Міністерство освіти і науки України
Державний університет «Одеська політехніка»
Інститут штучного інтелекту та робототехніки
Кафедра «Комп'ютерні системи»

Баєв Вячеслав Федорович,
студент групи УК-161

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Дослідження методів підвищення ефективності навчання нейронних мереж

Спеціальність: 123 – "Комп'ютерна інженерія"
Спеціалізація: Спеціалізовані комп'ютерні системи

Керівник:
Стрельцов Олег Васильович,
кандидат техн. наук, доцент

Одеса — 2021

Міністерство освіти і науки України
Державний університет «Одеська політехніка»

Інститут штучного інтелекту та робототехніки
Кафедра комп'ютерних систем

Рівень вищої освіти другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
(шифр і назва)
Спеціалізація / освітня програма Спеціалізовані ком'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____

“____” _____ 2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Баєву Вячеславу Федоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Дослідження методів підвищення ефективності навчання нейронних мереж
_____
Керівник роботи Стрельцов Олег Васильович, кандидат техн. наук, доцент ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ректора ОНПУ від “___” _____ року № _____
2. Зміст роботи
Аналітичний огляд, Огляд нейронних мереж, Розпізнавання образів і класифікація, Прогнозування, Етапи розв’язання задач, Класифікація, Функції активації, Нейрон, Синапс, Глибинна нейронна мережа, Сучасні методи підвищення ефективності роботи нейронних мереж, Побудова моделі та методу навчання нейронної мережі, Визначення архітектури та елементів для нейронної мережі, Зміна ваг для покращення роботи мережі, Зміна функції активації для покращення роботи мережі, Тестування розробленої нейронної мережі
3. Перелік ілюстративного матеріалу
Рисунок загальної архітектури нейронної мережі, результати тестування, таблиця, презентація 10 слайдів

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

5. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналітичний огляд		виконано
2	Огляд нейронних мереж		виконано
3	Прогнозування		виконано
4	Функції активації		виконано
5	Сучасні методи підвищення ефективності роботи нейронних мереж		виконано
6	Побудова моделі та методу навчання нейронної мережі		виконано
7	Визначення архітектури та елементів для моделі нейронної мережі		виконано
8	Побудова моделі нейронної мережі		виконано
9	Зміна ваг для покращення роботи мережі		виконано
10	Зміна функції активації для покращення роботи мережі		виконано
11	Тестування розробленої моделі нейронної мережі		виконано
12			
13			
14			

Здобувач вищої освіти _____ Баєв В.Ф. .

Керівник роботи _____ Стрельцов О.В.

ЗМІСТ

ВСТУП……..5
1 АНАЛІТИЧНИЙ ОГЛЯД 8
1.1 Огляд нейронних мереж…… 8
1.2 Розпізнавання образів і класифікація..... 11
1.3 Прогнозування..... 11
1.4 Етапи розв’язання задач 12
1.5 Класифікація..... 17
1.6 Функції активації 20
1.7 Нейрон..... 24
1.8 Синапс 25
1.9 Глибинна нейронна мережа 26
1.10 Сучасні методи підвищення ефективності роботи нейронних мереж……....28
1.11 Висновки 30
2 ПОБУДОВА МОДЕЛІ ТА МЕТОДУ НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ 31
2.1 Визначення архітектури та елементів для нейронної мережі 31
2.2 Побудова моделі нейронної мережі 34
2.3 Зміна ваг для покращення роботи мережі 39
2.4 Зміна функції активації для покращення роботи мережі..... 40
2.5 Висновки 42
3 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ 43
3.1 Тестування звичайної моделі нейронної мережі 43
3.2 Тестування моделі нейронної мережі після зміни ваг 46
3.3 Тестування моделі нейронної мережі після зміни функції активацій... 47
3.4 Висновки 49
ВИСНОВКИ...……51
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ…..……52
Додатки А-Г 55-63

ВСТУП

Актуальність дослідження: У сучасному світі нейронні мережі використовуються дуже активно багатьма великими, і не тільки, компаніями. Щоразу коли людина вводить запит у пошуковій системі, робить фотографію і т.д. починають працювати нейронні мережі. Вони проводять певний аналіз, щоб видати найкращий результат. І в нашому швидкодіючим світі дуже важливо, щоб нейронні мережі працювали швидко та чітко. В даний час дуже стрімко розвивається такий напрямок як "нейронні мережі". І все частіше великі фірми, підприємства та інші учасники сфери ІТ впроваджують в свої продукти Штучні нейронні мережі (штучний інтелект). Такий інтерес до нейронних мереж викликаний тим, що навіть самі сучасні комп'ютери поступаюся по швидкості обробки інформації людському мозку. Адже людський мозок абсолютно відрізняється методами обробки інформації від комп'ютерних методів. Людський мозок має таку здатність, що сам здатний розмістити нейрони таким чином, щоб вони могли виконувати різні завдання одночасно і без усвідомлення в цьому участі самої людини. Наш мозок виконує безліч дій такі як обробку сенсорної інформації, що надходить від органів чуття, пам'ять, увагу, прийняття рішень, управління рухами, емоції, координацію, планування. Одне з основних умінь мозку це розпізнавання образів, яке відбувається в рази швидше, ніж на це здатний самий швидкодіючий комп’ютер. Людський мозок здатний дізнатися, розпізнати знайомий або потрібний йому об'єкт всього за 100-200 мілісекунд, на що комп'ютера на таку ж задачу може знадобитися кілька днів. Нейронні мережі увійшли в практику всюди, де потрібно вирішувати завдання управління, класифікації, прогнозування. Сама суть машинного навчання за допомогою нейронних мереж не так вже й складна і в процесі роботи ви дізнаєтеся про це [1].

Виходячи з цього нейронні мережі стали використовувати в самих різних областях - медицині, фізиці, техніці, бізнесі, геології, космонавтиці. Наприклад додаток для мобільних пристроїв "FaceApp" яке вийшло в 2019 році, що дозволяє побачити себе через рік-два або тридцять років. Додаток

використовує нейронну мережу, яка має здатність до самонавчання і в результаті давати найвірніший результат. За десять днів розробники заробили 1 млн доларів. Їхній додаток в день скачували 1,6 млн разів.

Компанія IBM стоїть біля витоків розвитку II-технологій та нейронних мереж, про що свідчать поява та еволюція IBM Watson. Watson - надійне рішення для великих підприємств, яким потрібно впровадити передові технології глибокого навчання та обробки даних природною мовою у свої системи, спираючись на перевірений багаторівневий підхід до розробки та реалізації II [11].

Архітектура UIMA (Apache Unstructured Information Management Architecture) та програмне забезпечення IBM DeepQA, що лежать в основі Watson, дозволяють інтегрувати до додатків потужні функції глибокого навчання. За допомогою таких інструментів, як IBM Watson Studio, ваше підприємство зможе ефективно перенести II-проекти з відкритим вихідним кодом у робоче середовище з можливістю розгортання та виконання моделей у будь-якому хмарному середовищі.

В ході розробки нейронної мережі є кілька нюансів від якої буде залежати її швидкість її роботи і точність результатів. Такі як коефіцієнт нейрона (в машинному навчанні їх прийнято назвати вагою), функції активації. У зв'язку з цим в цій кваліфікаційної роботі проводиться дослідження вибору ваг і функцій активацій для підвищення ефективності роботи нейронної мережі [2].

Мета кваліфікаційної роботи є підвищення ефективності навчання нейронних мереж. Для того щоб це використати в першу чергу потрібно розуміти як працюють ваги, функції активації і навіщо вони потрібні в нейронних мережах. У багатьох джерелах пишуть, що ваги в нейронних мережах можуть бути будь-якими, що мережа в результаті сама їх підбере, але цей підбір - це час, який витрачає мережа на навчання. З правильними вагами та правильною функцією активації мережа навчатиметься швидше. Існує кілька найвідоміших функцій активацій - це Ступінчаста функція активації,

Лінійна функція активації, Сігмоїда, Гіперболічний тангенс, ReLu. Докладніше про кожну розповідається під час роботи.

