напрямів штучного інтелекту (ШІ). З розвитком технологій, ШІ стає дедалі потужнішими і зараз є багато прикладів алгоритмів для розпізнавання осіб і рухомих об’єктів. Досить згадати результати змагань на базі ImageNet. Але треба зрозуміти, яким чином комп’ютер може визначити що за об'єкт зображений на фотографії або картині, як його можна натренувати розрізняти образи так, як це робить людина.

Для цього можна використовувати нейромережі. Ідея полягає в моделюванні роботи людської нервової системи, а саме, її здатності до навчання і виправлення помилок.

Ідея магістерської атестаційної роботи базується на популярному інтернет сервісі "Quick Draw". Він був запропонований як експериментальна гра, для того щоб показати принцип роботи штучного інтелекту і його потенціал. Користувачеві пропонувалося намалювати об'єкт певної категорії, наприклад, банан, стіл та ін. Протягом часу, було накопичено близько одного мільярду зображень, збережених в базу даних, серед яких 50 мільйонів були завантажені в загальний доступ. На базі існуючої бази зображень розроблено новий алгоритм класифікації малюнків людини.

Мета роботи полягає у підвищенні точності та зменшенні часу розпізнавання інформації за рахунок використання нейронної мережі, підвищення продуктивності методів тестування і дедуктивного моделювання несправностей за рахунок використання memory-driven архітектур і кубітних структур даних.

Задачі роботи:

а) розробити кубітні структури даних для методів машинного навчання;

б) розробити структуру процесору синтезу кубітних даних;

в) здійснити послідовний синтез булевих похідних за кубітними покриттями;

г) виконати обробку похідних кубітів для генерації логічного тесту;

д) розробити алгоритм роботи нейронної мережі для розпізнавання образів на прикладі розпізнавання людських малюнків;

є) здійснити програмну реалізацію алгоритму роботи нейронної мережі для розпізнавання людських малюнків.

Отримані результати опубліковано в роботах [1-6, 12,13].

Автор має у Scopus h-index – 1; кількість документів – 16; цитувань – 12.

Список публікацій автора наведений у додатку Б.

1 КУБІТНІ СТРУКТУРИ ДАНИХ ДЛЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ

Мета розділу – зменшення часу розпізнавання інформації за рахунок використання memory-driven архітектур і кубітних структур даних, а також істотне підвищення продуктивності методів тестування і дедуктивного моделювання несправностей за рахунок алгоритмів паралельного комп'ютингу. Отримані результати опубліковано в роботах [2-6].

1.1 Класифікація квантового комп'ютингу

Інтегрально класифікація квантового комп'ютингу за апаратною, структурною і програмною реалізаціями наведена на рис. 1.1 [1-5]. Вона узагальнює світові розробки вчених університетів і провідних компаній планети в частині фізичних основ квантового комп'ютингу, логіки його роботи, а також вже створені програмні продукти і компіляторисимулятори, орієнтовані на рішення комбінаторних, оптимізаційних, криптографічних та інтелектуальних проблем.

Практично, всі провідні компанії планети з NASDAQ-списку вважають за необхідне вкладати інвестиції в розробку і створення власних квантових комп'ютерів, а також програмних продуктів для їх обслуговування. На стадії проведення експериментів переваги окремих компаній є практично незначними. На ринку квантових обчислень, що виникає, швидше за все переможе гравець, який інвестує максимальну кількість фінансів в memory-driven quantum computing.

Вже сьогодні можна виділити три комерційно життєздатних застосувань для ранніх квантових обчислювальних пристроїв – квантове моделювання, оптимізацію і комбінаторику для областей: охорони здоров'я, штучного інтелекту, кіберзахисту даних і хмарних сервісів, криптографії, транспорту, розпізнавання даних. Поки невідомо, чи зможуть існуючі (послідовні) алгоритми та програми збільшити свою продуктивність, використовуючи квантові процесори, які скоро будуть доступні споживачам. Тому сьогодні слід розробляти нові паралельні типи алгоритмів квантового моделювання, криптоаналізу, глибокого машинного навчання, оптимізації шляхом використання класичних, а також квантових процесорів Google, IBM, IonQ, доступних за допомогою хмарних сервісів [1].

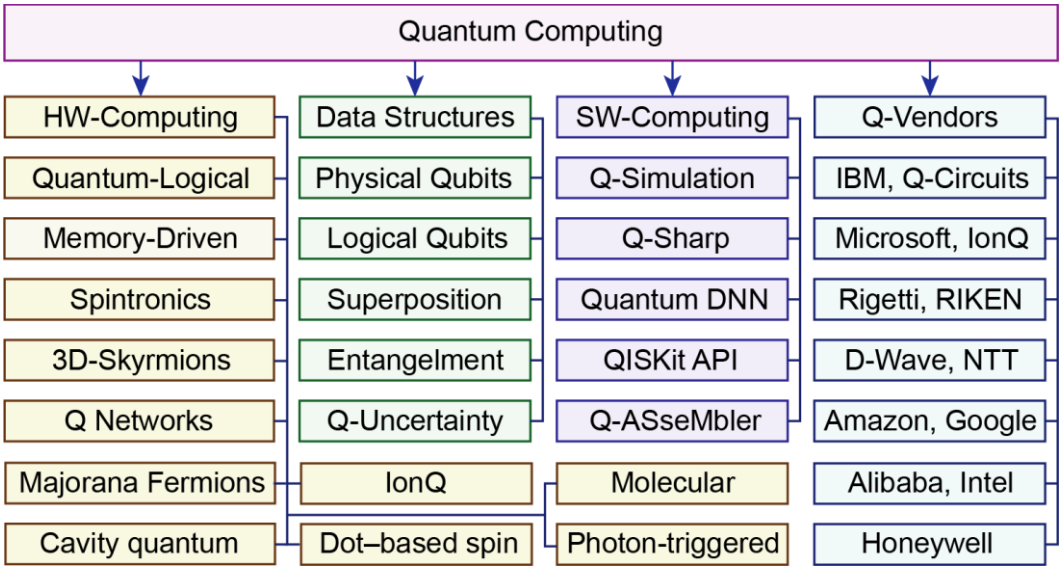
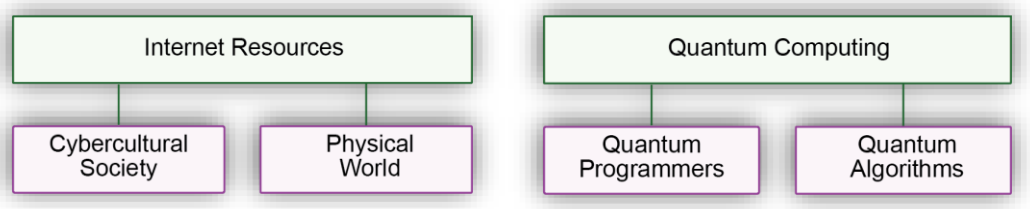


Рисунок 1.1 – Класифікація квантового комп'ютингу

Кіберкультура – технологічна досконалість і соціальна значущість моральних відношень між розумним суспільством, фізичним світом і кіберпростором, яке визначається використанням інтернет-сервісів для цифрового моніторингу та надійного метричного управління процесами в усіх сферах людської діяльності, у тому числі науці, освіті, охороні здоров'я, виробництві і транспорті, з метою поліпшення якості життя людей і збереження екосистеми планети (рис. 1.2, а) [1-5].

Квантова кіберкультура – технологічна та моральна досконалість у віднощеннях між парком ринково доступних квантових комп'ютерів, потужним банком квантових паралельних алгоритмів-додатків і досвіченим, паралельно мислячим, суспільством, спрямована на ефективне вирішення комбінаторних завдач (рис. 1.2, б).



а б

Рисунок 1.2 – Кіберкультура (а) і квантова культура (б) відношень

Кіберкультура квантового memory-driven комп'ютингу інтегрує в трикутну структуру технології паралельного рішення часовитратних комбінаторних задач, ринково доступні квантові обчислювачі і паралельно мислячих програмістів. Масштабована карта досліджень, відповідних квантовій кіберкультурі, що націлена на створення паралельних алгоритмів, зокрема, для SoC Design and Test, представлена на рис. 1.3.

Метрика квантового та класичного комп'ютингу не має суттєвих структурних відмінностей. Природно, існує аналогія між квантовими операціями на кубітних структурах даних і теоретико-множинними операціями на символах алгебри Кантора. Ізоморфізм двох структур: квантової логіки і алгебри множин полягає в подобі носіїв і сигнатур.

Взаємно-однозначна відповідність носіїв визначається відношеннями між булеаном і кубітними структурами даних [1]:

Boolean A^k =	0	1	X =0È 1	È =0È 1
Qubit | yÈ=	|0È	|1È	a |0È+b|1È	a |0È|b|1È

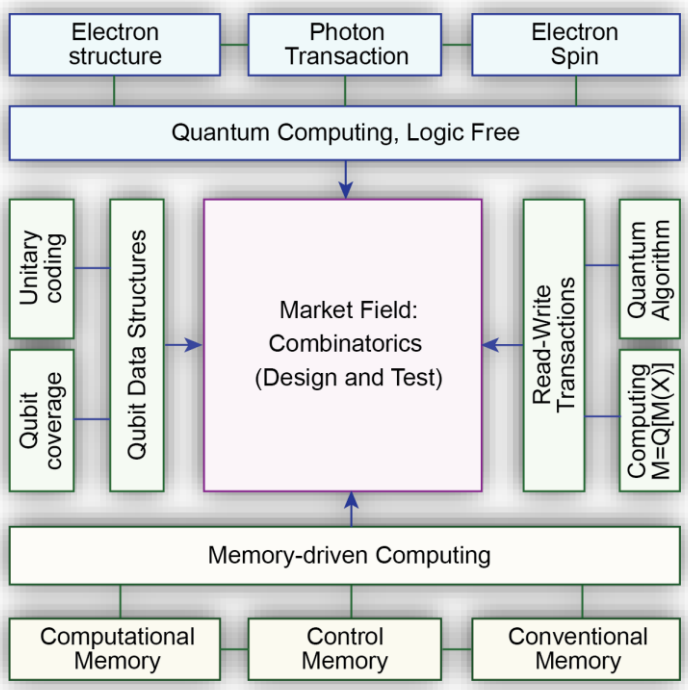


Рисунок 1.3 – Roadmap of quantum design and test memory-driven computing

Ізоморфізм сигнатур визначається взаємно-однозначною відповідністю між двома операціями: об'єднання – суперпозиція, доповнення – переплутування. Для операції перетину відповідний аналог у квантовій логіці не визначений. При цьому унітарне кодування символів алфавіту Кантора дає можливість паралельно виконувати будь-які логічні операції навіть на класичному комп'ютері. Але дізрапторне інноваційне рішення полягає у відмові від операцій суперпозиції і змішування в сторону створення memory-driven квантового комп'ютингу, що використовує тільки операції запису-зчитування на пам'яті, де виконавчі стани реалізовані, наприклад, у спінових моментах електронів.

Моделювання на класичних комп'ютерах паралельних квантових алгоритмів дає істотний приріст продуктивності за рахунок використання надлишкової пам'яті для унітарного кодування кубітних структур даних.

Більш того, моделювання квантових алгоритмів стратегічно покликане вирішувати проблему створення квантового інтелекту планети шляхом розробки банку паралельних програмних додатків для майбутнього парку ринково доступних квантових комп'ютерів.

Проектування і тестування є найбільш передовою галуззю діяльності вчених і компаній, спрямованою на створення нових технологій комп'ютингу за метрикою: швидкодія, енергозбереження, компактність і надійність. Параметр time-to-market при створенні комп'ютингової продукції, поряд з якістю, є домінуючим. Тому квантові паралельні методи та алгоритми проектування, тестування, верифікації, моделювання і діагностування виробів на кубітних структурах даних є актуальними при їх імплементації в сучасні класичні і в майбутні квантові комп'ютери, з метою істотного зменшення time-to-market [1].

Одним з основних досягнень кубітного покриття є паралелізм виконання логічних операцій над елементами бітових векторів. Метод кубітного або квантового моделювання використовує одне характеристичне рівняння, що оперує при транзакціях даних тільки адресами кубітного покриття і вектора моделювання, який створює схемну структуру. Секвенсор синтезу тестів, що використовує логічні операції: інверсії, зсуву, порівняння і об'єднання, є прототипом апаратної реалізації модуля, вбудованого в інфраструктуру SoC, для online-тестування функціональностей. Такий апаратний модуль може бути доступний, як хмарний сервіс з IP-адресою, який створює прямий зв'язок з будь-якою функціональністю в масштабах кіберпростору планети.

1.2 Процесор синтезу кубітних даних

Запропоновано метрику математичних і технологічних відношень в структурах даних, на яких будуються ефективні алгоритми і методи управління або обробки даних для збільшення їх продуктивності. Введено векторну модель або форму булевих похідних, яка використовується для синтезу дедуктивних матриць в кубітному методі моделювання несправностей і оцінки якості тестових послідовностей. Пропонується tree-driven ATPG processor та структури даних для обчислення кубітних булевих похідних, представлений двійковим деревом-графом хогелементів для паралельної обробки частин кубітного покриття.

1.2.1 State of the Art

Продуктивність комп'ютингу визначається метрикою математичних і технологічних відношень в структурах даних, на яких будуються ефективні алгоритми і методи управління або обробки даних для досягнення поставлених цілей. Структури даних квантового комп'ютингу відрізняються від класичних обчислень суперпозиційним відношенням між нулем і одиницею, позиціонування яких визначено в одній точці гильбертова простору [1,6]. Технологічно, суперпозиція формується спинами електронів, які також позиціонуються в одній точці міжатомного простору. Саме суперпозиція нуля й одиниці в одній точці простору, яка може бути поширена на кінцеву кількість дискретних станів, є першопричиною відношень в організації структур даних для реалізації паралельних методів та алгоритмів. Що стосується класичного комп'ютингу, то властивість суперпозиції, але не в одній точці, а розподілене у просторі, можна і потрібно використовувати для створення паралельних ефективних алгоритмів обробки великих даних,

представлених в унітарних кодах, які дають можливість здійснювати суперпозицію кінцевої кількості дискретних станів. Важливий висновок, який випливає з наведених міркувань, полягає в первинності відношень на структурах даних для синтезу паралельних ефективних, але вторинних, алгоритмів управління або обробки даних. Будь-яке проектування нового виробу повинно мати на увазі наступне відношення порядку між категоріями: а) мета; б) відношення; в) управління; г) архітектура; д) спостережуваність; є) дані (ресурси).

Інтерес вчених і практиків до нових рішень проблем тестування обчислювальних пристроїв ілюструється кількістю публікацій в бібліотеці IEEE Xplore. Так, наприклад, контентний пошук за запитом (Fault Simulation) дає 37892 робіт. У той же час високопродуктивний метод дедуктивного аналізу має всього 51 посилання на наукові публікації. Проте в бібліотеці представлено 290 публікацій, присвячених Quantum Fault Simulation, у тому числі дедуктивні алгоритми. Інший приклад пов'язаний з кількістю робіт в області синтезу і генерації тестів, які налічують 20408 і 64691 публікацій відповідно. При цьому кількість кубітних або квантових моделей і методів генерації (1020) і синтезу (264) тестів істотно зросла за останні 5 років. Такий інтерес вчених і промисловості пов'язаний з підвищенням якості комп'ютингу в усіх сферах людської діяльності: дата-центри, соціальні мережі, хмарні обчислення, транспорт, енергетика, медицина, будівництво, космонавтика, озброєння, управління процесами. Що стосується аналізу несправностей, тут картина технічної діагностики може бути представлена відношеннями між структурами даних, моделями справної поведінки, несправностями та методами синтезу та аналізу тестів, зображеними на рис. 1.4. Публікації останніх років відображають появу паралельних моделей і методів синтезу та аналізу, забезпечених штучним

інтелектом, хмарними сервісами і квантовими структурами даних [7-10]. Оборотні логічні схеми являють собою з'єднання декількох спеціальних типів квантових k-CNOT вентилів, для яких використовуються класичні алгоритми синтезу тестових наборів для виявлення всіх поодиноких пропущених несправностей в оборотній схемі на основі вентилів k-CNOT за допомогою методу булевих різниць [10].

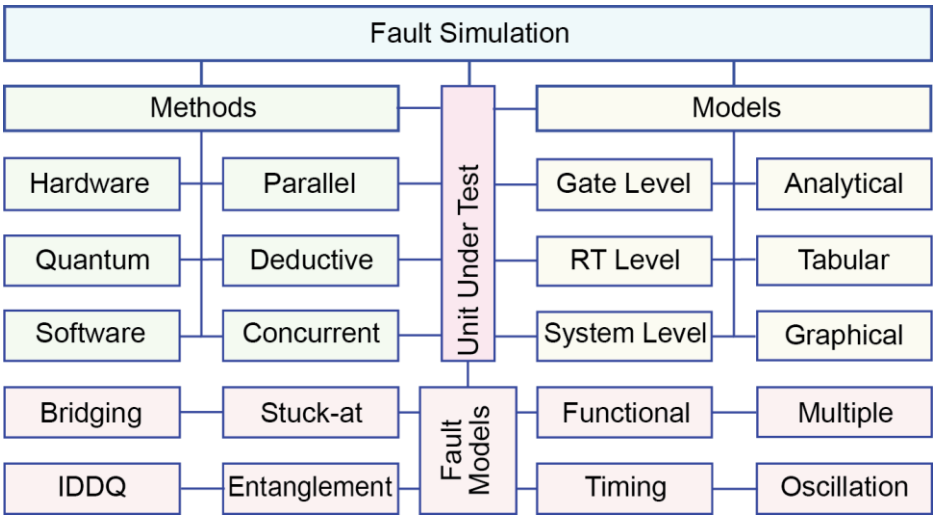


Рисунок 1.4 - Моделі та методи аналізу дефектів

Методи синтезу тестів для логічних функціональностей традиційно використовують три форми явного опису функціональної поведінки: табличну (TT – Truth Table), аналітичну (DF – Disjunctive Form), графову (альтернативні графи Раймунда Убара або виконавчі діаграми рішень [11]. Перша технологічна для комп'ютерної обробки, але витратна за часом, пам'яттю і продуктивністю обробки таблиць. Друга – компактна за формою, але вимагає створення потужних обчислювачів для аналізу або вирішення булевих рівнянь. Третя – наочна для людини, компактна для компьютера, але потребує спеціалізованих обчислювачів для синтезу та аналізу складних систем на основі альтернативних графів [11].

Далі пропонується кубітне покриття [1], як векторне подання стану виходів таблиці істинності з неявним описом вхідних впливів у вигляді адрес координат двійкового вектора вихідних значень. Порівняльний аналіз форм завдання логічної функції від трьох змінних, включаючи кубітний вектор або Q-покриття (QC – Qubit Coverage), наведений нижче.

Y(TT)=;

X1	X2	X3	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Y(DF)=X1X2X3ÚX1X2X3ÚX1X2X3ÚX1X2X3;

Y(QC)=01101001.