Нейронні мережі вже широко використовують у різних сферах життя — розпізнають обличчя (зокрема ловлять злочинців), діагностують хвороби, працюють як голосові помічники. У тому числі зростає їхнє застосування в бізнесі: оцінка ефективності співробітників, схвалення кредиту, чат-боти, управління кол-центрами [7].

У нейронних мережах функція активації нейрона визначає вихідний сигнал, що визначається вхідним сигналом чи набором вхідних сигналів. Звичайна комп'ютерна мікросхема може розглядатися як цифрова мережа функцій активації, які можуть приймати значення ON (1) або OFF (0) в залежності від входу. Це схоже на поведінку лінійного перцептрону у нейронних мережах [8].

Завдання дослідження: аналіз існуючих методів навчання нейронних мереж, удосконалити методи навчання нейронних мереж, протестувати нейрону мережу із запропонованими функціями активації.

Об'єкт дослідження – процес навчання нейронної мережі.

Предмет дослідження – швидкість навчання та зменшення вихідної помилки.

Інноваційність цієї роботи полягає у новому підході до підвищення швидкості навчання нейронних мереж і використанням ваг та функцій активації.

Практична значимість: нейронні мережі в основному застосовуються у розпізнавання образів, області прогнозування, оптимізація, прийняття рішень, аналіз даних. Нейромережі лежать в основі сучасних систем розпізнавання та синтезу мови. Вони застосовуються в деяких системах навігації, будь то промислові роботи або безпілотні автомобілі [14]. Тому виходячи з вище перерахованого швидкодія нейронної мережі дуже важлива складова.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Огляд нейронних мереж

Нейронна мережа (також штучна нейронна мережа, ШНМ) - математична модель, а також її програмне або апаратне втілення, побудована за принципом організації та функціонування біологічних нейронних мереж - мереж нервових клітин живого організму. Це поняття виникло при вивченні процесів, що протікають в мозку, і при спробі змоделювати ці процеси. Після розробки алгоритмів навчання одержувані моделі стали використовувати в практичних цілях: в задачах прогнозування, для розпізнавання образів, в задачах управління та ін.

ШНМ є система з'єднаних і взаємодіючих між собою простих процесорів (штучних нейронів). Такі процесори зазвичай досить прості (особливо в порівнянні з процесорами, використовуваними в персональних комп'ютерах). Кожен процесор подібної мережі має справу тільки з сигналами, які він періодично отримує, і сигналами, які він періодично посилає іншим процесорам. І, тим не менше, будучи з'єднаними в досить велику мережу з керованим взаємодією, такі окремо прості процесори разом здатні виконувати досить складні завдання [3].

На рисунку 1.1 зображена схема простої нейронної мережі. Зеленим кольором позначені вхідні нейрони, блакитним - приховані нейрони, жовтим - вихідний нейрон. У складних нейронних мережах прихованих нейронів може бути набагато більше і вони поділяються на окремі шари (приховані шари).

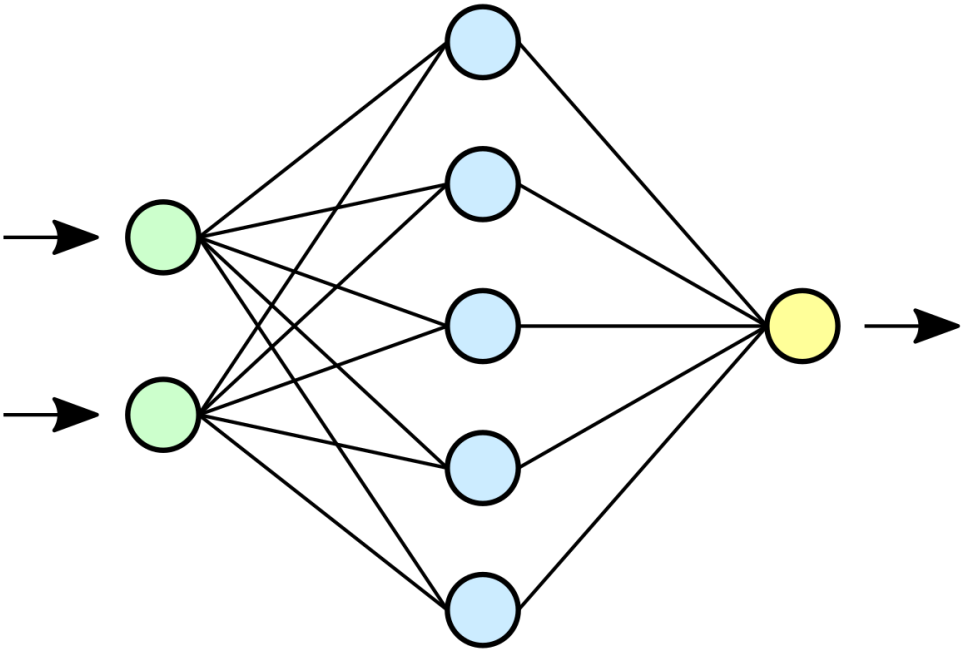


Рисунок 1.1 − Схема простої нейромережі

Нейронні мережі не програмуються в звичному сенсі цього слова, вони навчаються. Можливість навчання - одне з головних переваг нейронних мереж перед традиційними алгоритмами. Технічно навчання полягає в знаходженні коефіцієнтів зв'язків між нейронами. В процесі навчання нейронна мережа здатна виявляти складні залежності між вхідними даними і вихідними, а також виконувати узагальнення. Це означає, що в разі успішного навчання мережа зможе повернути вірний результат на підставі даних, які були відсутні в навчальній вибірці, а також неповних і / або «зашумлених», частково спотворених даних [26].

Ідея нейромережі полягає в тому, щоб зібрати складну структуру із дуже простих елементів. Насправді нейромережа можна зібрати навіть із сірникових коробок: це просто набір нехитрих правил, за якими обробляється інформація. «Штучним нейроном», або перцептроном, називається не якийсь особливий прилад, а лише кілька арифметичних дій [27].

Працює перцептрон простіше нікуди: він отримує кілька вихідних чисел, множить кожне на «цінність» цього числа (про це йдеться трохи пізніше), складає і в залежності від результату видає 1 або -1.

У реальній роботі формули трохи складніше, але принцип залишається тим самим. Перцептрон вміє виконувати лише одне завдання: брати числа та розкладати по двох стопках. Найцікавіше починається тоді, коли таких елементів кілька, адже вхідні числа можуть бути сигналами інших нейронів. Скажімо, один нейрон намагатиметься відрізнити сині пікселі від зелених, другий продовжить працювати з координатами, а третій спробує визначити, у кого з цих двох результати ближче до істини. Якщо ж надати роботу з синіми пікселями відразу кільком нейронам і підсумовувати їх результати, то вийде вже цілий шар, в якому «найкращий нейрон» отримуватимуть додаткові винагороди. Таким чином, досить об'ємна мережа може обробити величезну кількість даних і врахувати при цьому всі свої помилки [27].

Нейронні мережі використовуються в наступних областях: розпізнавання образів і класифікація, прийняття рішень і управління, кластеризація, прогнозування, стиснення даних, аналіз даних.

Нейронні мережі застосовуються на вирішення спеціальних завдань, які вимагають певних обчислень, які можна порівняти з обчисленням людського мозку. Найчастіше нейронні мережі використовуються для:

Класифікація – розподіл даних за параметрами. Наприклад, на вхід нейронної сеті дається список людей і потрібно визначити, кому з них схвалити кредит, а кому ні. З цим нейронна мережа легко впорається, аналізуючи таку інформацію про людей як: стать, кредитна історія, платоспроможність, вік, тощо.

Прогнозування − можливість передбачити наступний крок. Наприклад, результат матчу, зростання або падіння акцій, криптовалют, спираючись на дані з ринку.

Розпізнавання − найпопулярніше застосування нейронних мереж. Використовується світовими компаніями такими як Google, Apple, Microsoft, коли ви шукаєте фото або в камерах ваших смартфонів, коли воно визначає положення вашої особи і виділяє її або в підтвердженні особи та багато іншого.

1.2 Розпізнавання образів і класифікація

Для нейронних мереж як образи можуть бути різні об'єкти: зображення, зразки звуків, символи тексту і т.д.

Під час навчання нейронної мережі надають різні образи із зазначенням того, до якого класу вони належать. У цьому всі ознаки повинні однозначно визначати клас, якого належить приклад. Якщо ознак недостатньо, мережа може співвідносити той самий зразок з кількома класами, що неправильно. Після закінчення навчання мережі їй можна пред'являти невідомі раніше образи та отримувати відповідь про належність до певного класу [28].

Топологія такої мережі характеризується тим, що кількість нейронів у вихідному шарі, як правило, дорівнює кількості визначених класів. При цьому встановлюється відповідність між виходом нейронної мережі та класом, який він представляє. Коли мережі надається образ, одному з її виходів має з'явитися ознака те, що образ належить цього класу. У той самий час інших виходах може бути ознака те, що образ даному класу належить. Якщо так вийшло, що на двох або більше виходах є ознака приналежності до класу, вважається, що мережа не може дати точної відповіді [3].

1.3 Прогнозування

Можливості нейронної мережі до прогнозування результату залежать від її здатності до узагальнення та виділення прихованих залежностей між вхідними та вихідними даними. Після того, як мережа була навчена мережа здатна передбачити майбутнє значення певної послідовності на основі кількох попередніх значень. Прогнозування результату можливе лише у тому випадку, коли попередні зміни справді визначають майбутні. Наприклад, прогнозування котирувань акцій на основі котирувань за минулий тиждень може виявитися успішним, тоді як прогнозування результатів завтрашньої лотереї на основі даних за останні 80 років, швидше за все, не дасть жодних результатів [3].

1.4 Етапи розв'язання задач

Найскладнішим етапом розв'язання задачі – це вибір даних для навчання мережі та їх обробка. Сукупність даних для навчання має відповідати кільком критеріям:

* Репрезентативність — дані повинні показувати справжній стан речей у вибраній галузі;

* Несуперечність - суперечливі дані в навчальній вибірці даних призведуть до того, що мережа не зможе навчитися правильно.

Вихідні дані наводяться до виду, в якому їх можна надіслати на входи мережі. Будь-який запис у файлі даних називається навчальною парою або навчальним вектором. Навчальний вектор містить у собі по одному значенню за кожен вхід мережі і, залежно від типу навчання, з учителем чи ні, за одним значенням кожного виходу мережі. Навчання мережі на неякісному наборі даних, не дає позитивних результатів [28].

Коригування мереж виконується, коли різні входи подаються дані різної розмірності. Наприклад, на перший вхід нейронної мережі подаються дані зі значеннями від 0 до 1, а на другий — від 100 до 1000. За відсутності нормування значення на другому вході завжди будуть значно впливати на вихід мережі, ніж значення на першому вході [28].

Квантування виконується над безперервними величинами, котрим виділяється кінцевий набір дискретних значень. Квантування (англ. quantization) - в обробці сигналів - розбиття діапазону відлікових значень сигналу на кінцеве число рівнів і округлення цих значень до одного з двох найближчих до них рівнів (див рис. 1.2). При цьому значення сигналу може округлятися до найближчого рівня, або до меншого або більшого з найближчих рівнів залежно від способу кодування. Таке квантування називається скалярним. Існує також векторне квантування - розбиття простору можливих значень векторної величини на кінцеве число областей та заміна цих

значень ідентифікатором однієї з цих областей. Найяскравіший приклад квантування - використання цього методу завдання частот звукових сигналів при розпізнаванні промови;

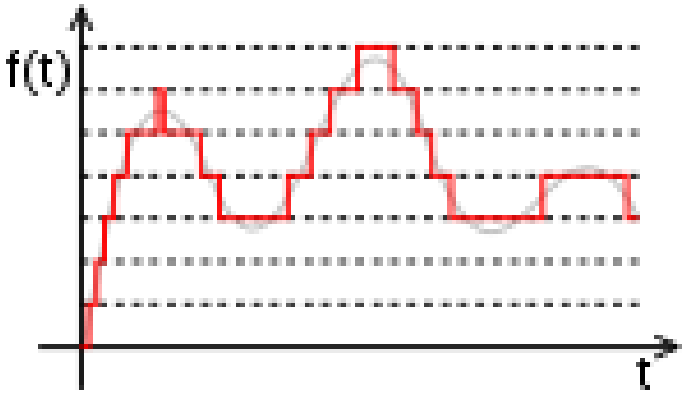


Рисунок 1.2 − Квантований сигнал

Фільтрування виконується для "зашумлених" даних. Крім того, важливу роль відіграє саме уявлення як вхідних, так і вихідних даних. Наприклад, мережа навчається розпізнаванню літер на зображеннях і вихід буде номер літери в алфавіті. У цьому випадку мережа отримає хибне уявлення про те, що літери з номерами 1 та 2 більш схожі, ніж літери з номерами 1 та 3, що загалом неправильно. Для того, щоб уникнути такої ситуації, використовують топологію мережі з великою кількістю виходів, коли кожен вихід має власний сенс. Чим більше виходів у мережі, тим більша відстань між класами і тим складніше їх сплутати [3].

Щоб вибрати тип мережі, потрібно виходити з постановки завдання та наявних вхідних даних для навчання. Для типу навчання з учителем потрібна наявність кожного елемента вибірки «експертної» оцінки. Але бувають ситуації, коли оцінка для великого масиву даних просто неможлива. У таких випадках правильним вибором є мережу, яка навчається без вчителя. При вирішенні інших завдань (таких, як прогнозування тимчасових рядів) експертна оцінка вже міститься у вихідних даних і може бути виділена під час їх обробки. І тут можна використовувати багатошаровий перцептрон [3].

У процесі навчання нейронна мережа певному порядку аналізує навчальну вибірку. Порядок аналізу даних може бути випадковим, послідовним і т. д. Деякі мережі, які навчаються без вчителя (наприклад, мережі Хопфілда), переглядають вибірку лише один раз. Інші (наприклад, мережі Кохонена), а мережі, які навчаються з учителем, можуть переглядати вибірку багато разів, у разі один повний прохід по вибірці називається епохою навчання. При навчанні з учителем набір вхідних даних поділяється на дві частини - навчальну вибірку та тестові дані. Навчальні дані надаються мережами для навчання, а перевірочні використовуються для розрахунку помилки мережі. Отже, якщо на перевірочних даних коефіцієнт помилки зменшується, то мережа дійсно виконує хороше навчання. Якщо помилка на навчальних даних продовжує зменшуватись, а помилка на тестових даних збільшується, потрібно припинити навчання. Під час навчання можуть виявитися інші проблеми, наприклад параліч або влучення мережі в локальний мінімум поверхні помилок. Заздалегідь передбачити появу якоїсь проблеми неможливо [28].

Все вище сказане стосується лише ітераційних алгоритмів пошуку нейромережевих рішень. Для них не можна нічого гарантувати та не можна повністю автоматизувати навчання нейронних мереж. Однак, поряд з ітераційними алгоритмами навчання, існують не ітераційні алгоритми, які мають високу стійкість і дозволяють повністю автоматизувати процес навчання[3].

У галузі комп'ютерного зору більшість завдань вже відомі, і майже для будь-якого з них існує навчальна вибірка, з якої можна розпочати роботу з нейромережею. Але щоб вирішити завдання ефективніше, доводиться збирати нові дані (картинки, відео тощо) і або розмічати їх, або залишати для нейромережі певні підказки, як ці дані обробляти [29].

Важливо зібрати великий масив даних, оскільки, навіть маючи нейромережу з простою архітектурою, ви отримаєте кращий результат, ніж якби завдання вирішувала нейромережа зі складною архітектурою, але

навчаючись на вибірці меншого розміру. Однак зараз на перший план висувається навчання на змішаних наборах даних, наприклад коли мережа навчається на двох наборах даних, один з яких великий і різноманітний, але має помилки в розмітці даних, або взагалі нерозмічений, а інший набір даних правильно оброблений і розмічений, але маленький та недостатньо різноманітний. У такому разі необхідно придумати новий, нестандартний алгоритм, за допомогою якого можна буде навчати нейромережу по обох таких вибірках одночасно [29].