Q-вектор є більш компактною формою порівняно з таблицею істинності, він має при цьому всі переваги таблиці, пов'язані з технологічністю обробки для синтезу та аналізу логічних функцій. Qвектор потребує в n раз меншого обсягу пам'яті для зберігання даних порівняно з таблицею істинності від n змінних. Q-вектор не потребує (n**2)-складних обчислювальних процедур, необхідних для визначення стану виходу логічної функції за диз'юнктивною нормальною формою або узагальненою таблиціею істинності. Для цього необхідна лише одна автоматна операція, яка використовує адресні операції запису-зчитування: Mi = Qi[M(Xi)], що характеризуються паралелізмом і лінійною обчислювальною складністю.

1.2.2 Послідовний синтез булевих похідних за кубітними покриттями

В основу синтезу тестів на кубітних покриттях покладена технологія визначення булевих похідних, які створюють активізацію логічних шляхів від входів до виходів схемної структури. Для цього використовується логічна xor-операція між симетричними частинами кубітного вектора [2]:

$$Q'(Xk) = \{QiL, QiR\} = QLi \; i=1,Å2^{k-1}QiR. \quad k=1,n$$

Рівнянню можна поставити у відповідність найпростіший секвенсор взяття кубітной похідної (рис. 1.5), де xor-операція формує обидві частини вектора результату.

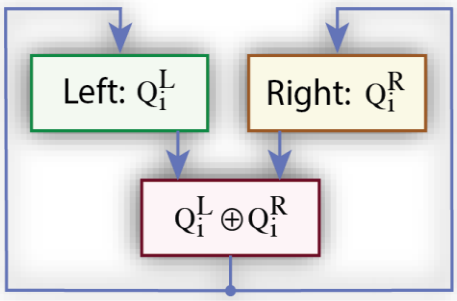


Рисунок 1.5 - Секвенсор кубітної похідної

Використання секвенсора для взяття трьох кубітних похідних для логічної функції від трьох змінних Q = (01 10 10 01) представлено в такому вигляді:

Q (X1) = (11 11 11 11), Q (X2) = (11 11 11 11), Q (X3) = (11 11 11 11).

Перший вектор кубітної похідної за змінною X1 сформований як xor-сума сусідніх комірок, другий вектор кубітної похідної за змінлою X2 створений як xor-сума сусідніх пар координат, третій вектор кубітної похідної за X3 синтезований як xor-сума сусідніх четвірок кубітного вектора. Цікаві наслідки таких X-функцій. Якщо кубітні похідні за всіма змінними логічної функції дорівнюють одиничному вектору, то: а) дедуктивна формула для моделювання несправностей инвариантна до вхідних тестових наборів або не залежить від них; б) будь-який тествектор перевіряє всі константні поодинокі несправності, інверсні до станів ліній; в) кількість логічних функцій від n змінних, де виконується умова

$$\text{"n éé df =1éé,}$$
$$\text{i=1édXi é}$$

завжди дорівнює двом; г) для активізації вхідної змінної Xi не потрібно ніяких логічних умов до інших входів; д) X-функція є логічною простою функцією від кінцевої кількості змінних (n = 1,2,3, ...), яку неможливо мінімізувати; є) довжина тесту перевірки поодиноких константних несправностей всіх ліній для X-функції від n змінних завжди дорівнює

$$Q = 1 + \_1 \acute{} 2^{2n}.$$
$$2$$

Наступні таблиці представляють процеси: синтез тестів (T) за кубітним покриттям Q = (01 10 10 01), моделювання несправностей (D),

мімізація тестових послідовностей (M) та отримання кубітного мінімального тесту цифрової структури – T (Q)):

T	1 2 3 4 5 6 7 8
0	0 0 0 0 0 0 0 0
1	0 1 1 0 0 0 1 0 1
2	0 0 1 0 0 1 0 1 1
3	0 0 0 0 0 1 0 0 0
4	0 1 0 1 1 0 1 0 0
5	0 0 0 1 1 0 0 0 0
6	0 0 1 1 1 0 0 0 1
7	1

D	1 2 3 4 5 6 7 8
0	1 1 1 1 1 1 1 1 1
1	1 0 0 . . . 0
2	1 0 1 . 0 . . 0 1 0
3	0 1 1 1 1 1 0 1 1 .
4	. 0 . 0 0 1 0 1 1 1
5	1 1 0 0 1 1 1 1 1
6	1 0 0 0 . . . 0 0
7	

M	1 2 3 4 5 6 7 8
0	1 1 1 1 1 1 1 1
1	1 1 0 0 . . . 0
2	1 0 1 . 0 . . 0
4	0 1 1 . . 0 . 0
7	0 0 0 . . . 0 0
C	x x x x x x x x

®®

T(Q)
1
1
1
0
1
0
0
1

Стовпець T (Q) = 11101001 створює мінімальну кубітну форму тесту – двійкових адрес для одиничних координат, які необхідно подати на зовнішні входи, щоб перевірити всі поодинокі константні несправності зовнішніх та внутрішніх ліній логічної схеми.

1.2.3 Обробка похідних кубітів для генерації логічного тесту

Характеристичне рівняння синтезу тестів використовує операції: інверсії, зустрічного зсуву даних, xor-підсумовування і диз'юнкції на кубітному покритті [1]:

$$T(S)=\text{Å}_{j=1}^{n}\left[Q\ \text{Å}_{j=1}^{\underline{n}}\ Sj(Q)\right].$$

Обчислювальна складність алгоритму синтезу тестів за допомогою даного рівняння дорівнює

$$Q = 2^n + n \times 2^n + n \times 2^n + n \times 2^n = 2^n + 3(n \times 2^n) = 2^n(1 + 3n).$$

Апарат кубітних булевих похідних істотно спрощує алгоритм генерування тестових послідовностей для логічних функцій за рахунок зменшення до двох операцій (взяття похідної за кожною змінною і диз'юнкції похідних, яка створює кубітну форму тесту) [12,13]:

$$T(Q') = \bigcup_{i=1}^{n} Q'(Xi)$$

Булева похідна на кубітному векторі зводиться до виконання хогоперації над частинами кубітного покриття, розмірність яких визначається ступенем двійки від номера даної змінної, що змінюється від 1 до n.

Кубітне покриття описує поведінку логічної функції від чотирьох змінних: Q (X) = (+10000000000000001), для якої розглядається взяття похідних з метою синтезу тестів.

Характеристичне рівняння взяття булевої похідної оперує частинами кубітного вектора-покриття, розмірність якого прив'язана до ступеня двійки від кількості змінних. Далі розглядається алгоритм взяття кубітної похідної для функції від 4-х змінних.

1. На першому кроці береться похідна за змінною X_1: Розглядаються дві рівні частини двійкового Q-вектора, розмірністю 2^n, до яких застосовується паралельна (покоординатна) xor-операція: $Q_0 \oplus Q_1 \longrightarrow$

$\{Q_0, Q_1\}$, після виконання якої результат заноситься в обидві частини Qвектора. Інакше дана процедура може бути записана як: $Q_0 = Q_0 \oplus Q_1$, $Q_1 = Q_0 \oplus Q_1$ або в компактному вигляді: $\{Q_0, Q_1\} = Q_0 \oplus Q_1$.

2. На другому кроці береться похідна по змінною X_2: розглядаються вже чотири рівні частини кубітного Q-вектора, до яких застосовуються попарно дві xor-операції, результат яких заноситься в операнди: $\{Q_0, Q_1\} = Q_0 \oplus Q_1$; $\{Q_2, Q_3\} = Q_2 \oplus Q_3$.

3. На третьому кроці береться похідна за змінною X_3: розглядаються вже вісім рівних частин кубітного Q-вектора, до яких застосовуються попарно чотири xor-операції, результат яких заноситься в операнди:

$\{Q_0, Q_1\} = Q_0 \oplus Q_1$; $\{Q_2, Q_3\} = Q_2 \oplus Q_3$.

$\{Q_4, Q_5\} = Q_4 \oplus Q_5$; $\{Q_6, Q_7\} = Q_6 \oplus Q_7$.

4. На четвертому кроці береться похідна за змінною X_4 : розглядаються вже 16 рівних частин кубітного Q-вектора, до яких застосовуються попарно вісім xor-операцій, результат яких заноситься в операнди:

$\{Q_0, Q_1\} = Q_0 \oplus Q_1$; $\{Q_2, Q_3\} = Q_2 \oplus Q_3$; $\{Q_4, Q_5\} = Q_4 \oplus Q_5$; $\{Q_6, Q_7\} = Q_6 \oplus Q_7$;

$\{Q_8, Q_9\} = Q_8 \oplus Q_9$; $\{Q_{10}, Q_{11}\} = Q_{10} \oplus Q_{11}$; $\{Q_{12}, Q_{13}\} = Q_{12} \oplus Q_{13}$;

$\{Q_{14}, Q_{15}\} = Q_{14} \oplus Q_{15}$.

Структура процесора для взяття булевих похідних за кубітним покриттям представлена на рис. 1.6 двійковим деревом-графом представлення Q-вектора функції, де вершинами є xor-перетворювачі, а дугами виступають операнди або частини кубітного покриття.

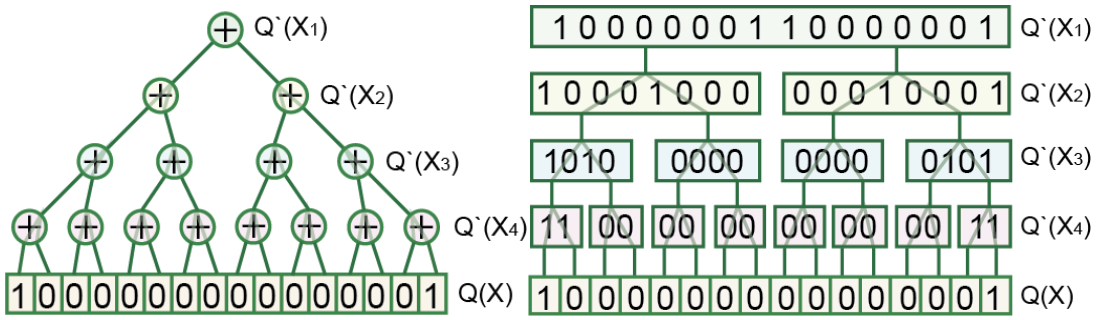


Рисунок 1.6 – ATPG процесор і структури даних

Права частина рис. 1.6 демонструє структури даних, що перетворюються за допомогою xor-операцій процесора. Особливістю останнього є паралельне виконання всіх операцій над даними, представленими кубітним покриттям логічної функціональності. Тому xor-елементи в графі і відповідні регістри в структурах даних мають наскрізний зв'язок до кожного рівня від кубітного покриття логічної функції, розташованого в нижньому ряду обох структур. Існують цікаві паралельні рішення аналізу цифрових пристроїв для синтезу тестів. Однак, аналогів обчислювальних архітектур за продуктивністю взяття булевих похідних за один автоматний такт не існує.

Рис. 1.7 ілюструє взяття чотирьох булевих похідних на основі використання запропонованого процесора, які далі об'єднуються в кубітний тест.

Derivative-based Test Pattern Generation

Q(X)	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Q'(X1)	1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1
Q'(X2)	1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1
Q'(X3)	1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1
Q'(X4)	1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1
$T(Q') = \bigcup_{i=1}^{n} Q'(Xi)$	1 1 1 0 1 0 0 1 1 0 0 1 0 1 1 1

Truth Table

Q(X)	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
X1	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
X2	0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
X3	0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
X4	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

Рисунок 1.7 – Таблиця істинності та булеві похідні

Таблиця істинності наведена для зручності запису аналітичної форми завдання похідних за станами змінних, заданих в Truth Table. З огляду на сказане, аналітична діз'юнктивна нормальна форма похідних за чотирма змінними: $Q'(X_1) = (1000000110000001)$, $Q'(X_2) = (1000100000010001)$, $Q'(X_3) = (1010000000000101)$, $Q'(X_4) = (1100000000000011)$ перетвориться до виду:

$$Q'(X_4) = \bar{X}_1\bar{X}_2\bar{X}_3\bar{X}_4 \vee \bar{X}_1\bar{X}_2X_3X_4 \vee X_1X_2X_3\bar{X}_4 \vee X_1X_2X_3X_4 = \bar{X}_1\bar{X}_2\bar{X}_3 \vee X_1X_2X_3.$$

$$Q'(X_3) = \bar{X}_1\bar{X}_2\bar{X}_3\bar{X}_4 \vee \bar{X}_1\bar{X}_2X_3\bar{X}_4 \vee X_1X_2\bar{X}_3X_4 \vee X_1X_2X_3X_4 = \bar{X}_1\bar{X}_2\bar{X}_4 \vee X_1X_2X_4.$$

$$Q'(X_2) = \bar{X}_1\bar{X}_2\bar{X}_3\bar{X}_4 \vee \bar{X}_1X_2\bar{X}_3\bar{X}_4 \vee X_1\bar{X}_2X_3X_4 \vee X_1X_2X_3X_4 = \bar{X}_1\bar{X}_3\bar{X}_4 \vee X_1X_3X_4.$$

$$Q'(X_1) = \bar{X}_1\bar{X}_2\bar{X}_3\bar{X}_4 \vee \bar{X}_1X_2X_3X_4 \vee X_1\bar{X}_2X_3\bar{X}_4 \vee X_1X_2X_3X_4 = \bar{X}_2\bar{X}_3\bar{X}_4 \vee X_2X_3X_4.$$

Аналітична форма отриманого результату не має ліній, за якими береться похідна, оскільки рівняння показують двійкові умови активізації змінних на поєднаннях інших ліній, які формують два логічних шляхи від кожного входу до виходу, що є основою синтезу тестів перевірки константних несправностей. Узагальнюючи сказане вище, зміна кожної змінної в Q-похідних гарантує перевірку всіх поодиноких константних несправностей на вхідних внутрішніх і вихідних лініях за всіма можливими логічними шляхами активізації. Маючи в наявності кубітне

покриття як завгодно складної булевої функції, кубітний метод синтезу тестів зводиться до однієї паралельної xor-операції синтезу похідних над частинами Q-вектора, що дає можливість отримати тест для поодиноких константних несправностей за n-1 автоматний такт шляхом об'єднання векторів отриманих похідних.

Обчислювальна складність базового алгоритму синтезу тестів на основі взяття кубітних похідних дорівнює $Q = n \times 2^n + n \times 2^n = Q = 2(n \times 2^n)$. При використанні процесора складність алгоритму синтезу тестів зменшується до n автоматних тактів $Q = 1 + (n - 1) = n$.

Що стосується синтезу тестів для X-функцій (xor, not-xor), то тут взагалі не потрібно проводити обчислення. Необхідно тільки записати тест довжиною

$$Q = 1 + \frac{1}{2} \cdot 2^{2n}.$$

за Q-покриттям відповідно до наступних правил – кубітний тест дорівнює кубітному покриттю X-функції, де будь-яка 0-координата додатково замінена на середнє арифметичне значення:

$$T = T^1 \acute{U} T_i^0,$$

$$T^1 = " \, Ti : f(Ti) = 1 \, T_i^0 \, \hat{I}$$

$$T^0 = " \, Ti : f(Ti) = 0;$$

$$T(01101001) = (001\acute{U}010\acute{U}100\acute{U}111)\acute{U}000;$$
$$T(10010110) = (000\acute{U}011\acute{U}101\acute{U}110)\acute{U}001.$$

Таким чином, тестами для кубітних покриттів двох булевих Хфункцій Q1 = 01101001 та Q2 = 10010110 є вектори: T1 = 11101001 та T2 = 11010110.

1.3 Структури кубітних даних для синтезу та аналізу цифрових систем

Структури кубітних даних для синтезу та аналізу цифрових систем і компонентів оперують такою моделлю опису логічних схем:

$$S=\{M,X,Y,Q\},$$
$$M=(M1,M2,...,Mi,...,Mm),$$
$$X=(X1,X2,...,Xi,...,Xn),$$
$$Y=(Y1,Y2,...,Yi,...,Yk),$$
$$Q=(Q1,Q2,...,Qi,...,Qq),\ Mi=Qi[M(Xi)].$$

Тут представлені такі системні компоненти: M – вектор моделювання цифрового пристрою, який пов'язує всі кубітні покриття примітивних елементів в структуру; X – вектор вхідних змінних, заданих двійковими значеннями; Y – вектор вихідних змінних, які формують реакцію цифрового пристрою; Q – кубітне покриття, представлене у вигляді вектора вихідних станів логічного елемента і призначене для формування його функції. Основне характеристичне рівняння для моделювання цифрового пристрою оперує обчисленням адрес для записузчитування даних, що створює прості і швидкодіючі транзакції між вектором моделювання і кубітнимі покриттями:

$$Mi=Qi[M(Xi)].$$

Щоб визначити двійкове значення логічної змінної або лінії Mi, необхідно сформувати адресу комірки кубітного покриття, який створюється конкатенацией двійкових станів вектора моделювання M, де адреси комірок вектора задаються номерами-ідентифікаторами вхідних змінних X. Характеристичне рівняння безпосередньо впливає на швидкодію квантового методу моделювання, яке залежить від операцій конкатенації k, зчитування r і записи w бітів, кількості q кубітних покриттів в цифровій схемі або логічних примітивів, а також довжини тесту (вхідних наборів) t:

$$Q=(k+r+w)´ q´ t.$$

Таким чином, всього три транзакційних операції необхідні для обробки логічного елемента будь-якої функціональної складності.

Сучасна стратегія компьютингового бізнесу полягає в масовому переведенні всіх процесів і явищ з чисто фізичного в кібер-фізичний простір [1]. Це робить сервіси менш уразливими для кіберзлочинності, економічно вигідними і технологічно доступними в просторі і в часі, без будь-яких обмежень. Тому високотехнологічний бізнес Design and Test, який визначається компаніями, що формують індекс капіталізації NASDAQ, посиленими темпами йде в хмарний кіберфізичний комп'ютинг, який має наступну загальну структуру, представлену на рис. 1.8.

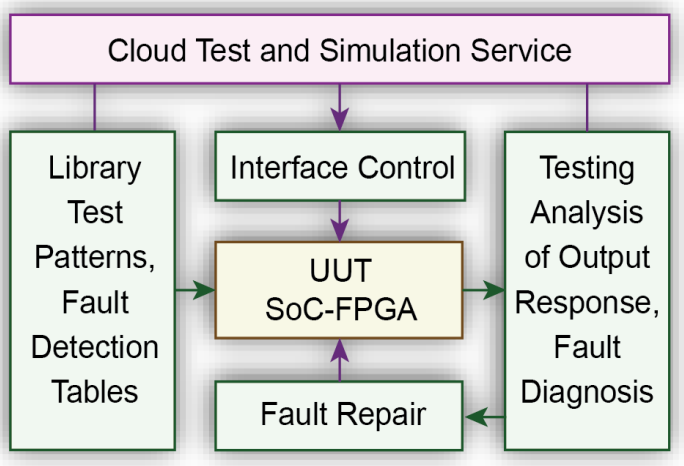


Рисунок 1.8 – Загальна структура сервісу Desing and Test

Тут є тільки один фізичний модуль з IP-адресою, Unit Under Test (будь-які цифрові пристрої, сенсори, актюатори, мобільні пристрої, автомобільні комп'ютери, кінцеві гаджети), який знаходиться у фізичному просторі. Решта блоків, зазначених в структурі, створюють хмарні сервіси, доступні в режимі онлайн 24/7. Таким чином, будь-який фізичний пристрій може бути легко продіагностований завдяки його підключенню до хмарних сервісів в режимі автономного тестування або навіть в режимі робочого функціонування.

Реалізація квантового моделювання в якості хмарного сервісу [3-5] має ряд переваг перед програмним додатком: а) інваріантність до апаратних обчислювальних засобів, як до основи, на якій функціонує сервіс, що швидко застаріває; б) відсутність необхідності купівлі апаратних засобів для реалізації програмного додатка як сервісу; в) низький рівень фінансових витрат для оренди хмарного простору, як платформи для реалізації компьютингового сервісу; г) інваріантність місця знаходження розробника (-ів) для швидкої імплементації хмарного сервісу; д) інвариантность хмарного сервісу до географічних координат всіх користувачів, що знаходяться на планеті; є) доступність хмарного сервісу з будь-якої точки земної кулі в часі і просторі; ж) високий рівень

надійності дата центрів, які формують хмарно-орієнтовані платформи від компаній: Google, Amazon, Microsoft; з) високий рівень software-driven кібербезпеки хмарних сервісів, що надаються згаданими платформами; і) швидка технологічна масштабованість хмарних сервісів, орієнтована на розширення фунціональних можливостей і велику кількість реальних користувачів.

1.4 Архітектура кубітно-логічного процесора

Таблиці істинності або кубітні покриття для опису логічних елементів є ефективними структурами даних для вирішення проблем комп'ютингу і пошуку необхідних даних [1-6,12,13]. Автоматичний синтез кубітних покриттів функциональностей є однією з основних важко формалізованих завдань, без якої неможливо здійснювати аналітику великих даних. Для цих цілей запропоновано аналітичну модель W кубітно-логічного процесора, яка оперує двома матрицями: універсумів U примітивів і кубітних функциональностей Q, а також логічними примітивами L:

$$
\begin{gathered}
W = (U,Q,L), \\
U = (U_1, U_2, ..., U_i, ..., U_n); \\
\bigcup_{i-1}^{n} U_i = U;\ U_i \bigcap_{i,k=1,n}^{i\neq k} U_k = \varnothing; \\
Q = (Q_1, Q_2, ..., Q_i, ..., Q_n); \\
\bigcup_{i-1}^{n} Q_i = Q;\ Q_i \bigcap_{i,k=1,n}^{i\neq k} Q_k = \varnothing; \\
Q_i = (Q_{i1}, Q_{i2}, ..., Q_{ij}, ..., Q_{im});\ Q = [Q_{ij}]; \\
U_i = (U_{i1}, U_{i2}, ..., U_{ij}, ..., U_{im});\ U = [U_{ij}]; \\
L = f[Q] = (Q_1 \circ Q_2 \circ, ..., \circ Q_i \circ, ..., \circ Q_n) \\
\circ = \{\wedge, \vee, \oplus\}; \\
U_{ij} \in U_i \in U;\ Q_{ij} \in Q_i \in Q;\ Q_i \in U_i;\ Q \in U; \\
Q_{ij} = 1 \leftarrow \max \mu(R, U_{ij}).
\end{gathered}
$$

Метрика-універсум U тут виконує роль еталонного зразка для порівняльного аналізу еталона з вхідним потоком даних R=X, що реалізується за допомогою аналізатора-компаратора, який видає максимальне значення функції приналежності, яке, у свою чергу, трансформується в одиницю на відповідній координаті одного з кубітів

$$Qij = 1 \neg \max m(R,Uij).$$

Архітектура метричної взаємодії U-матриці універсумів з потоком вхідних великих даних R для обчислення функцій приналежності □(R, U), з метою отримання Q-матриці значень і подальшого L-об'єднання кубітів в комбінаційну схему кубітно-логічного процесора, наведена на рис. 1.9.

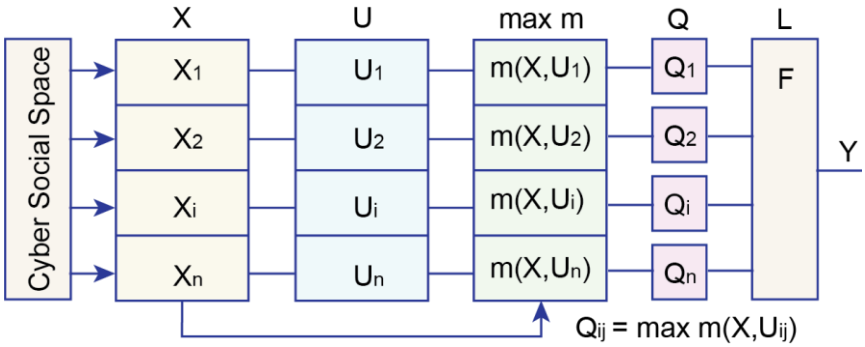


Рисунок 1.9 – Архітектура для синтезу кубітно-логічного процесора

Тут вхідний потік модельованих великих даних R має такий же формат, як U-, Q-матриці і комбінаційна схема кубітно-логічного процесора. Алгоритм синтезу Q-матриці полягає у визначенні максимального значення функції приналежності вхідного фрейму розглянутої змінної до одного зі значень відповідного рядка матриці універсумів. В результаті такого порівняння по всіх координатах Uматриці формуються поодинокі координати кубітної матриці, де кожен рядок являє

собою примітивну функціональність відповідно розглянутої змінної. Разом всі рядки Q-матриці створюють комбінаційну схему кубітно-логічного процесора для моделювання будь-якого вхідного впливу з метою визначення його приналежності до даного еталону кіберфізичного процесу або явища.

Процедура синтезу кубітів необхідна для аналізу потоків даних шляхом використання логічних еталонів. Фрагменти даних надходять на входи одного або декількох логічних елементів, що формують метрику процесу:

$$Uij \hat{I} Ui \hat{I} U;\ Qij \hat{I} Qi \hat{I} Q;\ Qi \hat{I} Ui;\ Q \hat{I} U;$$

В результаті моделювання вхідного потоку великих даних формуються бінарні значення еталона в кубітному векторі кожного логічного елемента, відповідного одному параметру. Для цього використовується метричний вимір функції приналежності вербальних даних до кожного значення наперед заданого універсуму примітивів логічної змінної. Так автоматично створюються кубітні зразки функціональності.

Формування повної множини параметрів-змінних процесу або явища також пов'язано з аналітикою великих даних, спрямованою на отримання ключових понять-слів, максимально віддалених один від одного за метрикою (кодовою відстанню) класів еквівалентності і покривають всі змінні процесу або явища. Слід зауважити, що універсум примітивів ототожнюється з класом непересічних еквівалентностей, які створюють всі можливі значення даної змінної-класу в той час, як множина еквівалентних класів відповідна універсуму змінних вищого рівня ієрархії.

Дані властивості використовуються при аналітичному синтезі універсуму змінних, що покривають цифровий образ процесу або явища гранями, які формують еталон-функціональності.

Після синтезу кубітних векторів за всіма параметрами в Q-матриці всі значення виходів кубітних елементів надходять на входи інтегратора L, що працює за функцією and (може бути й інша функція, наприклад, notand), який видає два значення {1,0}: позитивний або негативний результат моделювання, який можна інтерпретувати як бінарну функцію приналежності до еталону-функціональності.

Таким чином, logic-процесор, синтезований на основі використання Q-матриць квантових структур даних здатний online моделювати будь-які процеси і явища, недоступні сьогодні для класичного комп'ютингу в базисі традиційних логічних елементів.

Формалізм створення еталон-схеми для соціального процесу або явища полягає у визначенні кількості істотних параметрів, де всередині кожного з них генерується множина значень, унітарно кодованих для синтезу кубітного вектора логічного елемента. Логічні примітиви, відповідні істотним параметрам, об'єднуються за функціями (and, or, not, xor), які регулюють взаємні відносини між параметрами для формування кінцевого результату про валідність вхідного процесу або явища відносно одного або декількох стандартів.

1.5.Унітарно-кодовані структури даних, сигнатурний аналіз

Інтерес представляє проблема аналізу великих даних з метою встановлення нових функциональностей на природному тлі вже визначених. Тут важливо отримати компактну таблицю функціональності, інваріантну до часу. Першим кроком в цьому напрямку є мінімізація

таблиці унітарного кодування функціональності за дозволеними логічними правилами (суперпозиція), які дають можливість отримати один стовпець, що ідентифікує функціональність. Структурні протиріччя при об'єднанні координат стовпців унітарно-кодованої матриці відсутні. Природно, що в результуючому стовпці-сигнатурі буде втрачена структурна інформація про порядок виконання сервісу, що є платою за компактність і швидкодію ідентифікації стовпців функціональності. Однак структурна інформація не стирається і може бути затребувана, в разі необхідності. Теоретичним підтвердженням і обгрунтуванням запропонованої суперпозиційної інновації зі стиснення стовпців в один є той факт, що кодування будь-якої таблиці істинності двома і навіть одним кубітним вектором, отриманим за допомогою суперпозиції унітарних кодів вхідних впливів будь-якого, як завгодно складного цифрового пристрою (рис. 1.10).

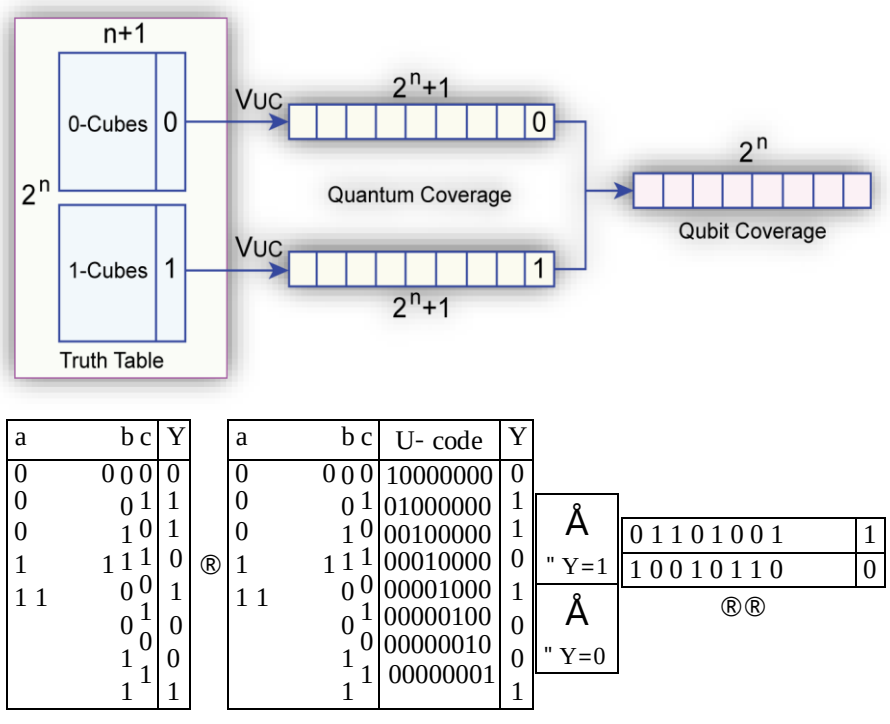


Рисунок 1.10 – Унітарне кодування універсуму примітивів за кожною

змінною

Обмеження: всі атрибути в матриці унітарного кодування, що підлягають суперпозиції за конкретними данними, повинні бути незалежними один від одного. Суперпозиція стовпців унітарної матриці дає можливість отримати покриття всіх атрибутів-змінних одиничними значеннями різноманітності даних. Якщо одиницями покриті в повному обсязі значення атрибутів, то існує некоректність в аналізі та кодування даних за конкретним атрибутом. В масштабах метрики значень інтегральний стовпець функціональності завжди буде являти собою підмножину з нульових і одиничних координат на тлі повністю одиничних значень інтегрального стовпчика універсуму P□R if P□R = P.

Суперпозіційна модель представлення функціональності инвариантна у часі. Ідея класифікації полягає в порівнянні великих даних з інтегральним вектором, який виходить шляхом суперпозиції або об'єднання всіх стовпців функціональності

$$P = \bigcup_{i=1}^{n} Pi.$$

Процедура ідентифікації зводиться до операції перетину між стовпцем вхідних даних та інтегральним стовпцем функціональності:

SÅ P«SÅ P=S, яка повинна дорівнювати вектору вхідних даних.

Природно, виникнуть ситуації, коли не буде виконуватися наведена вище умова за всіма порівняннями зі стовпцями функціональності. Тоді слід керуватися правилом домінування мінімальної кодової відстані Хеммінга:

$$S_i \text{ Å } P_j \text{«} \min(S_i \text{ j} \text{Å}_{=1} P_j = \text{Å}), i = 1,n;$$

$$S_i \text{ Å } P_j \text{«} \min(S_i \text{ j} \text{Å}_{=1} P_j = 1), i = 1,n.$$

Для аналізу детермінованої двійкової моделі функціональності існує ефективний апарат булевих похідних, який визначає істотність і неістотність змінних щодо формування вихідного значення функціональності. Якщо зміна стану змінної-атрибута не призводить до зміни функціональності, то така змінна є несуттєвою і її можна виключити з моделі функціональності.

Модельна надлишковість, як правило, є корисною для прискорення обчислювальних процесів за рахунок диверсифікації структур даних. Наприклад, просторово-часова модель функціональності за рахунок конволюції часу в точку може бути компактно представлена одним інтегральним стовпцем даних.

Багатозначність значень параметрів функціональності укладається в таку матричну модель:

$$P = [P_{ij}], i = 1,n; j = 1,m;$$

$$P = (P_1, P_2, ..., P_i, ..., P_n);$$

$$P_i = (P_{i1}, P_{i2}, ..., P_{ij}, ..., P_{im}).$$

Тут n – кількість рядків матриці функціональності, m – кількість значень параметра Pi при її кодуванні.

Для оптимізації функціональності необхідно і достатньо використовувати відомі аксіоми алгебри логіки:

1) aÅa =a «1Å1=1;
2) aÅab =a(1Åb)=a «1xÅ11=1;
3) abÅab =a(bÅb)=a «11Å10 =1x =1;
4) abcÅb =b.

Логічні аксіоми трансформуються в закони теорії множин, де фігурують елементи в форматі унітарних кодів значень вхідних змінних:

1) aÅ a =a; aÅ a =a;
2) aÅ ab =a(1Å b)=a «1xÅ 11=1x =1;
3) abÅ ab =a(bÅ b)=a «11Å 10 =1x =1; 4) abcÅ b =b«111Å 010 =010.

Всі вербальні значення або частини істотних (додаткових) параметрів повинні бути унітарно і єдино-метрично закодовані з метою представлення координат матриці функціональності двійковими векторами, які дають можливість в паралельному режимі визначати приналежність вхідного вектора одному або кільком стовпцям функціональності шляхом застосування логічної процедури:

a Å ab«aÅ ab=a ®10Å11=10Å11=10.

У загальному випадку метрична взаємодія двох компонентів однієї розмірності може мати тільки п'ять випадків, рис. 1.11 [1].

1. Належність чи рівність об'єктів один одному, якщо виконується умова:

a =b«aÅ b={a,b}®10Å10=10Å10=10.

2. Належність першого A другого m, якщо виконується умова: a Å b«aÅ b=a ®10Å11=10Å11=10.
3. Приналежність другого m першому A, якщо виконується умова: b Å a«aÅ b=b®11Å10=11Å10=10.
4. Часткова приналежність об'єктів один одному, якщо виконується умова:

a Åb«aÅ bÅ{Å ,a,b}®110Å 011=110Å011=010.

5. Неналежність об'єктів один одному, якщо виконується умова: a Åb«aÅ b=Å®01Å10 =01Å10 =00.

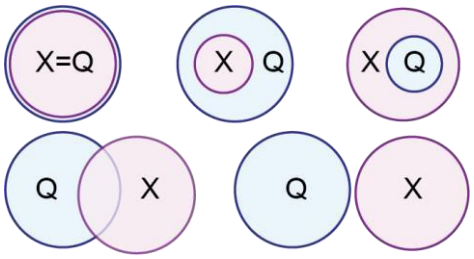


Рисунок 1.11 – Взаємодія даних за and-операцією

Структурна карта модулів комп'ютингу для аналізу С-процесів (рис. 1.12):

- синтез матриці істотних змінних;
- побудова унітарної матриці даних;
- декомпозиція унітарної матриці даних;
- синтез U-RPA (Robotic Process Automation) на основі застосування ML-технології до матриць С-функціональностей.

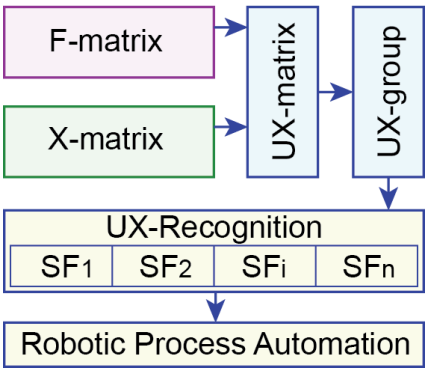


Рисунок 1.12 – Структура для аналізу процесів

Визначення унітарної матриці істотних змінних процесів і кодування всіх значень.

Рішення. Організовується цикл за n змінними функціональності, де всередині створюється цикл за значеннями змінних, де є ще один вкладений цикл, перераховує всі існуючі функціональності, які обробляються на предмет їх оригінальності (рис. 1.13). Таким чином, програмний модуль P-matrix, що містить три вкладених цикли, створює таблицю відповідності текстових значень істотних змінних їх десятковим номерам або унітарним кодам для подальшого аналізу процесів.