Серед парадигм навчання штучних нейронних мереж розрізняють алгоритми навчання з учителем та без вчителя. Перший передбачає наявність навчальної вибірки даних, що складається з пар вхідних та цільових векторів (тобто відомі умови задачі та саме рішення) та яку пред'являють нейронній мережі. Кожен навчальний приклад подається на вхід мережі, далі у внутрішніх шарах відбувається його обробка, обчислюється вихідний сигнал мережі, і він порівнюється зі значенням цільового вектора очікуваним результатом. На основі цього обчислюється сигнал помилки, який враховується при подальшому перенастроюванні вагових коефіцієнтів нейронів мережі. Процес такого коригування ваг відбувається до того часу, поки значення помилки по всій безлічі вхідних векторів навчального масиву стане мінімальним на відкладеному (валідаційному) наборі даних. Завдяки цьому мережа вчиться. повторювати розмітку даних, зроблену вчителем, і, як наслідок, робити розумні передбачення нових даних [29].

Навчання без вчителя відбувається тоді, коли відомі лише вхідні вектори, на основі яких нейронна мережа вчиться шукати у цих даних якісь закономірності та давати кращий результат – значення вихідного сигналу. Які вектори будуть сформовані на виході, залежить від конкретного типу навчання без вчителя [29].

Робота з нейромережею організується найефективніше, якщо навчання без вчителя звести до навчання з учителем. Дуже багато проривів останніх п'яти років у цьому напрямі пов'язано з використанням саме такої схеми,

сутність якої полягає в наступному. Нейромережа отримує пошкоджені дані - це можуть бути фрази, в яких відсутнє слово або його частини, тоді нейромережа намагається відновити ці прогалини. Або показуємо нейромережі чорно-білі фотографії, і вона має відновити їх колір. Перед тим як поставити нейромережі подібне завдання і запустити навчання, нам не потрібно помічати дані, визначаючи, який підсумковий результат має бути: у нас вже є у величезній кількості подібні картинки, які можна завантажити з інтернету. Але з погляду самої нейромережі таке навчання схоже на навчання з учителем, оскільки на вхід дається чорно-біла картинка або фраза із закритим словом і їй потрібно відновити дані відомим на момент навчання чином [29].

Проміжною ланкою між навчанням з учителем, коли після кожного кроку нейромережі експерт вказує їй, чи помилилася вона чи видала правильний результат, і навчанням без вчителя, при якому нейромережі не приходить із зовнішнього середовища жодних явних підказок, є навчання з підкріпленням. Воно допомагає навчити нейромережа не пророкувати потрібні значення, а поводитися певним чином, наприклад створювати тексти певного жанру. Це дуже схоже на те, як вчиться людина: нейромережа, враховуючи власний досвід та наслідки тих чи інших власних дій, змінює модель прийняття рішень [29].

Найзручніше вдаватися до навчання з учителем, але тільки коли є хороше джерело розмічених даних або необхідні дані зібрати і розмітити. Але зазвичай ситуація така, що обсяг вибірки дуже великий і тому дані складніше підготувати. Тоді проводиться навчання без вчителя, потім нейромережа доучивается на невеликому розміченому наборі даних. У зв'язку з цим можна згадати один із методів у машинному навчанні - це перенесення даних, коли нейромережа, навчену вирішувати конкретне завдання, переключають на нове завдання, маючи обмежену кількість нових даних. Допустимо, нейромережа навчилася розпізнавати породи собак, а далі потрібно, щоб вона зналася і на породах кішок. І оскільки собаки і кішки чимось схожі, наприклад, вовною, то

досвід, який нейромережа отримала, коли навчилася розпізнавати собак, можна використовувати при вирішенні цього нового завдання [29].

Навіть у разі успішного, на перший погляд, навчання мережа не завжди навчається саме тому, чого від неї хотів автор. Відомий випадок, коли мережа навчалася розпізнаванню зображень танків за фотографіями, проте пізніше з'ясувалося, що всі танки були сфотографовані на тому самому тлі. В результаті мережа «навчилася» розпізнавати цей тип ландшафту замість того, щоб «навчитися» розпізнавати танки. Таким чином, мережа «розуміє» не те, що від неї вимагалося, а те, що найпростіше узагальнити.

Тестування якості навчання нейромережі необхідно проводити з прикладів, які брали участь у її навчанні. При цьому кількість тестових прикладів має бути тим більшою, чим вища якість навчання. Якщо помилки нейронної мережі мають ймовірність близьку до однієї мільярдної, то й для підтвердження цієї ймовірності потрібен мільярд тестових прикладів. Виходить, що тестування добре навчених нейронних мереж стає дуже важким завданням[3].

1.5 Класифікація

Класифікація за характером навчання:

- Навчання з учителем - набір правильних рішень на виході з нейронної мережі відомий;
- Навчання без вчителя - нейронна мережа формує вихідний набір правильних рішень лише з урахуванням вхідних даних. Такі мережі називають самоорганізуючими;
- Навчання з підкріпленням – система призначення штрафів та заохочень від середовища[3].

Класифікація за типом вхідної інформації:

- Аналогові нейронні мережі (використовують інформацію у формі дійсних чисел);
- Двійкові нейронні мережі (оперують з інформацією, поданою у двійковому вигляді);
- Образні нейронні мережі (оперують з інформацією, поданою у вигляді образів: знаків, ієрогліфів, символів)[3].

Класифікація за характером налаштування синапсів:

- Мережі з фіксованими зв'язками (вагові коефіцієнти нейронної мережі вибираються відразу, виходячи з умов завдання, при цьому **dW**/dt = 0, де W - вагові коефіцієнти мережі);
- Мережі з динамічними зв'язками (для них у процесі навчання відбувається налаштування синоптичних зв'язків, тобто **dW**/dt ≠ 0, де W - вагові коефіцієнти мережі).

Класифікація за часом передачі сигналу:

У ряді нейронних мереж активуюча функція може залежати не тільки від вагових коефіцієнтів зв'язків ω_{ij}, а й від часу передачі імпульсу (сигналу) каналами зв'язку τ_{ij}. Тому в загальному вигляді активуюча (передавальна) функція зв'язку c_{ij} від елемента u_i до елемента u_j має вигляд: $c_{ij}^* = f\left|\omega_{ij}(t), u_i^*(t-\tau_{ij})\right|$. Тоді синхронною мережею називають таку мережу, у якої час передачі τ_{ij} кожного зв'язку дорівнює або нулю, або фіксованою постійною τ. Асинхронною називають таку мережу, у якої час передачі τ_{ij} для кожного зв'язку між елементами u_i і u_j своє, але теж постійне[3].

Класифікація характеру зв'язків:

- **Нейронні мережі прямого розповсюдження**: У нейронних мережах прямого поширення (англ. feedforward neural network) всі зв'язки

спрямовані суворо від вхідних нейронів до вихідних. Прикладами таких мереж є перцептрон Розенблатта, багатошаровий перцептрон мережі Ворда.

- **Рекурентні нейронні мережі:** Сигнал з вихідних нейронів або нейронів прихованого шару частково передається на входи нейронів вхідного шару (зворотний зв'язок). Рекурентна мережа Хопфілда сортує вхідні дані, повертаючись до сталого стану і, таким чином, дозволяє вирішувати завдання компресії даних та побудови асоціативної пам'яті. Окремим випадком рекурентних мереж є двонаправлені мережі. У таких мережах між шарами існують зв'язки як у напрямку від вхідного шару до вихідного, і у зворотному. Класичним прикладом є Нейронна мережа Коско.
- **Радіально-базисні функції:** Розроблено нейронні мережі, які використовують як активаційні функції радіально-базисні (також називаються RBF-мережами).

$$f(x) = \varphi(\frac{x^2}{\sigma^2})$$

Наприклад,

$$f(x) = e^{-\frac{x^2}{\sigma^2}}$$

де x - вектор вхідних сигналів нейрона, σ - ширина вікна функції, φ(y) - спадна функція (найчастіше, рівна нулю поза деяким відрізком).

Радіально-базова мережа характеризується трьома особливостями: Єдиний прихований шар; Лише нейрони прихованого шару мають нелінійну активаційну функцію; Синаптичні ваги зв'язків вхідного та прихованого шарів дорівнюють одиниці.

- **Самоорганізовані карти:** Такі мережі є змагальною нейронною мережею з навчанням без вчителя, що виконує завдання візуалізації та кластеризації. Є методом проектування багатовимірного простору в простір з нижчою розмірністю (найчастіше двовимірне), застосовується також для вирішення завдань моделювання, прогнозування та ін. Є однією з версій нейронних мереж Кохонена. Картки Кохонена, що самоорганізуються, служать, насамперед, для візуалізації та початкового («розвідувального») аналізу даних. Сигнал у мережу Кохонена надходить відразу на всі нейрони, ваги відповідних синапсів інтерпретуються як координати положення вузла, і вихідний сигнал формується за принципом «переможець забирає все» — тобто ненульовий вихідний сигнал має нейрон, найближчий (у сенсі ваг синапсів) до поданого на вхід об'єкту. У процесі навчання ваги синапсів налаштовуються таким чином, щоб вузли решітки «розташовувалися» у місцях локальних згущень даних, тобто описували кластерну структуру хмари даних, з іншого боку, зв'язки між нейронами відповідають відносинам сусідства між відповідними кластерами у просторі ознак [3].

1.6 Функції активації

Що в результаті робить створений штучний нейрон? Простими словами, штучний нейрон підраховує зважену суму(вага) на своїх входах, додає зміщення (bias) і приймає рішення, чи слід це значення використовувати далі або ігнорувати його. Функція активації визначає вихідне значення нейрона в залежності від результату виваженої суми входів та порогового значення.

Ступінчаста функція активації:

Для початку слід зрозуміти, що вважати межею активації для активаційної функції. Якщо значення Y більше деякого порогового значення, вважаємо нейрон активованим, інакше нейрон неактивний.

- Функція A = активована, якщо Y > кордон, інакше ні.
- Інший спосіб: A = 1, якщо Y > кордон, інакше A = 0.

Ця функція і називатиметься ступінчастою.

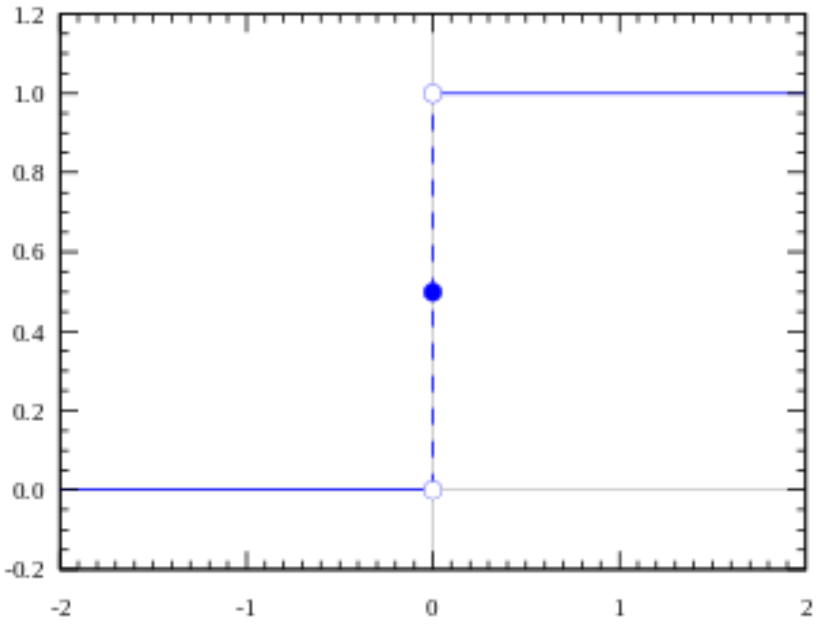


Рисунок 1.3 − Ступінчаста функція активації

Функція приймає значення 1 (активована), коли Y > 0 (кордон), і значення 0 (не активована) інакше. Уявімо, що створюється бінарний класифікатор — модель, яка має в результаті видати так чи ні (активований чи ні). Ступінчаста функція зробить це за вас - вона точно виводить 1 або 0[4].

Лінійна функція активації:

Вибір лінійної функції активації дозволяє отримувати спектр значень, а не тільки бінарну відповідь. У цьому випадку можна поєднати кілька нейронів разом і, якщо більше одного нейрона активовано, рішення приймається на основі застосування операції max (або softmax). Похідна від A = cx по x дорівнює c. Це означає, що градієнт ніяк не пов'язаний з X. Градієнт є постійним вектором, а спуск здійснюється за постійним градієнтом. Якщо у

результаті вийшло хибне передбачення, то зміни, зроблені зворотним поширенням помилки, теж постійні і залежить від зміни на вході delta(x) [4].

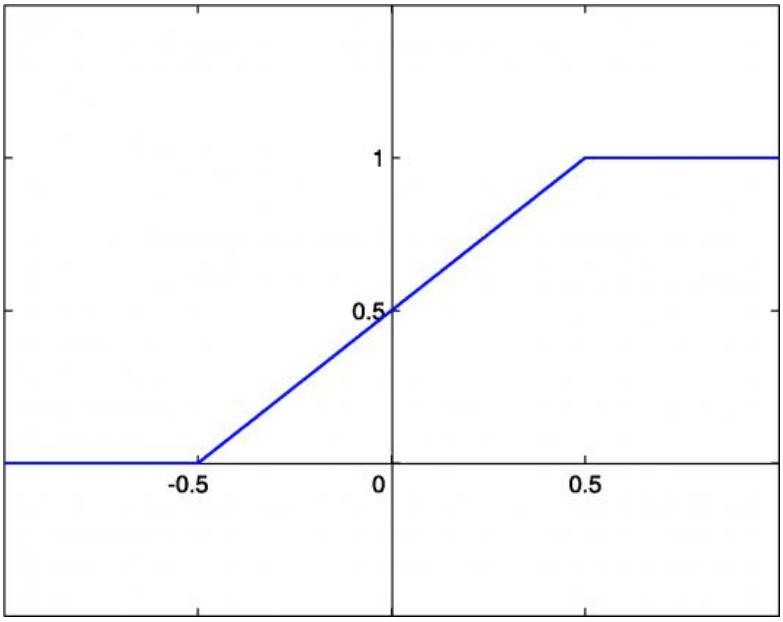


Рисунок 1.4 − Лінійна функція активації

Сігмоїда:

Сігмоїдна функція виглядає гладкою і трохи подібна до ступінчастої функції (див. рис 1.5) її формула

$$f(x) = \frac{1}{1 + e^{-x}}$$

Розглянемо її переваги. По-перше, сігмоїда — нелінійна, а комбінація таких функцій робить також нелінійну функцію. Тепер з'являється можливість працювати із шарами. По-друге, вона не бінарна і це робить активацію аналоговою, на відміну від ступінчастої функції. У діапазоні значень X від -2 до 2 значення Y змінюється дуже швидко. Це означає, що будь-яке мале зміна значення X у цій галузі тягне за собою істотне зміна значення Y. Така поведінка функції свідчить про те, що Y має тенденцію притискатися одного з країв кривої. Сігмоїда дійсно виглядає підходящою функцією для

класифікаційних завдань. Вона прагне привести значення однієї зі сторін кривої (наприклад, до верхнього при x=2 і нижньому при x=-2). Така поведінка дозволяє знаходити чіткі межі під час передбачення [4].