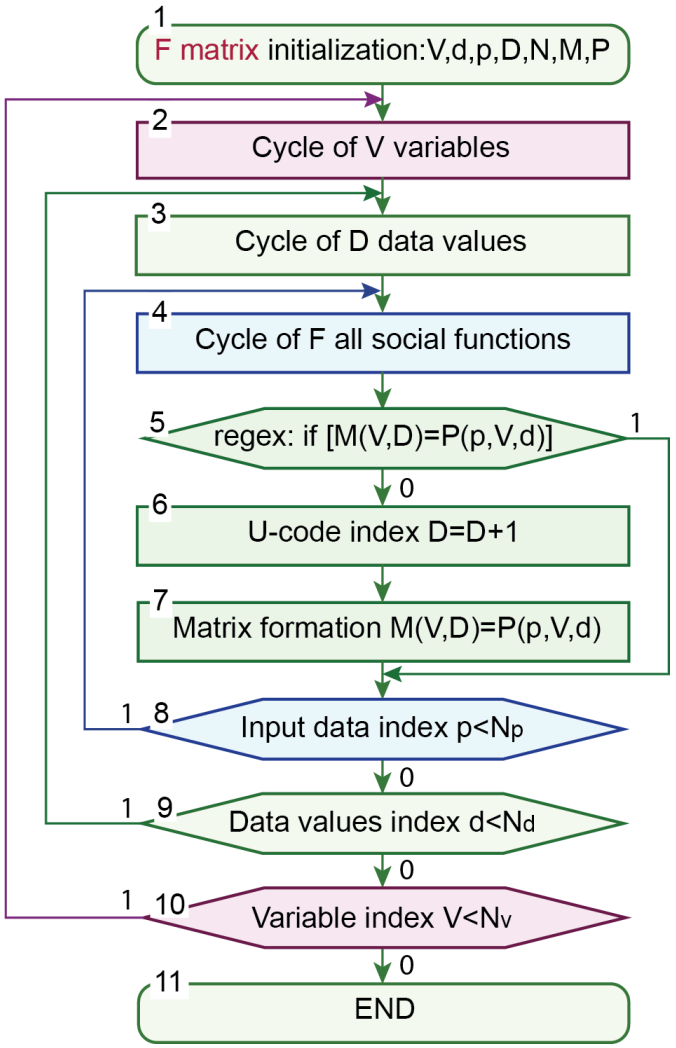


Рисунок 1.13 – Алгоритм формування матриці значень параметрів

1.6 Таблиця істинності функціональності

Таблиці істинності для завдання функциональностей формуються на основі позиційного або унітарного кодування значень змінних. При цьому передбачається замкнутість значень змінних в межах функціональності, які формують групу логічних функцій, заданих кубітними покриттями таблиць істинності. Конкретна функціональність може оперувати не більше, ніж n значеннями, змінної, які складають алфавіт або універсум примітивів-значень A = {A1, A2, ..., Ai, ..., An}. Нижче показана структура, яка ілюструє два види кодування універсуму примітивів-значень змінної для формування таблиць істинності трьох логічних функцій, які беруть участь у створенні функціональності:

A	Hash 16	P-Code	U-code	P1	P2	P3
A1	0100…10	00	1000	1	0	0
A2	0100…11	01	0100	1	0	1
A3	1100…10	10	0010	0	1	1
A4	0101…11	11	0001	0	1	0

Аналогічні таблиці істинності створюються для всіх змінних, що в сукупності формує матрицю таблиць істинності для кожного процесу або явища. За фактом, стовпець позиційного кодування P-code не використовується на практиці, але він слугує базовим компонентом для доказу застосовності класичної таблиці істинності при описі будь-яких процесів. Стовпці P1, P2, P3 тривіально використовуються для мінімізації кількості стовпців.

Далі пропонується матрична модель процесів на метриці значень змінних. Стовпці-кубіти P1, P2, P3 з попередньої таблиці трансформуються в рядки-вектори, які представляють собою суперпозицію унітарних кодів значень параметрів, що беруть участь у формуванні процесів: PT1 - PT3:

Pi	PT1	PT2	PT3
P1	1100	1101	1011
P2	0011	1011	0111
P3	0110	0111	0110
P4	1001	1011	1101

Дуалізм інтерпретації даної таблиці формує ієрархію, яку необхідно враховувати при синтезі моделі процесів. 1) Отримана таблиця або матриця об'єднання унітарних кодів, розміщених в координатах, являє собою двійкову модель функціональності, прив'язану до модельного часу PT1-PT3. Тут істотно, що кожна координата матриці є векторною або кубітною формою опису таблиці істинності. 2) Однак, формат матричної моделі також адекватно створює структури даних для опису сукупного

процесу. При цьому кожен стовпець матриці РТі інтегрально задає функціональність.

1.7 Метод синтезу логічних схем для моделювання процесів

Модель, алгебра, структура, граф, таблиця, матриця, система, рівняння є еквівалентними математичними поняттями, в основу яких покладено структуру взаємопов'язаних компонентів. При цьому завжди розглядається замкнутий алфавіт або множина примітивних компонентів, які створюють основу структури або універсум примітивів. Всі можливі зв'язки між елементами алфавіту або універсуму формують сигнатуру або базові операції алгебри. Булева алгебра на найнижчому рівні представлена алфавітом або універсумом примітивних символів {0,1}, які фігурують в значеннях булевих змінних і функції Y = f (X), де {X, Y} = {0,1}. При цьому поведінка функції визначається таблицею істинності, де впорядкованій двійковій послідовності вхідних змінних ставиться у відповідність двійкове значення функції. Менш поширеною є інтерпретація таблиці істинності, де кожному двійковому коду або адресі ставиться у відповідність єдине або нульове значення функції. При цьому сукупність кодів або адрес являє собою універсум примітивних компонентів або основу алгебри, на якій визначаються базові операції. Таким чином, вводиться алгебра логіки, де багатозначні стани вхідної змінної (алфавіт) кодуються в таблиці істинності двійковими векторами, які представляють собою адреси комірок пам'яті, де зберігаються {1,0}значення функції. Інтегрально 1-значення функції в таблиці істинності формують підмножину існуючих примітивів A = {a, c, e, f} на заданому універсумі A = {a, b, c, d, e, f, g, h}, який суперпозиційно створює функціональність:

String	Code	Function
a	000	1
b	001	0
c	010	1
d	011	0
e	100	1
f	101	1
g	110	0
h	111	0

Кубітним покриттям даної функціональності є F-вектор двійкових станів вихідної змінної, розмірність якого дорівнює універсуму примітивних компонентів, які формують функцію, а число одиничних значень дорівнює підмножині примітивів універсуму, яка бере участь у формуванні заданої функціональності. Слід зазначити, що функціональність формується значеннями суттєвої змінної у часових фреймах процесу. З огляду на те, що кількість істотних змінних, як правило більше 1, то необхідно синтезувати цифрові логічні схеми з кубітних покриттів функцій, число яких дорівнює кількості істотних змінних. Таким чином, кінцева множина істотних змінних є базовими елементами для синтезу цифрових логічних схем управління функціональності або процесу. Далі представлена структура, яка оперує кубітнимі покриттями примітивів, об'єднаними логікою елементів: and, or (рис. 1.14). Логічні схеми, синтезовані з кубітних форм значень змінних, призначені для моделювання процесів з метою визначення поведінки кіберфізичної архітектури комп'ютингу на заданих вхідних робочих впливах. Робочими впливами є суперпозиції унітарних кодів вхідних значень змінних. Стан виходу логічної схеми, що дорівнює одиниці, свідчить про позитивний результат взаємодії вхідних значень істотних змінних на хід виконання процесу для досягнення поставленої мети. Таким чином, замість ланцюжка даних, що ілюструє послідовність дій

функціональності, пропонується принципово нова форма – комбінаційна цифрова логічна схема, що паралельно інтегрує тільки істотні властивості функціональності, де основною відмінністю є можливість моделювання процесів.

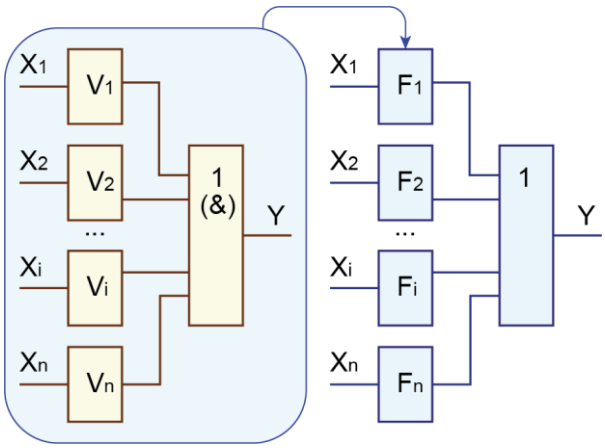


Рисунок 1.14 – Logic Business Functions

Схемна структура формує функціональну поведінку або модель функціональності на основі використання кубітних покриттів змінних, інваріантну до часу. Сукупність функциональностей створює основу для синтезу паралельної цифрової моделі процесу, яка являє собою спеціалізований обчислювач, який реалізує кібер-фізичний комп'ютинг для моніторингу, моделювання та управління процесами. Ієрархія комп'ютингу представлена компонентами:

<Значення – змінна – функціональність – процес» або

« value – variable – function – process>.

Логічна структура або процесор інваріантний до часу, створює функціональність на основі кубітних форм таблиць істинності логічних елементів, прив'язаних до універсуму примітивних значень змінних. Переваги процесора полягають в компактності представлення і високій швидкодії логічної схеми функціональності, яка визначається кубітними

векторами змінних, бітова розмірність яких дорівнює кількості примітивних даних-значень кожної змінної. Кубітні структури даних інваріантні до їх hardware реалізації у вигляді логічної схеми, або software у вигляді таблиць-матриць опису функциональностей. Кубітна двійкова форма опису бізнес-функциональностей дає можливість технологічно моделювати як завгодно складні процеси шляхом впливу на схему двійковими вхідними наборами, які відповідають унітарно-кодованим даним, в цілях визначення вихідних станів процесора, що виконують роль класифікатора і/або актюатора. Позитивним є той факт, що кожен кубітний логічний елемент, заданий у векторному форматі кількості значень змінної, створює образ функціональності одиничними значеннями своїх координат. При цьому число кубітних векторів-елементів, об'єднаних в схему and (or)-елементом, дорівнює кількості змінних синтезованої функціональності. Крім того, візуалізація досить компактної логічної схеми завдання функціональності дає можливість користувачеві побачити сутність функциональності без прив'язки до часу.

1.8 Висновки

Створено модель відношень порядку в структурах даних і компьютингу, яка формує ефективність паралельного управління алгоритмами обробки великих даних за рахунок суперпозиціонування кінцевої множини дискретних станів.

Розроблено структурну модель взаємодії кубітних покриттів логічних функцій і похідних компонентів, орієнтованих на синтез і аналіз цифрових систем з метою отримання перевіряльних тестових

послідовностей для поодиноких константних несправностей, а також зменшення часу проектування і тестування обчислювальних пристроїв.

Введено поняття простих X-функцій від кінцевої кількості змінних, які характеризуються відсутністю мінімізації та наявністю властивостей тестопридатності, що дає можливість синтезувати цифрові пристрої, технологічні для вирішення задач тестування, моделювання та діагностування.

Запропоновано tree-driven ATPG процесор і структури даних для обчислення кубітних булевих похідних, представлений двійковим деревом-графом xor-елементів для паралельної обробки частин кубітного покриття.

Запропоновано кубітні методи синтезу тестів для логічних функцій, які характеризуються квадратичною і лінійною обчислювальною складністю алгоритмів для генерування перевіряльних послідовностей.

2 ВИРІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ МАЛЮНКІВ

База даних «Quick Draw» складається з 50 мільйонів зображень, що належать до однієї з 340 категорій. Тестова вибірка отримана із загального датасета в розмірі 10 відсотків від загальної кількості об’єктів. Зображення в датасеті зберігаються у вигляді записів про рухи пензлем в деякій системі координат. Кожен малюнок складається зі списку штрихів, а кожен штрих визначається набором координат рухів пензлем (рис. 2.1).

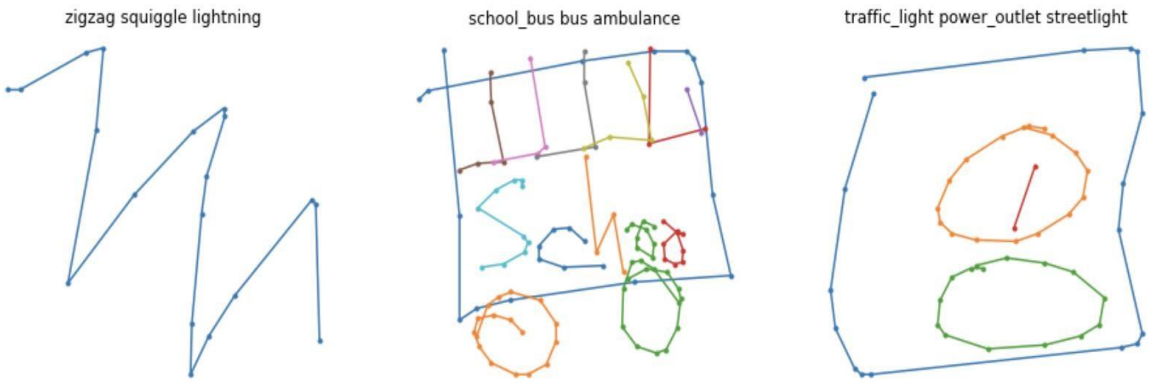


Рисунок 2.1 – Приклад відмальованих зображень з бази Quick Draw із зазначенням категорії

Алгоритм розпізнавання реалізований мовою Python з використанням фреймворку для машинного навчання – Keras.

Для прискорення, навчання алгоритму використано платформу Google Cloud Platform в якості віддаленого серверу та операційну систему Ubuntu 16.04.

2.1 Моделі і методи розпізнавання образів

Розпізнавання образів – це розділ кібернетики, що розвиває теоретичні основи та методи класифікації й ідентифікації предметів, явищ, процесів, сигналів, ситуацій і об'єктів, які характеризуються кінцевим набором деяких властивостей і ознак [14-15]. Задача розпізнавання образів вирішується, наприклад, при переході або проїзді вулиці за сигналами світлофора. Розпізнавання кольору лампи світлофора, що засвітилася, і знання правил дорожнього руху дозволяє прийняти правильне рішення про те, можна, чи не можна переходити вулицю в цей момент.

У процесі біологічної еволюції багато тварин за допомогою зорового й слухового апарата вирішили задачу розпізнавання образів досить добре. Створення штучних систем розпізнавання образів залишається складною теоретичною й технічною проблемою.

Необхідність у розпізнаванні виникає в різних областях – від військової справи та систем безпеки до оцифровування аналогових сигналів.

Традиційно задачі розпізнавання образів відносять до штучного інтелекту. Одним з напрямів вирішення цих задач є нейронні мережі.

2.2 Нейронні мережі

Штучні нейронні мережі (ШНМ) – це обчислювальні системи, що здійснюють симуляцію мозку живих істот. За рахунок навчання ці системи поступово покращують продуктивність вирішення задач. Наприклад, у розпізнаванні зображень ШНМ можна навчити ідентифікувати зображення котів, на основі аналізу прикладів зображень, позначених як «кіт» і «не кіт», і використовувати результати для ідентифікування котів в інших зображеннях. Системи розпізнавання здійснюють це без жодного апріорного знання про котів, наприклад, що вони мають хутро, хвости, вуса. Натомість, вони розвивають свій власний набір характеристик з навчального матеріалу, який вони обробляють.

ШНМ ґрунтується на сукупності з'єднаних вузлів, які називають штучними нейронами (аналогічно до біологічних нейронів у головному мозку тварин). Кожне з'єднання штучних нейронів може передавати сигнал від одного нейрону до іншого аналогічно синапсу. Штучний

нейрон, що отримує сигнал, може обробляти його і потім сигналізувати штучним нейронам, приєднаним до нього.

В поширених реалізаціях ШНМ сигнал на з'єднанні між штучними нейронами є дійсним числом, а вихід кожного штучного нейрону обчислюється нелінійною функцією суми його входів. Штучні нейрони та з'єднання зазвичай мають вагу, яка підлаштовується під час навчання. Вага збільшує або зменшує силу сигналу з'єднання. Штучні нейрони можуть мати такий поріг, що сигнал надсилається лише якщо сукупний сигнал перетинає цей поріг. Штучні нейрони зазвичай організовано в шари. Різні шари можуть виконувати різні види перетворень своїх входів. Сигнали проходять від першого (входового) до останнього (виходового) шару, можливе проходження скрізь шари декілька разів.

Первинною метою підходу ШНМ було розв'язання задач шляхом імітації роботи мозку людини. З часом увага зосередилася на відповідності системи певним розумовим здібностям людини. ШНМ використовується в ряді різноманітних задач, у тому числі в системах комп'ютерного зору, розпізнавання мовлення, машинного перекладу, соціально-мережевих фільтрах, гри в настільні та відеоігри, та медичному діагностуванні.

2.2.1 Складові штучної нейронної мережі

Нейронна мережа містить такі складові:

а) нейрон з міткою j , що отримує вхід $p_j(t)$ від нейронів-попередників, складається з таких компонентів: лінійна функція трансформації вхідних даних завдяки вагам вхідних з’єднань, а також функції активації або збудження, яка стискає значення лінійної функції в певний діапазон (рис. 2.2);

б) сінапси – зв'язки між нейронами які мають ваги;

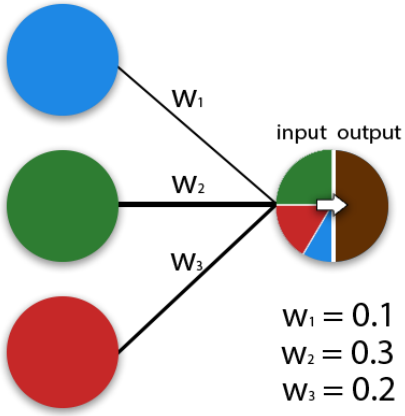


Рисунок 2.2 – Схема нейрону: output – функція збудження, input – лінійна функція

2.2.2 Функція збудження

У штучних нейронних мережах функція активації нейрона визначає вихідний сигнал за допомогою вхідного сигналу або набору вхідних сигналів. Стандартна комп'ютерна мікросхема може розглядатися як цифрова мережа функцій активації, які можуть набувати значень «ON» (1) або «OFF» (0) в залежності від входу. Це схоже на поведінку лінійного перцептрону в нейронних мережах. Однак тільки нелінійні функції активації дозволяють таким мережам вирішувати нетривіальні завдання з використанням малого числа вузлів. У штучних нейронних мережах ця функція також називається функцією передачі (рис. 2.3).

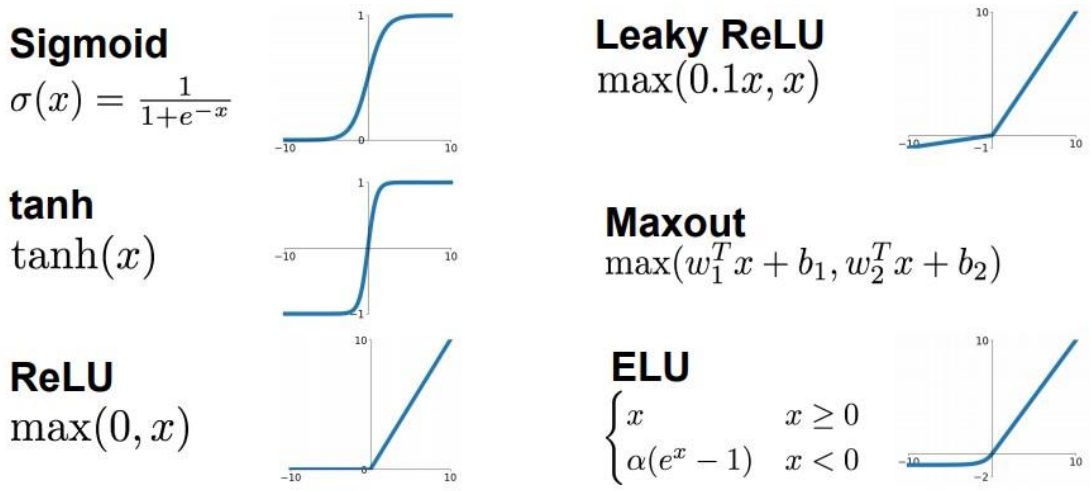


Рисунок 2.3– Типові функції збудження

Зазначеніі функції використовуються щоб зробити нейронну мережу більш гнучкою і навченою, але також вони потрібні для завдання діапазону значень виходу нейрона. Наприклад, якщо нам необхідно моделювати ймовірність будь-якого явища, тим самим обмежити вихідні значення (0,1), то використовується функція сигмоид, яка має відповідний розподіл. Ми не можемо використовувати ступінчасту функцію, так як нейронна мережа буде не здатна навчатися.