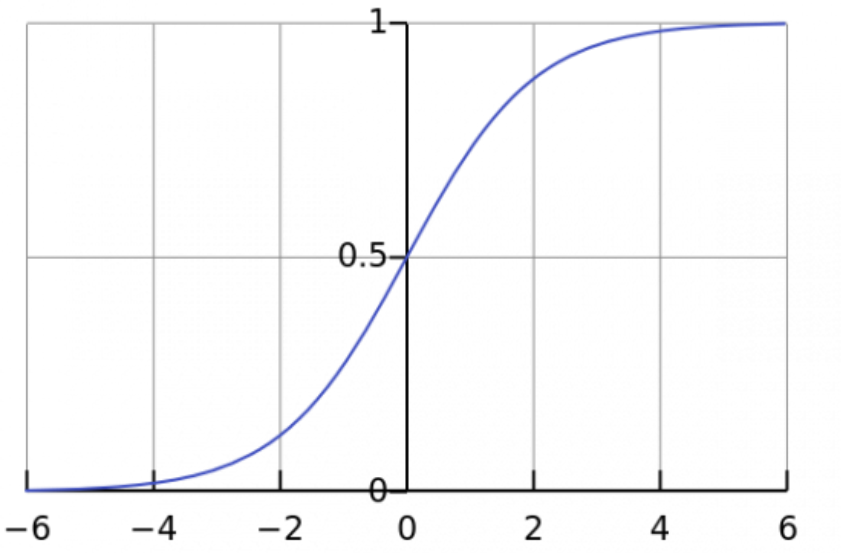


Рисунок 1.5− Сігмоїдна функція

Гіперболічний тангенс:

Ще один вид активної функції, що часто використовується, — гіперболічний тангенс(див рис. 1.6). Гіперболічний тангенс трохи схожий на сигмоїду. Він виглядає як скоригована сигмоїдна функція. Гіперболічний тангенс має самі характеристики, що й у сигмоїди. Вона нелінійна та добре підходить для комбінації шарів, а діапазон значень функції -(-1, 1). З цього можна дійти невтішного висновку, що активаційна функція не перевантажиться від великих значень. При цьому градієнт тангенціальної функції більше, ніж у сигмоїди (похідна крутіше). Рішення про те, чи вибрати сигмоїду чи тангенс, залежить від ваших вимог до амплітуди градієнта. Також як і сигмоїді, гіперболічний тангенс властива проблема зникнення градієнта [4].

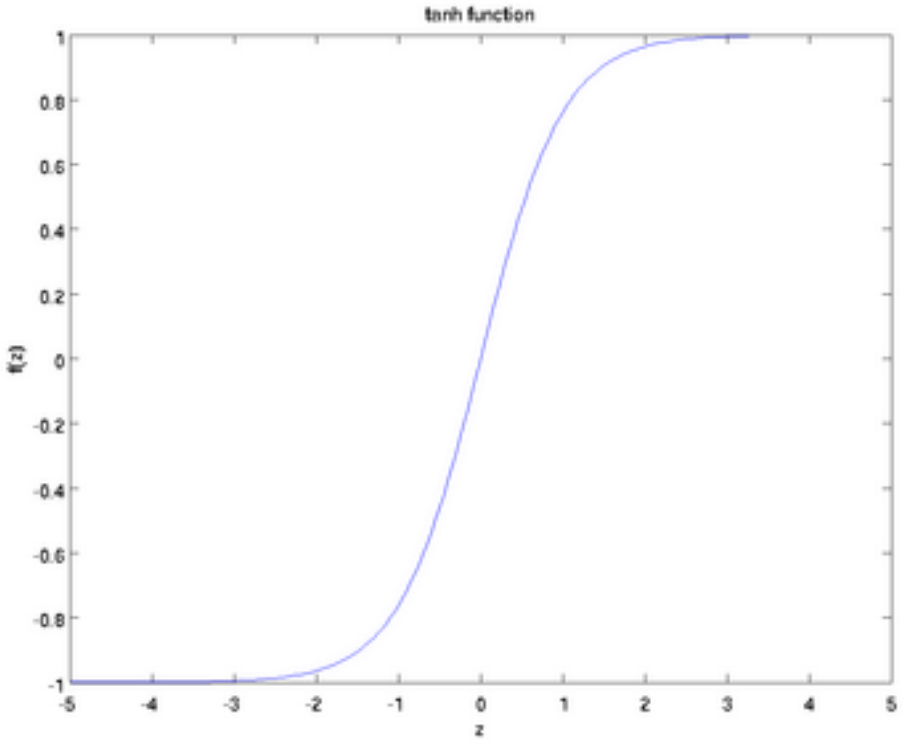


Рисунок 1.6− Гіперболічний тангенс

1.7 Нейрон

Нейрон - це обчислювальна одиниця, яка отримує інформацію, здійснює над нею прості обчислення і передає її далі. Вони поділяються на три основні типи: вхідний, прихований та вихідний. Також є нейрон зміщення та контекстний нейрон. У разі, коли нейромережа складається з великої кількості нейронів, вводять термін шару. Відповідно, є вхідний шар, який отримує інформацію, n прихованих шарів (зазвичай їх не більше 3), які її обробляють та вихідний шар, який виводить результат. У кожного з нейронів є 2 основні параметри: вхідні дані (input data) та вихідні дані (output data). Що стосується вхідного нейрона: input=output. В інших, в поле input потрапляє сумарна інформація всіх нейронів з попереднього шару, після чого вона нормалізується за допомогою функції активації (поки що просто представимо її f(x)) і потрапляє в поле output [19].

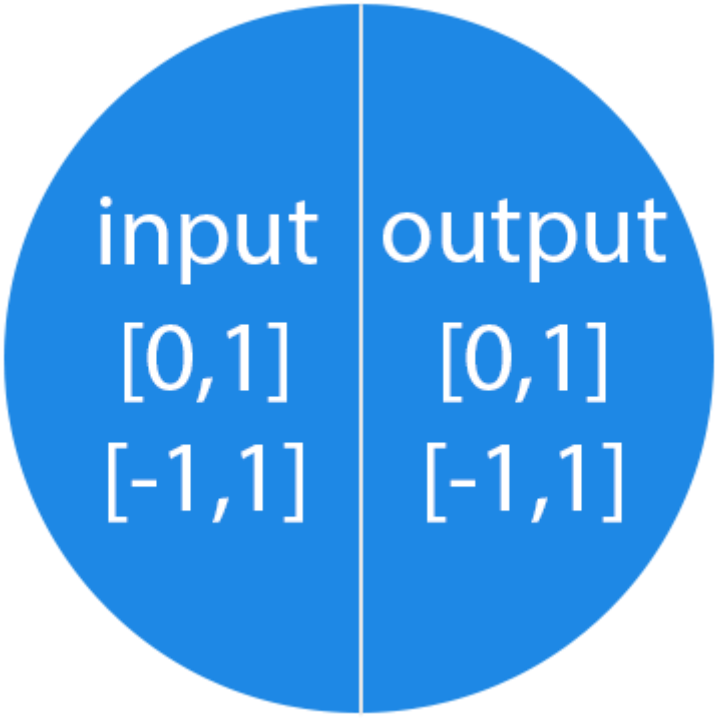


Рисунок 1.7 − Вигляд нейрона

1.8 Синапс

Синапс це зв'язок між двома нейронами. У синапсів є один параметр - вага. Завдяки йому вхідна інформація змінюється, коли передається від одного нейрона до іншого. Допустимо, є 3 нейрони, які передають інформацію наступному. Тоді є 3 ваги, які відповідають кожному з цих нейронів. У того нейрона, у якого вага буде більшою, та інформація і буде домінуючою в наступному нейроні (приклад - змішання кольорів). Насправді, сукупність терезів нейронної мережі або матриця терезів - це своєрідний мозок всієї системи. Саме завдяки цим вагам, вхідна інформація обробляється та перетворюється на результат.

Важливо пам'ятати, що під час ініціалізації нейронної мережі ваги розставляються у випадковому порядку [19].

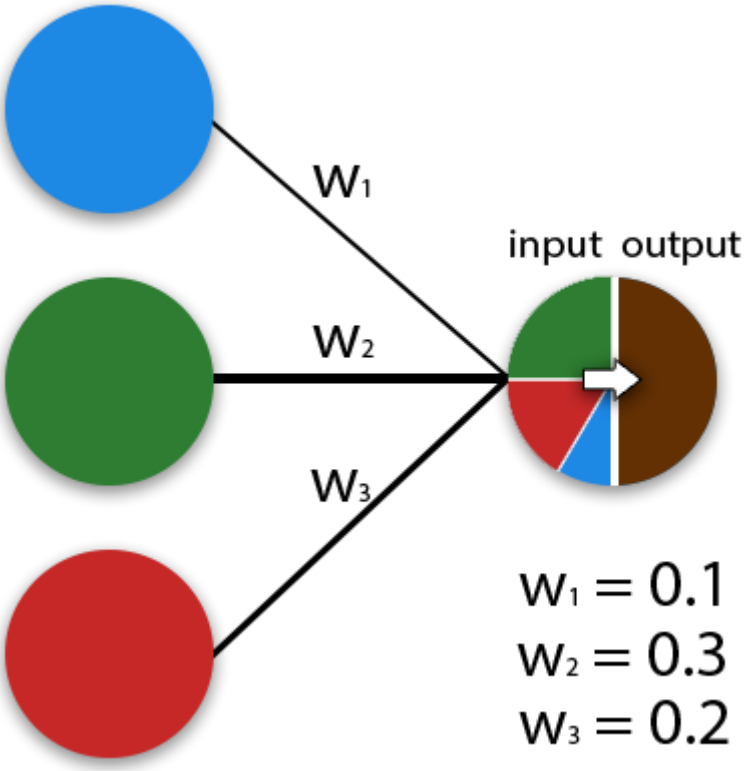


Рисунок 1.8 − Вигляд синапсу

1.9 Глибинна нейронна мережа

Глибинна нейронна мережа (ГНМ, англ. DNN - Deep neural network) - це штучна нейронна мережа (ШНМ) з декількома шарами між вхідним та вихідним шарами [20] [21]. ГНМ знаходить коректний метод математичних перетворень, щоб перетворити вхідні дані на вихідні, незалежно від лінійної чи нелінійної кореляції. Мережа просувається шарами, розраховуючи ймовірність кожного виходу. Наприклад, ГНМ, яка навчена розпізнавати породи собак, пройде за заданим зображенням і обчислить ймовірність того, що собака на зображенні відноситься до певної породи. Користувач може переглянути результати та вибрати ймовірності, які має відображати мережа (вище певного порога, наприклад), та повернути до мережі запропоновану

мітку. Кожне математичне перетворення вважається шаром, а складні ГНМ мають багато шарів, звідси і назва "глибинні" або "глибокі" мережі.

Глибині нейронні мережі можуть моделювати складні нелінійні відносини. Архітектури ДПС генерують композиційні моделі, у яких об'єкт виражається як багаторівневої композиції примітивів [22]. Додаткові рівні дозволяють складати елементи з нижчих рівнів, потенційно моделюючи складні дані з меншою кількістю одиниць, ніж дрібна мережа з аналогічними показниками [20].

Глибока архітектура включає безліч варіантів декількох основних підходів. Кожна архітектура знайшла успіх у певних сферах. Не завжди можна порівняти продуктивність декількох архітектур, якщо вони не були оцінені на тих самих наборах даних.

Глибині нейронні мережі, як правило, являють собою мережі з прямим зв'язком, в яких дані передаються від вхідного рівня до вихідного рівня без зворотного зв'язку. Спочатку глибина нейронна мережа створює карту віртуальних нейронів і призначає випадкові числові значення або «ваги» з'єднань між ними. Вага та вхідні дані множаться і повертають вихідний сигнал від 0 до 1. Якщо мережа не точно розпізнала конкретний шаблон, алгоритм коригуватиме вагові коефіцієнти [23]. Таким чином, алгоритм може зробити певні параметри більш значущими, доки він не визначить правильні математичні маніпуляції для повної обробки даних [24].

Існують завдання, які нейромережі вирішують так само добре як і людина, а деякі навіть краще: наприклад, нейромережі здатні точніше передбачають структуру білка, розпізнають сфотографовані особи, обігрують людей у настільні ігри такі як шашки, шахи та популярну китайську гру го. Вже зараз їм доручено переглядати камери відеоспостереження та реагувати на ситуації, що спричиняють небезпеку для людини.

Але зараз нейронні мережі гірше, ніж люди, здатні оцінити ступінь точності своїх пророцтв, і якщо людина може визнати свою невпевненість у чомусь, то нейромережа на таке не здатна. Наприклад, якщо доручити

нейронної мережі, навченої вирішувати завдання розпізнавання порід кішок, і на вхід дати їй вівцю, вона віднесе її до якої-небудь породи кішок. Звідси випливає така проблема: якщо в розмітці даних припущені помилки, то нейромережа їх просто не замислюючись вивчить.

Але все-таки найбільша ефективність у вирішенні більшості завдань досягається завдяки співпраці людини та нейромережі, які доповнюють одна одну. Наприклад, найкращими гравцями в го або шахи є «кентаври» - команди, що складаються з нейромережі та людини; хоча останнім часом, можливо, деякі нейромережі отримують вже менше переваг партнерства з людиною. Однак гра в шахи або го — це штучно створена ситуація, а в реальному світі з тим, що пов'язано з розумінням потреб інших людей і має на увазі common sense («здоровий глузд»), людина справляється набагато краще за нейромережі [29].

1.10 Сучасні методи підвищення ефективності роботи нейронних мереж

Швидкість навчання - визначення потрібної швидкості навчання важливий, тому що вирішує, чи ваша мережа наближається до глобальних мінімумів чи ні. За високої швидкості навчання мережа рідко коли підійде до глобальних мінімумів, оскільки великі шанси подолати її. Тому мережа в основному знаходиться навколо глобальних мінімумів, але ніколи не зводиться до них. Вибір нижчої швидкості навчання може допомогти нейронної мережі наблизитися до глобальних мінімумів, але це займає багато часу. Тому мережа навчатиметься протягом тривалого часу. При невеликій швидкості навчання мережі є ризик застрягання в локальному мінімумі. Тобто мережа буде сходитися до локальних мінімумів і не зможе вийти з неї через низьку швидкість навчання.

Мережева архітектура. Не існує однієї правильної архітектури, що забезпечує високу точність у всіх випадках застосування мереж. Потрібно

пробувати різні архітектури, отримувати висновки з результату, аналізувати та намагатися знову. Є варіант використати вже перевірену архітектуру замість створення власної. Наприклад початкова мережа від Google і т. д. Усі вони з відкритим вихідним кодом і показують високі результати. Її можна скопіювати та налаштувати їх для своїх цілей.

Оптимізатори та функція втрат – існує безліч варіантів, доступних для вибору. За потреби можна визначити свою власну функцію втрат. Але зазвичай використовуються оптимізатори RMSprop, Stochastic Gradient Descent та Adam. Ці оптимізатори працюють для більшості випадків використання.

Розмір партії та кількість епох. Не існує стандартного значення для розміру партії та епох, що працює для всіх випадків використання. Для кожної нейронної мережі вони можуть бути свої, потрібно пробувати і методом проб та помилок підбирати оптимальні. У випадку значення розміру пакета встановлюються як 4, 8, 16 тощо. Кількість епох залежить від переваг розробника та обчислювальної потужності.

Функція активації – функції активації відображають нелінійні функціональні входи на виходи. Вибір правильної функції активації допомагає вашій моделі вчитися краще. В даний час випрямлена лінійна одиниця (ReLU) є функцією активації, що найбільш широко використовується, так як ця функція вирішує проблему зникаючих градієнтів. Раніше Sigmoid і Tanh були найчастіше використовуваною функцією активації. Але у них є недолік – проблема зникаючих градієнтів. Під час зворотного поширення градієнти зменшуються у ціні, коли досягають початкових шарів. Це зупинило нейронну мережу від масштабування до великих розмірів із великою кількістю шарів. Функція активації ReLU змогла вирішити цю проблему і дозволила нейронним мережам мати великі розміри.

1.11 Висновки

У цьому розділі було проведено аналітичний огляд існуючих нейронних мереж, для яких цілей нейронні мережі використовуються, було показано які методи навчання існують. Був проведений аналіз різноманітних функцій активацій. Описано з яких елементів складаються нейронні мережі. Було розказано у яких областях використовують нейронні мережі. Проаналізовано що таке нейрон та синапс. Розповіли про способи, які існують зараз, для підвищення працездатності нейронних мереж.