Але для вирішення поставленого завдання необхідно мати такий вихід нейронної мережі, який би описував приналежність розглянутого зображення до одного із заданих класів. Для цього використовується особлива функція збудження softmax (рис. 2.4). На відміну від попередніх функцій, які брали на вхід виключно вихід одного нейрона, ця функція приймає множину значень і перетворює їх в розподіл ймовірностей, усі значення яких сумуються в 1. Таким чином ми можемо моделювати відсоткову належність вхідного екземпляру до всіх класів.

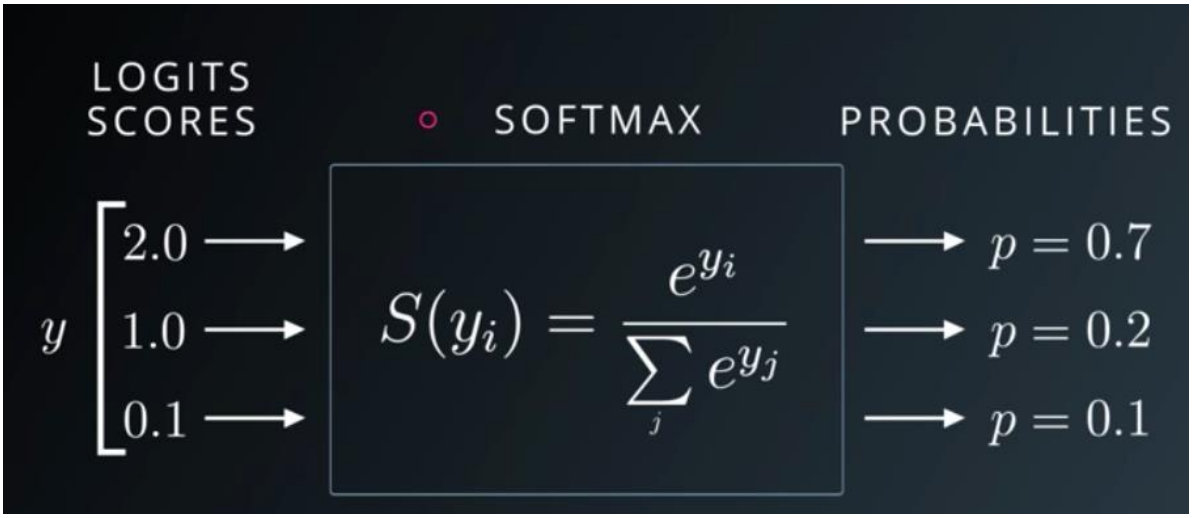


Рисунок 2.4 – Softmax функція збудження

2.2.3 Згорткові нейронні мережі

Згорткові нейронні мережі (ЗНМ) в машинному навчанні – це клас глибинних штучних нейронних мереж прямого поширення, який успішно застосовувався до аналізу візуальних зображень.

ЗНМ використовують різновид багатошарових перцептронів, які потребують мінімальний обсяг попередньої обробки. Вони відомі також як інваріантні відносно зсуву або просторово інваріантні штучні нейронні мережі, виходячи з їхньої архітектури спільних ваг та характеристик інваріантності відносно паралельного перенесення [14,15].

Згорткові мережі імітують біологічні процеси [14,15], в яких схема з'єднання нейронів базується на принципі організації зорової кори тварин. Окремі нейрони кори реагують на стимули лише в обмеженій області зорового поля, відомій як рецептивне поле. Рецептивні поля різних нейронів частково перекриваються таким чином, що вони покривають усе зорове поле.

ЗНМ використовують порівняно невелику кількість попередньої обробки, порівняно з іншими алгоритмами класифікування зображень. Це означає, що мережа навчається за допомогою фільтрів, які в традиційних алгоритмах розроблялися вручну. Ця незалежність конструювання ознак від апріорних знань та зусиль оператора є перевагою ЗНМ. Зазначені системи застосовуються в розпізнаванні зображень та відео, рекомендаційних системах та обробленні природної мови.

2.2.3.1 Структура ЗНМ

ЗНМ складається з шарів входу та виходу, а також із декількох прихованих шарів. Приховані шари ЗНМ зазвичай складаються зі згорткових, агрегувальних, повноз'єднаних та нормалізувальних шарів.

Процес функціонування нейронної мережі описують як згортку за домовленістю. З математичної точки зору цей процес є більше взаємною кореляцією, ніж згорткою. Це має значення лише для індексів матриці, визначення ваг та їх розташування.

2.2.3.2 Згорткові шари

Згорткові шари застосовують на вході операції згортки та передають результат до наступного шару. Згортка імітує реакцію окремого нейрону на зоровий стимул.

Кожен згортковий нейрон обробляє дані лише для свого рецептивного поля. Повноз'єднані нейронні мережі прямого поширення можуть бути застосовані для навчання ознак і для класифікування даних, але використання цієї архітектури для розпізнавання зображень є непрактичним. Була б необхідна дуже велика кількість нейронів, навіть у поверхневій (протилежній до глибинної) архітектурі, через дуже великі

розміри входу, пов'язані з зображеннями, де кожен піксель є відповідною змінною. Наприклад, повноз'єднаний шар для (маленького) зображення розміром 100 × 100 має 10 000 ваг. Операція згортки дає змогу розв'язати цю проблему, оскільки вона зменшує кількість вільних параметрів, дозволяючи мережі бути глибшою за меншої кількості параметрів [16]. Наприклад, незалежно від розміру зображення, області замощування розміру 5 × 5, кожна з одними й тими ж спільними вагами, вимагають лише 25 вільних параметрів. Таким чином, це розв'язує проблему зникання або вибуху градієнтів у тренуванні традиційних багатошарових нейронних мереж з багатьма шарами за допомогою зворотного поширення.

2.2.3.3 Агрегувальні шари

Згорткові мережі можуть включати шари локального або глобального агрегування, які об'єднують виходи кластерів нейронів одного шару до одного нейрону наступного шару. Наприклад, максимізаційне агрегування використовує максимальне значення з кожного з кластерів нейронів попереднього шару. Іншим прикладом є усереднювальне агрегування, що використовує усереднене значення з кожного з кластерів нейронів попереднього шару.

2.2.3.4 Повноз'єднані шари

Повноз'єднані шари з'єднують кожен нейрон одного шару з кожним нейроном наступного шару таким чином, як і традиційна нейронна мережа багатошарового перцептрону (БШП).

ЗНМ використовують спільні ваги в згорткових шарах, що означає, що для кожного рецептивного поля шару використовується один і той же

фільтр (банк ваг); це зменшує обсяг необхідної пам'яті та поліпшує продуктивність.

2.2.3.5 Пакетна нормалізація

Пакетна нормалізація (Batch-normalization) – це метод, який дозволяє підвищити продуктивність і стабілізувати роботу штучних нейронних мереж. Сутність даного методу полягає в тому, що до деяких шарів нейронної мережі на вхід подаються дані, попередньо оброблені, які мають нульове математичне сподівання і одиничну дисперсію.

Нормалізація вхідного шару нейронної мережі зазвичай виконується шляхом масштабування даних, що подаються до функцій активації. Наприклад, коли є ознаки зі значеннями від 0 до 1 і деякі ознаки зі значеннями від 1 до 1000, то їх необхідно нормалізувати, щоб прискорити навчання. Нормалізацію даних можна виконати і в прихованих шарах нейронних мереж за допомогою методу пакетної нормалізації.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)_2$$

$$x_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

2.2.4 Рекуррентні нейронні мережі

Рекурентні нейронні мережі — це клас штучних нейронних мереж, у якому з'єднання між вузлами утворюють граф, орієнтований у часі [17]. Це

створює внутрішній стан мережі, що дозволяє їй проявляти динамічну поведінку в часі. На відміну від нейронних мереж прямого поширення, РНМ можуть використовувати свою внутрішню пам'ять для обробки довільних послідовностей входів. Це робить їх застосовними до таких задач, як розпізнавання несегментованого неперервного рукописного тексту та розпізнавання мовлення.

2.2.4.1 Повнорекурентна мережа

Повнорекурентна мережа – це основна архітектура, розроблена у 1980-х роках: мережа нейроноподібних вузлів, кожен з орієнтованим з'єднанням до кожного іншого вузла [17]. Кожен з вузлів має змінну в часі дійснозначну активацію. Кожне з'єднання має змінювану дійснозначну вагу. Деякі з вузлів називаються входовими вузлами, деякі – виходовими, а решта– прихованими вузлами. Більшість із наведених нижче архітектур є окремими випадками.

Для керованого навчання з дискретним часом тренувальні послідовності входових векторів стають послідовностями активацій входових вузлів, по одному вектору на кожен момент часу. В кожен заданий момент часу кожен не входовий вузол обчислює свою поточну активацію як нелінійну функцію від зваженої суми активацій всіх вузлів, від яких до нього надходять з'єднання. Для деяких із виходових вузлів на певних тактах можуть бути задані вчителем цільові активації. Наприклад, якщо входова послідовність є мовним сигналом, що відповідає вимовленій цифрі, то кінцевий цільовий вихід у кінці послідовності може бути міткою, яка класифікує цю цифру. Для кожної послідовності її похибка є сумою відхилень усіх цільових сигналів від відповідних активацій, обчислених

мережею. Для тренувального набору численних послідовностей загальна похибка є сумою похибок усіх окремих послідовностей.

У процесі навчання з підкріпленням не існує вчителя, який надавав би цільові сигнали для РНМ, натомість час від часу застосовується функція пристосованості або функція винагороди для оцінювання продуктивності РНМ, яка впливає на її входовий потік через виходові вузли, з'єднані з приводами, що впливають на середовище.

2.2.4.2 Рекурсивні нейронні мережі

Рекурсивна нейронна мережа створюється рекурсивним застосуванням одного й того ж набору ваг до диференційовної графоподібної структури шляхом обходу цієї структури в топологічній послідовності [18]. Також, такі мережі зазвичай тренують зворотним режимом автоматичного диференціювання. Їх було введено для навчання розподілених представлень структури, таких як терміни логіки. Окремим випадком рекурсивних нейронних мереж є РНМ, структура яких відповідає лінійному ланцюжкові. Рекурсивні нейронні мережі застосовували до обробки природної мови. Рекурсивна нейронна тензорна мережа використовує функцію компонування на основі тензорів для всіх вузлів дерева.

Ідея RNN полягає в послідовному використанні інформації. У традиційних нейронних мережах мається на увазі, що всі входи і виходи незалежні. Але для багатьох завдань це не підходить. Якщо потрібно передбачити наступне слово в реченні, краще враховувати попередні слова. RNN називаються рекурентними, тому що вони виконують одну й ту ж задачу для кожного елемента послідовності, причому вихід залежить від попередніх обчислень. Ще одна інтерпретація RNN: це мережі, у яких

є «пам'ять», яка враховує попередню інформацію. Теоретично RNN можуть використовувати інформацію в довільно довгих послідовностях, але на практиці вони обмежені лише кількома кроками. На рис. 2.5 показано, що RNN розгортається в повну мережу.

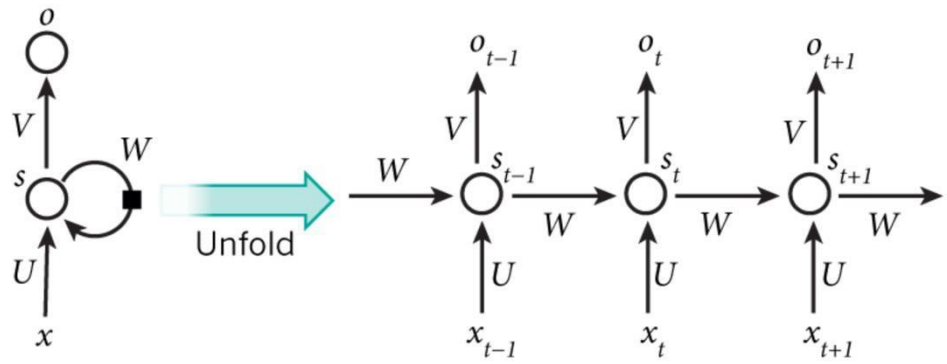


Рисунок 2.5 - Рекурентна нейронна мережа і її розгортка

Розгорткою виписується мережа для повної послідовності. Наприклад, якщо послідовність являє собою пропозицію з 5 слів, розгортка буде складатися з 5 шарів, по шару на кожне слово. Формули, що задають обчислення в RNN наведено нижче.

Параметр x_t – вхід на часовому кроці t. Наприклад, x_1 може бути вектором з одним гарячим станом (one-hot vector), відповідним другому слову пропозиції.

Параметр s_t – прихований стан на кроці t. Це «пам'ять» мережі. Параметр s_t залежить, як функція, від попередніх станів і поточного входу x_t: s_t = f(Ux_t + Ws_{t-1}). Функція f зазвичай нелінійна, наприклад tanh або ReLU. Параметр s_{- 1}, потрібний для обчислення першого прихованого стану, зазвичай ініціалізується нулем (нульовим вектором).

Параметр o_t – вихід на кроці t. Наприклад, якщо потрібно передбачити слово в реченні, вихід може бути вектором ймовірностей в словнику: o_t = softmax (Vs_t).

2.2.4.3 Довга короткострокова пам'ять (LSTM)

Довга короткострокова пам'ять (Long short-term memory; LSTM) – особливий різновид архітектури рекурентних нейронних мереж, здатна до навчання довготривалим залежностям [19]. Вони були представлені Зеппом Хохрайтер і Юргеном Шмідхубер (Jürgen Schmidhuber) у 1997 році, а потім вдосконалені і популярно викладені в роботах багатьох інших дослідників. Вони прекрасно вирішують цілий ряд різноманітних завдань і в даний час широко використовуються.

LSTM розроблені спеціально, щоб уникнути проблеми довготривалої залежності. Запам'ятовування інформації на довгі періоди часу – це їх звичайна поведінка, а не щось, чого вони насилу намагаються навчитися.

Будь-яка рекуррентна нейронна мережа має форму ланцюжка повторюваних модулів нейронної мережі. У звичайній RNN структурі, наприклад, він може являти собою один шар з функцією активації tanh (гіперболічний тангенс) (рис. 2.6).

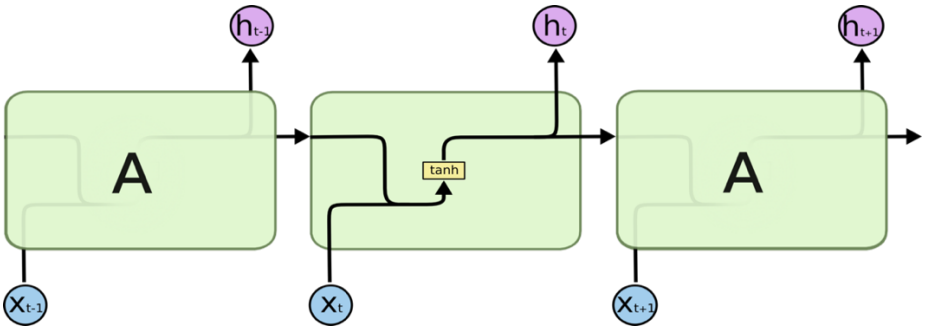


Рисунок 2.6 - Схема простої нейронної мережі

Повторюваний модуль в стандартній RNN складається з одного шару.

Структура LSTM також нагадує ланцюжок, але модулі виглядають інакше. Замість одного шару нейронної мережі вони містять цілих чотири, і ці шари взаємодіють особливим чином (рис. 2.7).

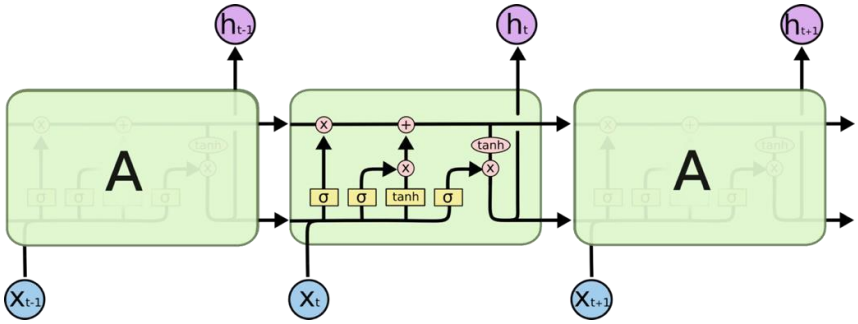


Рисунок 2.7 – Схема LSTM мережі

Ключовий компонент LSTM – це стан комірки (cell state) – горизонтальна лінія, що проходить по верхній частині схеми.

Стан комірки нагадує конвеєрну стрічку (рис. 2.8). Вона проходить безпосередньо через весь ланцюжок, беручи участь лише в декількох лінійних перетвореннях. Інформація може легко проходити по ній, не змінюютись.

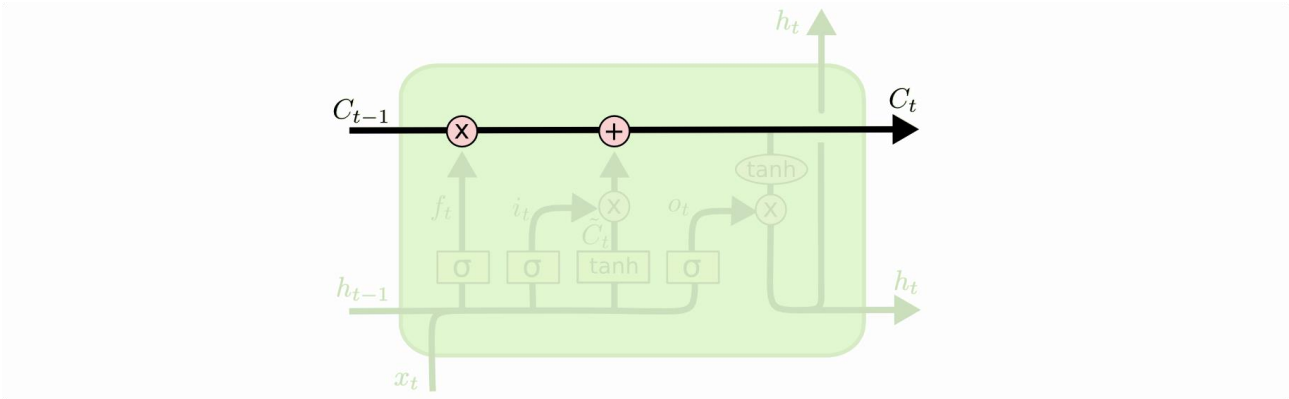


Рисунок 2.8 - Стан LSTM мережі

Проте, LSTM може видаляти інформацію зі стану комірки; цей процес регулюється структурами, що називають фільтрами (gates).

Фільтри дозволяють пропускати інформацію на підставі деяких умов. Вони складаються з шару сигмоїдальної нейронної мережі та операції поточечного множення.

2.2.4.4 Керовані рекурентні блоки (GRU)

Керовані рекурентні блоки – це механізм вентилів для рекурентних нейронних мереж, представлений в 2014 році. Було встановлено, що його ефективність при вирішенні задач моделювання музичних і мовних сигналів порівнянна з використанням довгої короткостроковій пам'яті (LSTM) [16]. У порівнянні з LSTM у даного механізму менше параметрів, тому що відсутній вихідний вентиль.

Архітектура блоку представлена наступним чином:

$$z_t = \sigma_g(W_z X_t + U_z h_{t-1} + b_z);$$

$$r_t = \sigma_g(W_r X_t + U_r h_{t-1} + b_r);$$

$$h_t = z_g \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h),$$

де x_t – вхідний вектор; h_t – вихідний вектор; z_t – вектор вентиля оновлення; r_t – вектор вентиля скидання; W, U, b – матриці параметрів та вектор відповідно.

2.3 Навчання нейронних мереж

Навчання нейронних мереж відбувається шляхом подачі вихідних

даних на вхід мережі (наприклад вхідних зображень), і співставленням отриманих вихідних значень з істинними даними які ми хочемо отримати (наприклад категоріями зображень), так званими мітками. Такий підхід називається навчання з учителем, коли у нас є і дані і відповідні їм мітки при навчанні. Також існують види навчання без міток (навчання без вчителя), із не повністю розміченими даними (часткове навчання з учителем).

Таке навчання, є ітераційним, тобто с кожною итераціэю ми все ближче і ближче підбираємося до бажаних міток. Навчання триватиме до тих пір поки схожість вихідних значень мережі і міток не стане максимальною, або інакше кажучи відмінність між ними не стане мінімальною. Ця різниця між одержуваними значеннями та істинними мітками називається помилкою. І весь процес навчання може бути визначений як мінімізація цієї помилки за рахунок оновленя параметрів нейронів мережі, які називаються вагами. Найпоширеніший алгоритм мінімізації помилки це градієнтний спуск. Він є базовим алгоритмом, на основі якого базуються більш просунуті методи.

2.3.1 Функція помилки

У тих випадках, коли вдається оцінити роботу мережі, навчання нейронних мереж можна уявити як задачу оптимізації. Оцінити – означає вказати кількісно, добре або погано мережа вирішує поставлені їй завдання. Для цього будується функція оцінки. Вона, як правило, явно залежить від вихідних сигналів мережі і неявно (через функціонування) – від всіх її параметрів. Найпростіший і найпоширеніший приклад оцінки – сума квадратів відстаней від вихідних сигналів мережі до їх необхідних значень:

$$H = \frac{1}{2} \sum_{\tau \in v_{out}} (Z(\tau) - Z^*(\tau))^2,$$

де v_{out} – необхідне значення вихідного сигналу.

Метод найменших квадратів далеко не завжди є кращим вибором оцінки. Ретельне конструювання функції оцінки дозволяє на порядок підвищити ефективність навчання мережі, а також отримувати додаткову інформацію – «рівень впевненості» мережі е відповіді, що надається.

2.3.2 Перехресна ентропія

Перехресна ентропія – одна з можливих функцій помилки, використовується для кількісної оцінки різниці між двома розподілами ймовірностей. Зазвичай «справжній» розподіл, виражається в термінах одноразового поширення, він використовується в задачах мультикласової класифікації. Саме вона була використана в цьому проекті:

$$H = - \sum_{\tau \in v_{out}} p(x) log q(x).$$

2.3.3 Градієнтний спуск

Градієнтний спуск – метод знаходження мінімального значення функції помилки (існує багато видів цієї функції). Мінімізація будь-якої функції означає пошук найглибшої западини в цій функції. Майте на увазі, що функція використовується, щоб контролювати помилку в прогнозах моделі машинного навчання. Пошук мінімуму означає отримання найменшої можливої помилки або підвищення точності моделі (рис. 2.9).

Ми збільшуємо точність, перебираючи набір навчальних даних при налаштуванні параметрів нашої моделі (ваг і зміщень).

Сутність алгоритму – процес отримання найменшого значення помилки (рис. 2.10). Аналогічно це можна розглядати як спуск в западину в спробі знайти золото на дні ущелини (найнижче значення помилки).

Надалі, щоб знайти найнижчу помилку (найглибшу западину) в функції втрат (відносно однієї ваги), потрібно налаштувати параметри моделі. У цьому допоможе математичний аналіз. Завдяки аналізу ми знаємо, що нахил графіка функції – похідна від функції за змінною. Ций нахил завжди вказує на найближчу западину.

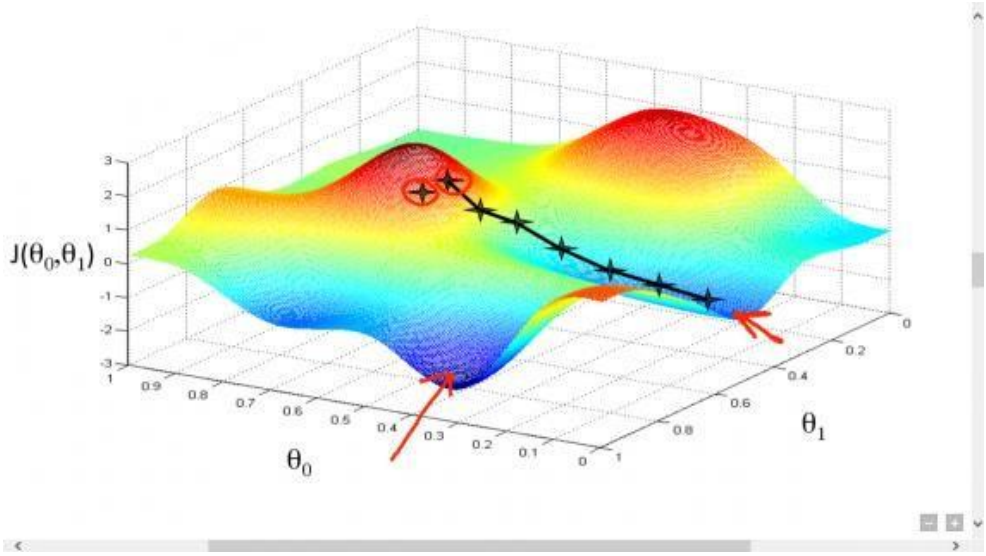


Рисунок 2.9 – Пошук мінімуму функції

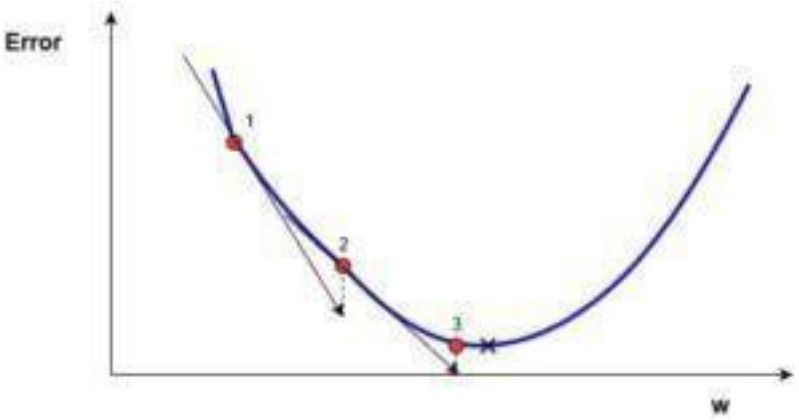


Рисунок 2.10 – Функція помилки

На рис. 2.10 ми бачимо графік функції втрат (названий «Помилка» з символом «J») з однією вагою. Тепер, якщо ми обчислимо нахил (позначимо це dJ/dw) функції втрат щодо однієї ваги, то отримаємо напрям, в якому потрібно рухатися, щоб досягти локального мінімуму. Припустимо, що наша модель має тільки одну вагу. Коли ми перебираємо всі навчальні дані, ми продовжуємо додавати значення dJ/dw для кожної ваги. Так як втрати залежать від прикладу навчання, dJ/dw також продовжує змінюватися. Потім ділимо зібрані значення на кількість прикладів навчання для здобуття середнього значення. Потім ми використовуємо це середнє значення (кожної ваги) для настроювання кожної ваги.

Тепер, коли ми визначили напрямок, в якому потрібно шукати вагу, потрібно використовувати коефіцієнт швидкості навчання, який називають гіпер-параметром. Зазвичай ці значення можуть бути отримані експериментальним шляхом. Коефіцієнт швидкості навчання можна розглядати як «крок у правильному напрямку», де напрямок походить від dJ/dw. Формула зміни ваг наведена нижче, де α – це швидкість навчання.

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta).$$

Функція помилки побудована тільки для однієї ваги. У реальній моделі все перераховане вище виконується для всіх ваг, перебираючи все приклади навчання. Навіть у відносно невеликій моделі машинного навчання буде більше ніж 1 або 2 ваги. Це ускладнює візуалізацію, оскільки графік буде мати великі розміри.

2.3.4 Інші види оптимізаторів

Градієнтний спуск широко використовується для навчання мережі, проте в розлянутому прикладі ми мали тільки одну вагу. Навіть найменші нейронні мережі, як правило конструюються з декількох шарів і мають більше декількох десятків ваг, не кажучи вже про глибокі мережі, використовувані в реальних задачах, з кількістю параметрів, які обчислюються від декількох тисяч до мільйонів. Такі моделі дозволяють будувати дуже складні відображення, проте мають надвелику розмірність. Існують такі проблеми навчання:

а) застрявання в локальних мінімумах або сідлових точках, яких для функції від > 10^6 змінних може бути дуже багато;

б) складний ландшафт цільової функції: плато чередуються із зонами сильної нелінійності; похідна на плато практично дорівнює нулю, а раптовий обрив, навпаки, може відправити алгоритм занадто далеко;

в) деякі параметри оновлюються значно рідше за інші, особливо коли в даних зустрічаються інформативні, але рідкісні ознаки, що погано позначається на нюансах узагальнюючого правила мережі; з іншого боку, надання занадто великої значущості взагалі всім параметрам рідко зустрічається і може привести до перенавчання;

г) занадто маленька швидкість навчання змушує алгоритм сходитися дуже довго і застрявати в локальних мінімумах, занадто велика – «пролітати» вузькі глобальні мінімуми або зовсім розходитися.

В наш час відомі просунуті алгоритми другого порядку, яким під силу знайти хороший мінімум на складному ландшафті, але тут з'являється інша проблема – кількість ваг. Щоб скористатися чесним методом другого порядку «в лоб», доведеться порахувати гессіан $\backslash J(\theta)$ – матрицю похідних

за кожною парою параметрів, а для методу Ньютона – ще й зворотний до неї.

2.3.4.1 Прискорений градієнт Нестерова

Ідея методів з накопиченням імпульсу проста: «Якщо ми деякий час рухаємося в певному напрямку, то, ймовірно, нам слід туди рухатися деякий час і в майбутньому». Для цього потрібно вміти звертатися до недавньої історії змін кожного параметра. Можна зберігати останні n екземплярів $\Delta\theta$ і на кожному кроці по-чесному визначати середнє, але такий підхід потребує надто багато пам'яті для великих n. Для вирішення задачі не потрібно точне середнє, а лише оцінка, тому можна скористатися експоненціальним ковзним середнім:

$$v_t = \gamma v_{t-1} + (1-\gamma)x.$$

Для накопичення інформації, потрібно множити вже накопичене значення на коефіцієнт збереження $0 < \gamma < 1$ і додавати чергову величину, помножену на $1 - \gamma$. Чим ближче γдо одиниці, тим більше вікно накопичення і сильніше згладжування – історія x починає впливати сильніше, ніж кожне чергове x. Якщо x = 0 починаючи з певного моменту, v_t зменшуються у геометричній прогресії, експоненціально. Застосуємо експоненціальне біжуче середнє, щоб накопичувати градієнт цільової функції мережі:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta);\ \theta = \theta - v_t.$$

2.3.4.2 Оптимізаційний алгоритм Adam

Adam (Adaptive Moment Estimation), ще один оптимізаційний алгоритм. Він поєднує в собі ідею накопичення руху та ідею слабшого поновлення ваг для типових ознак:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t..$$

Від алгоритма Нестерова Adam відрізняється тим, що ми накопичуємо не $\Delta\theta$, а значення градієнта. Крім того, потрібно знати, як часто градієнт змінюється. Автори алгоритму запропонували для цього оцінювати також середню нецентровану дисперсію:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_{t2} .$$

Важлива відмінність полягає в початковій калібрування m^t і v^t:
вони мають таку ж саму проблему, що і $E[g^2]_t$ в RMSProp: якщо задати нульове початкове значення, то вони будуть довго накопичуватися, особливо при великому вікні накопичення ($0 \ll \beta_1 < 1,\ 0 \ll \beta_2 < 1$), а початкові значення – це ще два гіперпараметра. Якщо не додавати гіперпараметри, треба штучно збільшити m^t і v^t на перших кроках (приблизно 0 < t < 10 для m^t і 0 < t < 1000 для v^t):

$$m_t = \frac{m_t}{1 - \beta_{1t}}, \quad v_t = \frac{v_t}{1 - \beta_{2t}}.$$

У підсумку, правило поновлення має вигляд:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} m_t.$$

2.3.5 Метод зворотного поширення помилки

Метод зворотного поширення помилки – метод навчання багатошарового перцептрона. Вперше метод був описаний в 1974 р А.І. Галушкіно, а також незалежно і одночасно Полом Дж. Вербосом. Далі істотно розвинений в 1986 р Девідом І. Румельхартом, Дж. Е. Хінтон і Рональдом Дж. Вільямсом і незалежно і одночасно С.І. Барцевим і В.А. Охоніним. Це ітеративний градієнтний алгоритм, який використовується з метою мінімізації помилки роботи багатошарового перцептрона і отримання бажаного виходу.

Основна ідея цього методу полягає в поширенні сигналів помилки від виходів мережі до її входів, в напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи. Загальний метод («принцип подвійності») Барца і Охоніна може бути застосований до більш широкого класу систем, у тому числі системи з запізненням, розподіленої системи.

Для застосування методу зворотного поширення помилки передавальна функція нейронів повинна бути диференційована. Метод є модифікацією класичного методу градієнтного спуску.

Алгоритм зворотного поширення помилки застосовується для багатошарового перцептрона. У мережі є множина входів $x_1, \ldots, x_n$, множина виходів Outputs і множина внутрішніх вузлів. Перенумеруємо всі вузли (у тому числі входи і виходи) числами від 1 до N (наскрізна

нумерація, незалежно від топології шарів). Позначимо через $w_{i,j}$ вагу, що стоїть на ребрі, яке з'єднує і-й та j-й вузли, а через o_i – вихід і-го вузла. Якщо нам відомий навчальний приклад (правильні відповіді мережі $t_k, k \in Outputs$), то функція помилки, отримана за методом найменших квадратів, виглядає так:

$$E(\{w_{i,j}\}) = \frac{1}{2} \sum_{k \in Outputs} (t_k - o_k)^2$$

Для модифікації ваг будемо реалізовувати стохастичний градієнтний спуск, тобто підправляти ваги після кожного навчального прикладу і, таким чином, «рухатися» в багатовимірному просторі ваг. Щоб «дістатися» до мінімуму помилки, потрібно «рухатися» в сторону, протилежну градієнту, тобто, на підставі кожної групи правильних відповідей, додавати до кожної ваги $w_{i,j}$

$$\Delta w_{i,j} = -\eta \frac{\partial E}{\partial w_{i,j}},$$

де $0 < \eta < 1$ – множник, що задає швидкість «руху».

Похідна визначається таким чином. Нехай спочатку $j \in Outputs$, тобто потрібна вага входить в нейрон останнього рівня. Спочатку відзначимо $w_{i,j}$, що впливає на вихід мережі лише як частина суми $S_j = \sum_i w_{i,j} x_i$, де сума береться по входах j-го вузла. Тому

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial S_j} \frac{\partial S_j}{\partial w_{i,j}} = x^i \frac{\partial E}{\partial S_j}$$

Аналогічно, S_j впливає на загальну помилку тільки в рамках виходу j-го вузла o_j (вихід всієї мережі). Тому

$$\partial\partial ES_j = \partial\partial Eo_j \partial\partial oS_{jj} = (\partial\partial o_j 12 \sum_{k\in Outputs}(t_k - o_k)2) (\partial f \_\_\_\_\_ \partial S(S) |_{S=S_j}) =$$

$$= (1\ \partial \quad {}^2)\ (o_j(1-o_j))2\alpha = -2\alpha o_j(1-o_j)(t_j-o_j).$$

$$(t_j - o_j)\ 2\ \partial o_j$$

де $f(S)$ – відповідна сигмоїда, в даному випадку – експоненціальна.

$$\partial(1 + \partial Se{-}2\alpha S) = (1 + e{-}12\alpha S)2 \times \partial(1 + \partial Se{-}2\alpha S)$$

$$= \frac{-1}{(1+e^{-2\alpha S})^2} \times (-2\alpha e^{-2\alpha S}) =$$

$$= (1 \frac{2\alpha e^{-2\alpha S}}{} \frac{1 + e^{-2\alpha S}}{} \frac{1}{} + e^{-2\alpha S})2$$

$$= ((1+e^{-2\alpha S})2 - (1+e^{-2\alpha S})2) \times 2\alpha$$

$$= 2\alpha(f(S) - f^2(S))$$

Якщо j-й вузол розташовується не на останньому рівні, то у нього є виходи, позначимо їх через Children (j). В цьому випадку

$$\frac{\partial E}{\partial S_j} = \sum_{k\in Children(j)} \frac{\partial E}{\partial S_k} \frac{\partial S_k}{\partial S_j},$$

$$\frac{\partial S_k}{\partial S_j} = \frac{\partial S_k}{\partial o_j}\frac{\partial o_j}{\partial S_j} = w_{j,k}\frac{\partial o_j}{\partial S_j} = 2\alpha w_{j,k} o_j(1 - o_j).$$

Тут $\frac{\partial E}{\partial S_k}$ – це в точності аналогічна поправка, але обчислена для вузла наступного рівня. Будемо позначати її через δ_k – від Δ_k вона відрізняється відсутністю множника$(-\eta x_{i,j})$. Обчислення поправки для вузлів останнього рівня і визначення поправки для вузла нижчого рівня через поправки вищого дало назву алгоритму зворотного поширення помилки (backpropagation).

2.3.6 Навчання RNN

Навчання RNN аналогічно навчанню звичайної нейронної мережі. Використовується алгоритм зворотного поширення помилки (backpropagation) з невеликими змінами. Оскільки одні й ті ж параметри використовуються на всіх часових етапах в мережі, градієнт на кожному виході залежить не тільки від розрахунків поточного кроку, але і від попередніх часових кроків. Наприклад, щоб обчислити градієнт при t = 4, нам потрібно було б «поширити помилку» на 3 кроки і підсумовувати градієнти. Цей алгоритм називається «алгоритмом зворотного поширення помилки крізь час» (Backpropagation Through Time, BPTT). Рекурентні нейронні мережі, які пройшли навчання з BPTT, мають недолік з визначення довгострокових залежностей (наприклад, залежності між кроками, які знаходяться далеко один від одного) через загасання/виснаження градієнта. Щоб вирішити цю проблему, існує

певний механізм і розроблено спеціальні архітектури PNN (наприклад LSTM).

3 РОЗРОБКА АЛГОРИТМУ

3.1 Огляд технологій

TensorFlow – відкрита програмна бібліотека для машинного навчання, розроблена компанією Google для вирішення задач побудови і тренування нейронної мережі з метою автоматичного знаходження та класифікації образів, досягаючи якості людського сприйняття [20]. Застосовується для досліджень і розробки власних продуктів Google. Основний API для роботи з бібліотекою реалізований для Python, також існують реалізації для C Sharp, C ++, Haskell, Java, Go і Swift.

Фреймворк є продовженням закритого проекту DistBelief. Спочатку TensorFlow була розроблена командою Google Brain для внутрішнього використання в Google, в 2015 році система була переведена у вільний доступ з відкритою ліцензією Apache 2.0.

Закрита система машинного навчання DistBelief розроблялася Google Brain для внутрішніх проектів з 2011 року для роботи з нейронними мережами глибокого навчання. Вона стала використовуватися в багатьох дослідницьких і комерційних проектах групи фірм холдингу Alphabet. Після успіху DistBelief, фірма Google вирішила вивести проект на новий рівень, і для рефакторінга виділила групу з кількох розробників, в яку увійшов Джефф Дін; метою групи було спрощення і оптимізація кодів бібліотеки, збільшення надійності та зручності користування. Нова бібліотека отримала назву TensorFlow. У 2013 році до проекту приєднався Джеффрі Хінтон – учений, під керівництвом якого в 2009 році був

створений метод узагальненого зворотного поширення помилки і ряд інших поліпшень, що дозволили істотно поліпшити точність нейронних мереж (що призвело, зокрема, до зниження похибки в розпізнаванні мови на 25 %).

TensorFlow 9 листопада 2015 року було відкрито для вільного доступу. TensorFlow є системою машинного навчання Google Brain другого покоління. У той час як еталонна реалізація працює на одиничних пристроях, TensorFlow може працювати на багатьох паралельних процесорах, як CPU, так і GPU, спираючись на архітектуру CUDA для підтримки обчислень загального призначення на графічних процесорах. TensorFlow доступна для 64-розрядних Linux, macOS, Windows, і для мобільних обчислювальних платформ, у тому числі Android та iOS.

Обчислення TensorFlow виражаються у вигляді потоків даних через граф станів. Назва TensorFlow походить від операцій з багатовимірними масивами даних, які також називаються «тензорами». У червні 2016 року Джефф Дін з Google відзначив, що до TensorFlow зверталися 1500 репозиторіїв на GitHub, і тільки 5 з них були від Google.

У травні 2016 року Google повідомила про застосування для задач глибинного навчання апаратного прискорювача власної розробки – тензорного процесора (TPU) – спеціалізованої інтегральної схеми, адаптованої під завдання для TensorFlow, що забезпечує високу продуктивність в арифметиці зниженої точності (наприклад, для 8-бітної архітектури) і спрямованої скоріше на застосування моделей, ніж на їх навчання.

Повідомлялося, що після використання TPU у власних завданнях Google з обробки даних вдалося домогтися на порядок кращих показників продуктивності на ват витраченої енергії.

3.1.1 Pandas

Pandas – програмна бібліотека мовою Python для обробки і аналізу даних. Робота pandas з даними будується поверх бібліотеки NumPy, що є інструментом нижчого рівня. Надає спеціальні структури даних і операції для маніпулювання числовими таблицями і часовими рядами. Назва бібліотеки походить від економетричного терміна «панельні дані» (Panel data), використовуваного для опису багатовимірних структурованих наборів інформації. Pandas поширюється під новою ліцензією BSD.

Основна область застосування – забезпечення роботи в рамках середовища Python не тільки для збору і очищення даних, але й для задач аналізу та моделювання даних.

Пакет перш за все призначений для очищення і первинної оцінки даних за загальними показниками, наприклад середнім значенням; набори даних типів DataFrame і Series застосовуються в якості вхідних в більшості модулів аналізу даних і машинного навчання (SciPy, Scikit-Learn).

3.1.2 Matplotlib

Matplotlib – бібліотека мовою програмування Python для візуалізації даних двовимірної (2D) графікою (3D графіка також підтримується). Одержувані зображення можуть бути використані в якості ілюстрацій в публікаціях [22].

Matplotlib написаний і підтримується в основному Джоном Хантером і поширюється на умовах BSD-подібної ліцензії. Генеровані в різних форматах зображення можуть бути використані в інтерактивній графіці, наукових публікаціях, графічному інтерфейсі користувача, вебдодатках, де потрібна побудова діаграм.

Остання стабільна версія 2.1.1 потребує Python версії 2.7 або від 3.4 і вище і версію NumPy від 1.7.1 і вище.

Бібліотека Matplotlib побудована на принципах ООП, але має процедурний інтерфейс pylab, який надає аналоги команд MATLAB.

Matplotlib є гнучким, легко конфігурованим пакетом, який разом з NumPy, SciPy і IPython надає можливості, подібні MATLAB. В даний час пакет працює з декількома графічними бібліотеками, у тому числі wxWindows і PyGTK. Пакет підтримує багато видів графіків і діаграм:

а) графіки (line plot);

б) діаграми розкиду (scatter plot);

в) стовпчасті діаграми (bar chart) і гістограми (histogram);

г) кругові діаграми (pie chart);

д) стовбур-лист діаграми (stem plot);

є) контурні графіки (contour plot);

ж) поля градієнтів (quiver);

з) спектральні діаграми (spectrogram);

і) користувач може вказати осі координат, додати написи і пояснення, використовувати логарифмічну шкалу або полярні координати.

3.1.3 NumPy

NumPy – бібліотека з відкритим вихідним кодом для мови програмування Python [23]. Можливості NumPy:

а) підтримка багатовимірних масивів (у тому числі матриць);

б) підтримка високорівневих математичних функцій, призначених для роботи з багатовимірними масивами.

Математичні алгоритми, реалізовані на інтерпретованих мовах (наприклад, Python), часто працюють набагато повільніше тих же алгоритмів, реалізованих на компільованих мовах (наприклад, Фортран, Сі, Java). Бібліотека NumPy надає реалізації обчислювальних алгоритмів (у вигляді функцій і операторів), оптимізовані для роботи з багатовимірними масивами. В результаті будь-який алгоритм, який може бути виражений у вигляді послідовності операцій над масивами (матрицями) і реалізований з використанням NumPy, працює так само швидко, як еквівалентний код, що виконується в MATLAB.

Розглянемо приклад роботи з NumPy в інтерактивній оболонці IPython.

Запуск Python з командного рядка:

```
x = linspace(0, 2*pi, 100)
y = sin(x) plot(x, y, 'ro-
') show()
```

В результаті роботи сценарію бібліотека Matplotlib створить графік, зображений на рис. 3.1.

3.1.4 IPython notebook

IPython – інтерактивна оболонка для мови програмування Python, яка надає розширену інтроспекцію, додатковий командний синтаксис, підсвічування коду і автоматичне доповнення [24]. Є компонентом пакетів програм SciPy і Anaconda.

IPython дозволяє здійснювати неблокуючу взаємодію з Tkinter, GTK, Qt і WX. Стандартна бібліотека Python включає лише Tkinter. IPython може

інтерактивно керувати паралельними кластерами, використовуючи асинхронні статуси зворотних викликів та/або MPI. IPython може використовуватися як заміна стандартної командної оболонки операційної системи, особливо на платформі Windows, можливості оболонки якої обмежені. Поведінка за замовчуванням схожа на поведінку оболонок UNIX-подібних систем, але той факт, що робота відбувається в оточенні Python, дозволяє домагатися більшої настроюваності і гнучкості.

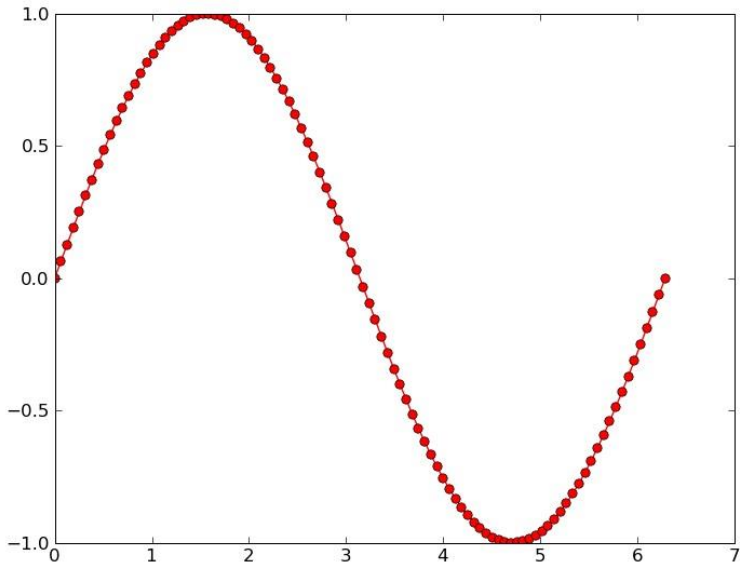


Рисунок 3.1 – Результат роботи тестового коду

Починаючи з версії 4.0, монолітний код був розбитий на модулі, і незалежні від мови модулі були виділені в окремий проект Jupyter. Найбільш відомою веб-оболонкою для IPython є Jupyter Notebook (раніше відомий як IPython Notebook), що дозволяє об'єднати код, текст і діаграми, і поширювати їх для інших користувачів.

3.1.5 Keras

Keras – відкрита нейромережева бібліотека, написана мовою Python

[25]. Вона являє собою надбудову над фреймворками Deeplearning4j, TensorFlow і Theano. Націлена на оперативну роботу з мережами глибинного навчання, при цьому спроектована так, щоб бути компактною, модульною та розширюваною. Вона була створена як частина дослідницьких зусиль проекту ONEIROS, а її основним автором і підтримувачем є Франсуа Шолль, інженер Google.

Планувалося що Google буде підтримувати Keras в основній бібліотеці TensorFlow, однак Шолль виділив Keras в окрему надбудову, так як згідно з концепцією Keras є скоріше інтерфейсом, ніж наскрізною системою машинного навчання. Keras надає високорівневий, більш інтуїтивний набір абстракцій, який робить простим формування нейронних мереж, незалежно від використовуваної в якості обчислювального бекенду бібліотеки наукових обчислень. Microsoft працює над додаванням до Keras і низькорівневих бібліотек CNTK.

3.2 Огляд моделі

В роботі використовується схема нейронної мережі, наведена на рис. 3.2. Модель може бути розділена на 3 рівні. Перший рівень згортки, далі йде рекурентний шар і повноз'єднана нейронна мережа, яка відповідає за класифікацію.

Перший рівень необхідний для попередньої обробки даних, добування прихованих ознак і взаємозв'язків у вихідних даних. Рівень складається з 3х шарів згортки першого ступеня, максимального пулінгу, і пакетної нормалізації.

Далі йде 2 рекурентних шари, які необхідні для аналізу послідовних даних. На вхід надходить оброблена інформація з першого рівня.

Останній рівень складається з повноз'єднаної мережі, яка аналізує останній вихід рекурентного шару цілком, і потім виконує класифікацію за допомогою softmax функції збудження.

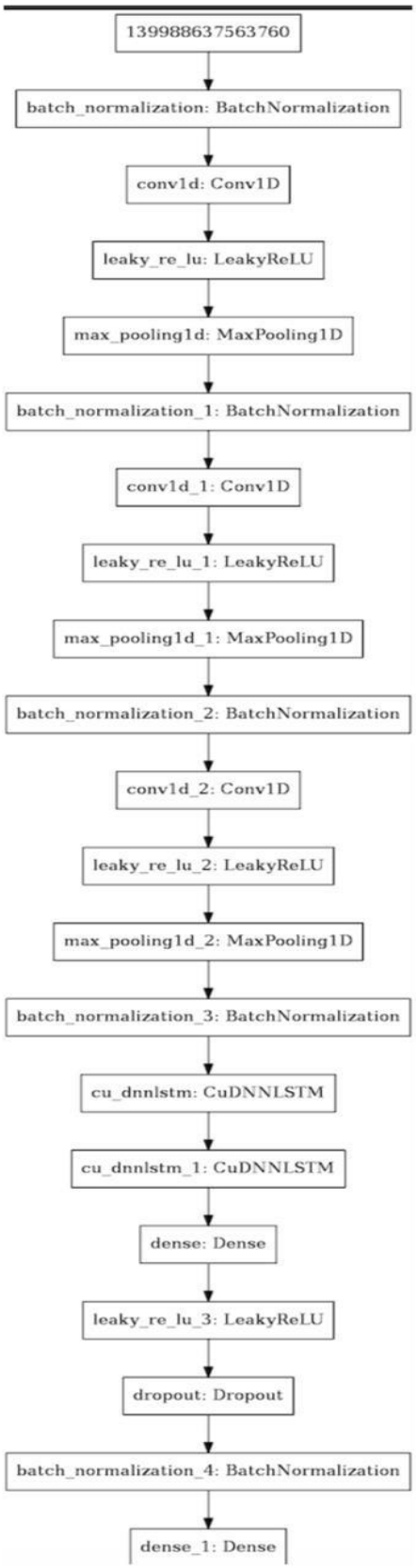


Рисунок 3.2 – Схема нейронної мережі

3.3 Реалізація алгоритму

Программа написана з використанням мови Python. Модель нейронної мережі сконструйована за допомогою фреймворка tensorflow. Обробка та аналіз даних виконаний за допомогою бібліотек sklearn та pandas. Весь код програми міститься в одному файлі ipython notebook.

Увесь пайплайн можна умовно розділити на 3 частини:

а) обробка та підготовка даних;

б) будування моделі нейронної мережі;

в) тренування мережі та її валідація.

3.3.1 Обробка та підготовка даних

Даних досить багато, тому неможливо було завантажити весь датасет одразу, бо він зайняв би всю оперативну пам'ять. Тому дані зчитувалися пакетами (Лістинг 3.1).

Лістинг 3.1 – Загрузка даних

```
def read_batch(start=0, end=TRAIN_SAMPLES):
    sample_list = os.listdir(TRAIN_PATH)     labels = []     data = []     for sample in sample_list:         sample_data = pd.read_csv(os.path.join(TRAIN_PATH, sample), nrows=end, skiprows=range(1, start))         sequences = get_sequences(sample_data['drawing'].values)         sample_labels = sample_data['word'].replace(' ', '_', regex=True)         data.append(sequences)     labels.extend(sample_labels)      encodings = label_encoder.fit_transform(labels)     data = np.concatenate(data, axis=0)
    one_hot = to_categorical(encodings)
    return data, one_hot
```

Інформація про штрихи була записана в файлах формату csv. Так що, в першу чергу, ми завантажуємо її. Потім ці дані необхідно перетворити в набір послідовностей, розділених в часі, які ми будемо подавати на вхід нашої моделі (про перетворення написано нижче). Далі ми замінюємо всі

пробіли в найменуванні категорій на нижнє підкреслення, щоб уникнути помилок, пов'язаних з інтерпретацією рядків. Потім ми конвертуємо строкове представлення категорій в числовий формат коду.

На виході мережі ми будемо мати вектор, який зберігає імовірнісний розподіл приналежності екземпляру до кожної категорії, тому мітки нам необхомо мати у відповідному вигляді. Таким чином ми перетворимо наші мітки до числових векторів де 1 буде стояти за індексом відповідної категорії, а інші значення будуть заповнені 0. Такий формат називається one-hot кодування (рис. 3.3).

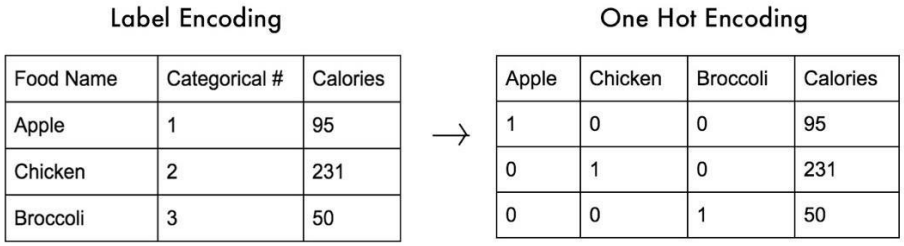


Рисунок 3.3 – Приклад one-hot кодування

Лістинг 3.2 – Кодування міток def

```
get_sequences(string_data):
   simplified_images = [ast.literal_eval(lst) for lst in string_data]
del string_data     strokes = []
   for image in simplified_images:
      stroke = [(xi, yi, i) for (i, (x, y)) in enumerate(image)
for xi, yi in zip(x, y)]         stroke = np.asarray(stroke)


      stroke[:,2] = [1] + np.diff(stroke[:,2]).tolist()
stroke[:,2] += 1


      stroke = pad_sequences([stroke], maxlen=MAX_STROKE_LENGTH,
padding='post')
```

```
        stroke = np.squeeze(stroke)
    strokes.append(stroke)        del
    stroke
```

Інформація про штрихи записана в текстовому вигляді, так що спочатку ми конвертуємо тексти в масиви, які містять координати кожного штриха. Потім все штрихи для кожного зображення ми конкатенуємо в довгу послідовність, спеціальним символом позначаючи початок і кінець кожного штриха. Далі нам необхідно привести всі послідовності до однакової довжини для подачі на вхід мережі. Це робиться завдяки заповненню менших послідовностей нулями з кінця:

```
test_x, test_y = read_batch(start=0, end=TEST_SAMPLES).
```

Далі резервуємо перші 50 тисяч экземплярів для валідації.

3.3.2 Будування моделі нейронної мережі

Спочатку імпортуємо необхідні модулі. Бібліотека для машинного навчання keras надає високорівневий інтерфейс, який дозволяє збирати моделі з блоків, що значно спрощує розробку. Перевіряємо систему на наявність графічних карт, які дозволяють в значній мірі прискорити навчання, за рахунок распаралелювання операцій (Лістинг 3.3).

Лістинг 3.3 – Використання бібліотеки keras

```
from tensorflow.keras.layers import LSTM, Dense, Conv1D, LeakyReLU, BatchNormalization, MaxPool1D, Dropout
from tensorflow.keras.models import Model
GPUs = get_available_gpus()
if len(GPUs) > 0:
```

```
    print('GPUs: ', GPUs)     from tensorflow.keras.layers import
    CuDNNLSTM as LSTM from tensorflow.keras.models import
    Sequential
    from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau,
EarlyStopping from tensorflow.keras.optimizers
    import Adam
```

Далі ми збираємо нашу модель, за допомогою складання блоків згортки, нормалізації, функцій збудження у правильному порядку (Лістинг 3.4).

Лістинг 3.4 – Збирання моделі

```
model = Sequential()
model.add(BatchNormalization(input_shape=(None, 3)))
model.add(Conv1D(64, (5), padding='same', activation=None))
model.add(LeakyReLU()) model.add(MaxPool1D())
model.add(BatchNormalization())

model.add(Conv1D(128, (5), padding='same', activation=None))
model.add(LeakyReLU()) model.add(MaxPool1D())
model.add(BatchNormalization())

model.add(Conv1D(256, (5), padding='same', activation=None))
model.add(LeakyReLU()) model.add(MaxPool1D())
model.add(BatchNormalization())

model.add(LSTM(256, return_sequences=True)) model.add(LSTM(256,
return_sequences=False))

model.add(Dense(512, activation=None))
model.add(LeakyReLU()) model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Dense(test_y.shape[1], activation='softmax'))
```

Потім ми визначаємо необхідні callback функції, які використовуються для управління процесом навчання, збору проміжних результатів (Лістинг 3.5). Серед них були використані:

а) ModelCheckpoint – зберігає проміжні стани моделі з найменшим показником помилки класифікації;

б) ReduceLROnPlateau – зменшує швидкість навчання, якщо помилка класифікації перестає падати;

в) EarlyStopping – зупиняє навчання моделі, якщо помилка класифікації перестає зменшуватися.

Лістинг 3.5 – Визначення callback функцій def

```
default_callbacks(model_name=MODEL_NAME):
checkpoint = ModelCheckpoint(filepath=model_name,
save_best_only=True,
verbose=1) plateo = ReduceLROnPlateau(factor=0.5, verbose=1,
    patience=4) early = EarlyStopping(monitor="val_loss",
    mode="min", patience=6, verbose=1)
      return [checkpoint, plateo, early]
```

Тепер переходимо безпосередньо до навчання мережі. Спочатку створюємо callback функції, визначаємо оптимізатор, функцію помилки, метрику оцінювання (в даному випадку використовується точність як метрика оцінювання – співвідношення правильно класифікованих малюнків до неправильних). Потім компілюємо модель з даними параметрами і починаємо цикл навчання. Оскільки даних багато, то вони завантажувались пакетами по 50000 екземплярів. Всього пакетів – 152. З кожною ітерацією тренування, швидкість навчання зменшувалася певним чином (Лістинг 3.6). Це зроблено для того, щоб процес оптимізації мережі проходив найбільш гладко.

Лістинг 3.6 – Навчання мережі

```
histories = []
callbacks = default_callbacks(model_name='best1.hdf5')
adam = Adam(lr=lr, decay=lr / 5) loss =
'categorical_crossentropy' metrics = ['accuracy']
model.compile(optimizer=adam, metrics=metrics, loss=loss)
for i in range(0,152):    print('\n\nBATCH: ', i)
   lr = 0.001#(INIT_LR / ((epoch + 1)*2 + i/9))
print('lr: ', lr)
   train_x, train_y = read_batch(start=TEST_SAMPLES + TRAIN_SAMPLES*i,
end=TRAIN_SAMPLES)
   loss_history = model.fit(train_x, train_y, batch_size=BATCH_SIZE,
validation_data=(test_x, test_y),
          epochs=1,
callbacks=callbacks)
histories.append(loss_history)
```

3.4 Результати роботи програми

В ході навчання мережі була досягнута максимальна точність 90.3 % на навчальній вибірці, 70.5 % – на тестовій (рис. 3.4, 3.5).

Модель навчалася протягом приблизно 9 годин.

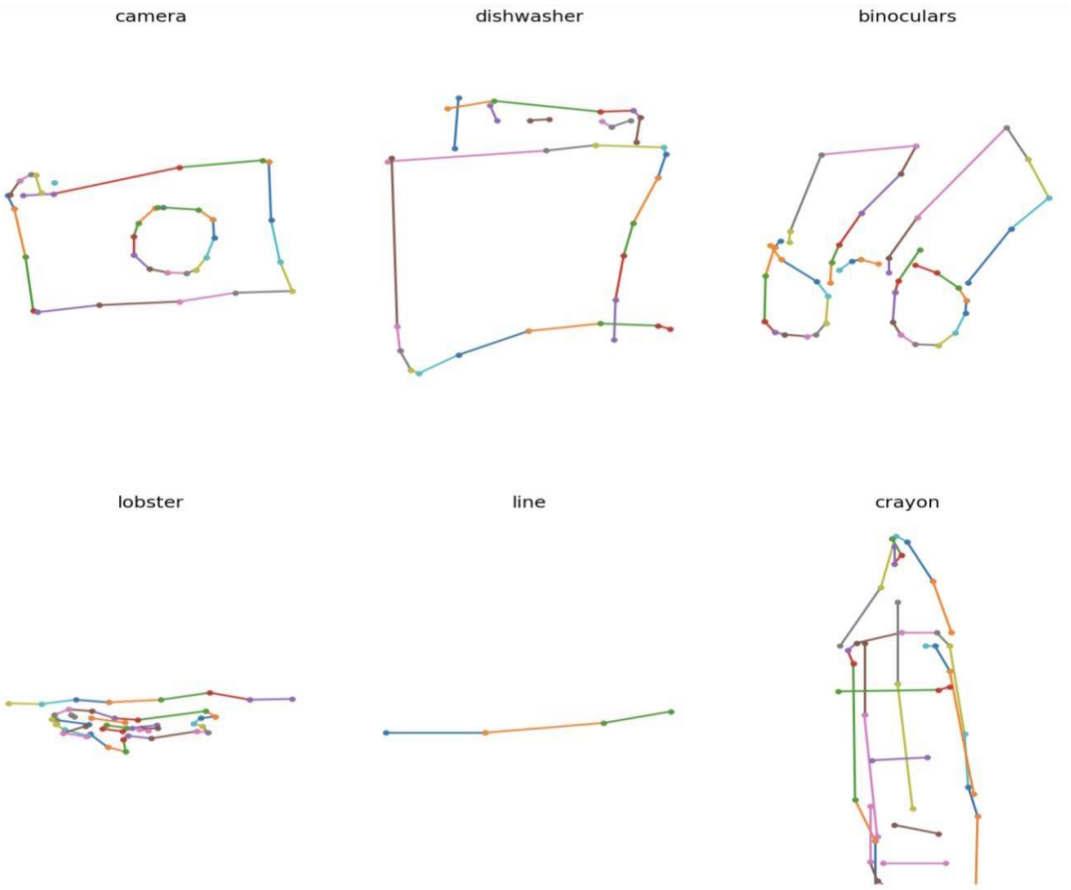


Рисунок 3.4– Приклад роботи нейронної мережі

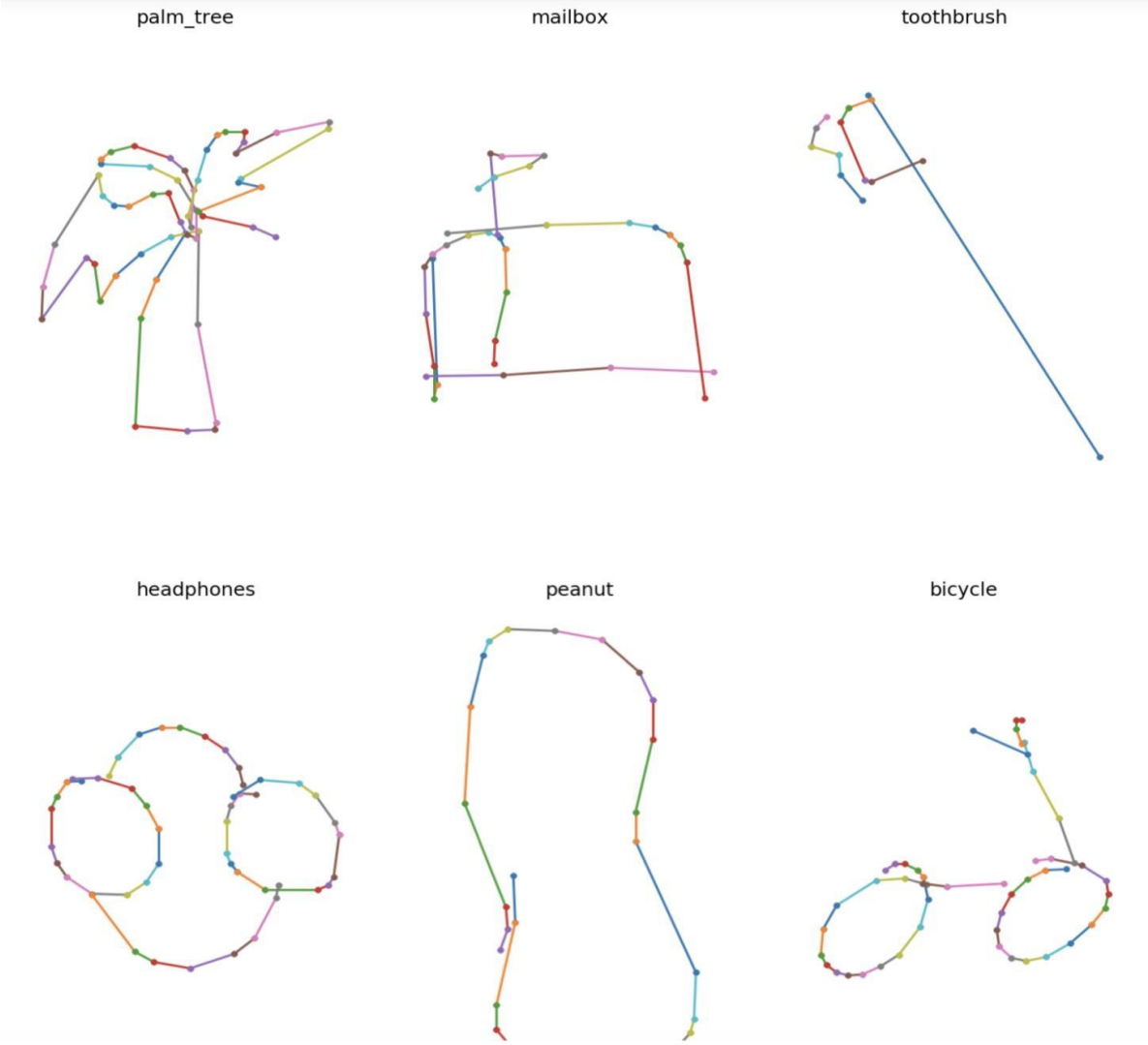


Рисунок 3.5 – Приклад роботи нейронної мережі

ВИСНОВКИ

В магістерській атестаційній роботі отримано такі результати:

- запропоновано модель відношень порядку в структурах даних і компьютингу, яка формує ефективність паралельного управління алгоритмами обробки великих даних за рахунок суперпозиціонування кінцевої множини дискретних станів;
- розроблено структурну модель взаємодії кубітних покриттів логічних функцій і похідних компонентів, орієнтованих на синтез і аналіз

цифрових систем з метою отримання перевіряльних тестових послідовностей для поодиноких константних несправностей з метою зменшення часу проектування і тестування обчислювальних пристроїв;

- визначено прості Х-функцій від кінцевої кількості змінних, які характеризуються відсутністю мінімізації та наявністю властивостей тестопридатності, що дає можливість синтезувати цифрові пристрої, технологічні для вирішення задач тестування, моделювання та діагностування;

- запропоновано tree-driven ATPG процесор і структури даних для обчислення кубітних булевих похідних, представлений двійковим деревом-графом xor-елементів для паралельної обробки частин кубітного покриття;

- запропоновано кубітні методи синтезу тестів для логічних функцій, які характеризуються квадратичною і лінійною обчислювальною складністю алгоритмів для генерування перевіряльних послідовностей;

- розроблено модель рекурентної нейронної мережі з довгою короткостроковою пам'яттю. В якості вихідних шарів використовувались однорозмірні згорткові шари. В якості функцій активації для прихованих шарів була обрана Leaky Relu, а для вихідного шару – softmax;

- розроблено алгоритм роботи нейронної мережі для розпізнавання образів на прикладі розпізнавання людських малюнків;

- здійснено програмну реалізацію алгоритму роботи нейронної мережі для розпізнавання людських малюнків;

- здійснено навчання РНС з використанням бази зображень отриманих з сервісу "Quick Draw". В якості вхідних даних задіяна

інформація про координати штрихів зроблених користувачами, розділених в часі.

Найвища точність розпізнання зображень, яка була досягнута на тестовому наборі – 70.5%.

Для того підвищення точності можна збільшити складність моделі мережі та час навчання.

Отримані результати дозволяють зрозуміти, що потенціал нейронних мереж практично безмежний, і хоча час навчання може бути досить тривалим, часові витрати окупаються позитивними результатами.

Отримані результати опубліковано в роботах [1-6, 12,13].

ПЕРЕЛІК ПОСИЛАНЬ

1. Hahanov V. Cyber Physical Computing for IoT-driven Services. Springer International Publishing. Switzerland. 2018. [Part 4. Qubit Description of the Functions and Structures for Service Computing Synthesis [Text] / I. Hahanov, I. Iemelianov, M. Liubarskyi, V. Hahanov. – P. 71-93]. [Part 6. QuaSim – Cloud Service for Quantum Circuits Simulation [Text] / *I. Hahanov*, Bani Amer T., I. Iemelianov, M. Liubarskyi, V. Hahanov – P. 135-147].

2. Liubarskyi M. IoT Computing Evolution [Text] / M. Liubarskyi, *I. Hahanov* // Матеріали XXI Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI ст.». – 25-27 квітня 2017. – Ч. 5. – С.4647.

3. Hahanov I. Qubit deductive fault simulation of black box functionalities [Text] / *I. Hahanov*, A. Hahanova, V. Hahanov, E. Litvinova, S.

Chumachenko, M. Liubarskyi // Proc. of 5th Prague Embedded Systems Workshop. June 29-30, 2017. Roztoky u Prahy, Czech Republic. P.83-91.

4. Hahanov V. Qubit Minimization of Boolean Functions [Text] / Vladimir Hahanov, Ka Lok Man, Mykhailo Liubarskyi, *Ivan Hahanov*, Svetlana Chumachenko // Proc. of the International MultiConference of Engineers and Computer Scientists. – Hong Kong. – March 14-16, 2018. – Vol II IMECS 2018. – Pp. 690-695.

5. Hahanov V. Quantum Deductive Fault Simulation [Text] / V. Hahanov, W. Gharibi, M. Liubarskyi, E. Litvinova, M. Gharibi, S. Chumachenko, *I. Hahanov*, and A. Hahanova // Proc. of Int'l Conf. Modeling, Sim. and Vis. Methods (MSV'18) in The 2018 World Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE2018) . – Jul 30-Aug 02, 2018. – Las Vegas, USA. – P. 10-16.

6. W. Gharibi, D. Devadze, V. Hahanov, E. Litvinova and *I. Hahanov*, "Qubit Test Synthesis Processor for SoC Logic," 2019 IEEE East-West Design & Test Symposium (EWDTS), Batumi, Georgia, 2019, pp. 1-5.

7. Dhare V. Defect characterization and testing of QCA devices and circuits: A survey / V. Dhare, U. Mehta // 2015 19th International Symposium on VLSI Design and Test, Ahmedabad. – 2015.– P. 1-2.

8. Abdollahi A. Analysis and Synthesis of Quantum Circuits by Using Quantum Decision Diagrams / A. Abdollahi, M. Pedram // Proceedings of the Design Automation & Test in Europe Conference.– Munich.– 2006.– P. 1-6.

9. Dhare V. A Simple Synthesis Process for Combinational QCA Circuits:
Q-Synthesizer / V. Dhare, U. Mehta // 2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID).– Delhi, NCR, India.– 2019.– P. 498-499.

10. Mondal B. Generator for Test Set Construction of SMGF in Reversible Circuit by Boolean Difference Method / B. Mondal, D. K. Kole, D.

K. Das and H. Rahaman // 2014 IEEE 23rd Asian Test Symposium, Hangzhou.– 2014.– P. 68-73.

11. Raik J. Sequential circuit test generation using decision diagram models / J. Raik, R. Ubar // Design, Automation and Test in Europe Conference and Exhibition, 1999. Proceedings (Cat. No. PR00078).– Munich, Germany.– 1999.– P. 736-740.

12. Любарский М.М. Синтез и анализ логических Х-функций [Text] / М.М. Любарский, В.Г. Абдуллаев, В.И. Хаханов, С.В. Чумаченко, Е.И.

Литвинова, *И.В. Хаханов* // Радиоэлектроника и информатика. – 2018. – №2. – С. 18-28.

13. Любарський М.М. Quantum-driven архітектури для паралельного проектування цифрових систем / М.М. Любарський, *І.В. Хаханов*, В.І.

Хаханов, С.В. Чумаченко // Радиоэлектроника и информатика.– 2018.– № 4(83).– С. 52-64.

14. Бодянский Е.В. Искусственные нейронные сети: архитектуры, обучение, применение / Е.В. Бодянский, О.Г. Руденко. – 2004.– 300 с.

15. Хайкин С. Нейронные сети: полный курс. М.: ООО «И.Д.Вильямс». 2-е издание. 2016. 1104 с.

16. Goodfellow I. Deep Learning (Adaptive Computation and Machine

Learning series) / I. Goodfellow, Y. Bengio, and A. Courville. – The MIT PressCambridge, Massachusetts London, England. – 2016. – 800 p.

17. Deng Li. Deep Learning: Methods and Applications / Li Deng, Dong

Yu // Foundations and TrendsR in Signal Processing.– Vol. 7.– No. 3–4.– 2013.– P. 197–387.

18. Нильсон M. Neural Networks and Deep Learning. 2019. URL: http://neuralnetworksanddeeplearning.com/chap1.html (Дата звернення: 30.11.2019)

19. A. Graves, J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural networks 18 (5-6), 2005. P. 602-610.

20. TensorFlow URL: https://www.tensorflow.org (Дата звернення: 30.11.2019)

21. URL: https://pandas.pydata.org (Дата звернення: 30.11.2019)

22. URL: https://matplotlib.org (Дата звернення: 30.11.2019)

23. URL: https://numpy.org (Дата звернення: 30.11.2019)

24. URL: https://ipython.org/notebook.html (Дата звернення: 30.11.2019)

25. URL: https://keras.io (Дата звернення: 30.11.2019)