2 ПОБУДОВА МОДЕЛІ ТА МЕТОДУ НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ

Для побудови прикладу нейронної мережі необхідно визначитися з деякими моментами:

1). Вибір архітектури для моделі нейронної мережі;

2). Вибір початкових ваг;

3). Вибір функції активації;

4). Вибір виду навчання;

5). Вибір мови програмування для створення моделі нейронної мережі.

2.1 Визначення архітектури та елементів для моделі нейронної мережі

Існує безліч різних архітектур на основі яких створюються нейромережі, такі як Перцептрон, Згорточні нейронні мережі, Карта Кохонена, що самоорганізується, Мережі адаптивного резонансу та інші. У разі використовується архітектура Перцептрон оскільки вона у реалізації і цілком задовольняє даним умовам. Принцип роботи цієї архітектури насправді дуже простий. На вхідний шар надходить інформація, після чого вона обробляється у прихованому шарі та передається на вихідний шар. Загалом топологія виглядає так (див. рис. 2.1).

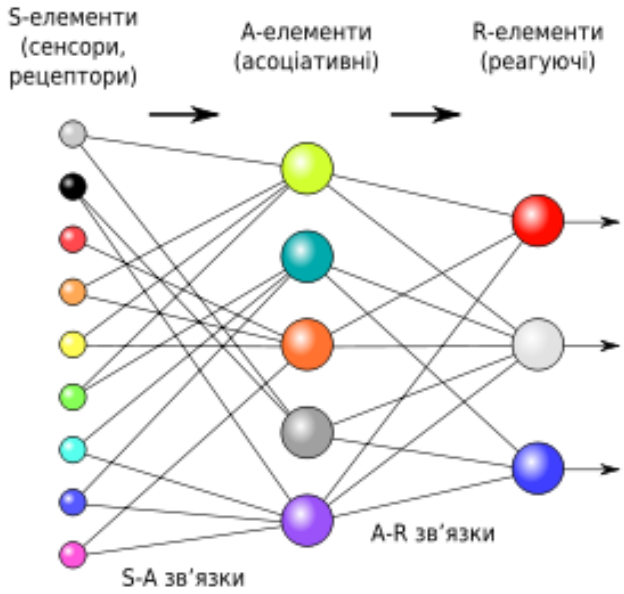


Рисунок 2.1− Логічна схема перцептрону з трьома виходами

В основі перцептрон лежить математична модель сприйняття інформації мозком. Різні дослідники по-різному визначають. У найзагальнішому своєму вигляді (як його описував Розенблатт) він представляє систему з елементів трьох різних типів: сенсорів, асоціативних елементів та реагуючих елементів.

Першими в роботу включаються S-елементи. Вони можуть бути або в стані спокою (сигнал дорівнює 0), або в стані збудження (сигнал дорівнює 1); Далі сигнали від S-елементів передаються A-елементів так званим S-A зв'язків. Ці зв'язки можуть мати ваги, рівні лише -1, 0 або 1. Потім сигнали від сенсорних елементів, що пройшли S-A зв'язків, потрапляють в A-елементи, які ще називають асоціативними елементами.

Далі сигнали, які провели збуджені A-елементи, направляються до суматора (R-елемент), дія якого нам вже відома. Однак, щоб дістатися до R-елемента, вони проходять A-R зв'язків, у яких теж є ваги (які вже можуть приймати будь-які значення, на відміну від S-A зв'язків);

R-елемент складає один з одним зважені сигнали від A-елементів, а потім, якщо перевищений певний поріг, генерує вихідний сигнал, що дорівнює 1;

Якщо поріг не перевищений, то вихід перцептрону дорівнює -1. Для елементів перцептрон використовують наступні назви: [6]

S-елементи називають сенсорами;

A-елементи називають асоціативними;

R-елементи називають реагуючими.

З вибором ваг все просто, на даному етапі можна вибирати будь-які значення в діапазоні від -1 до 1, головне щоб ваги не були однаковими. В цьому випадку мережа не буде здатна до самонавчання. Тому скористаємося генератором випадкових чисел отримання початкових ваг.

Функцій активацій також існує кілька видів, такі як сигмоїда, лінійна, ступінчаста, ReLu, гіперболічний тангенс. Гіперболічний тангенс дуже схожий на сигмоїду. І насправді це скоригована сигмоїдна функція. Сама функція виглядає так:

$$f(x) = \frac{2}{1+e^{-2x}} - 1.$$

Також існує кілька видів навчання нейронних мереж, такі як навчання з учителем, навчання без вчителя, змішане навчання. У разі використовується навчання без вчителя. Модель нейронної мережі наддасть файл, за яким і відбуватиметься навчання. Щоб результат навчання був точнішим, потрібно надати велику кількість прикладів. Створимо програму, яка згенерує такий файл.

```
for (int i = 2000; i >= 0; i--)
{
        int n1 = (int)(2.0 * rand() / double(RAND_MAX));
        int n2 = (int)(2.0 * rand() / double(RAND_MAX));
        int t = !(n1 & n2);
        std::cout << "in: " << n1 << ".0 " << n2 << ".0" << std::endl;
```

```
            std::cout << "out: " << t << ".0 " << std::endl;
        }
```

У результаті вихідний файл міститиме такі дані (див. рис. 2.2).

```
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 1.0 1.0
out: 0.0
```

Рисунок 2.2 − Інформація для навчання

Для створення моделі нейронної мережі можна використовувати будь-яку сучасну мову програмування. У разі використовується мову програмування C++.

2.2 Побудова нейронної мережі

Для початку створимо клас Network, це буде наша мережа. У конструктор передаватимемо топологію або її ще називають структуру нейронної мережі. Тобто кількість нейронів і скільки нейронів у кожному шарі.

Network(const std::vector<unsigned>& myTopol);

Далі знадобляться три методи - це feedForw який буде наповнювати мережу даними, backPropagation сам алгоритм тренування мережі, аргументом

там буде те, що планується отримати на виході і метод getRes для того щоб отримувати дані яка мережа апроксимує, тобто по суті вихідний результат.

void feedForw(const std::vector<double>& inpValues);

void backPropagetion(const std::vector<double>& targValues);

void getRes(std::vector<double>& resValues) const;

Далі створюється топологія мережі - це просто масив із чисел, де кожен елемент відповідатиме за конкретну частину архітектури (вхідні, вихідні нейрони, приховані шари). Необхідно створити кілька нейронів і поєднати їх. Тому створюється вектор не з нейронів, а вектор із шарів, в якому кожен шар містить один або кілька нейронів. Це буде як багатовимірний масив.

typedef std::vector<MyNeurn> Layer;

std::vector<Layer> layers; //layers[Номер Слоя][Номер Нейрона]

Напишемо реалізацію конструктора Network. У ній буде два цикли. Перший цикл буде створювати шари і додавати у вектор layers, а вкладений цикл додаватиме відповідні нейрони у відповідний layers.

for (unsigned layerNum = 0; layerNum < numLayers; ++layerNum) {

layers.push_back(Layer());

unsigned numOutputs = layerNum == myTopol.size() - 1 ? 0 : myTopol[layerNum + 1];

for (unsigned MyNeurnNum = 0; MyNeurnNum <= myTopol[layerNum]; ++MyNeurnNum) {

layers.back().push_back(MyNeurn(numOutputs, MyNeurnNum));}

layers.back().back().setOutputVal(1.0); }

Повний код класу Network дивіться у додаткуА.

Переходимо до створення класу для нейрона. Називається MyNeurn. У ньому буде два приватні поля, це те, що нейрон видає на вихід і вектор зі з'єднань з іншими нейронами.

```
double outputVal;
std::vector<Connection> outputWeights;
```

Структура Connection буде містити вагу нейрона, який буде змінюватися в ході навчання і поле, яке міститиме різницю зміни у вагах (дельта).

```
struct Connection {
double weight;
double deltaWeight; };
```

У конструкторі класу MyNeurn створюється з'єднання між усіма нейронами і призначаємо кожному останньому створеному з'єднанню випадкову вагу, яка буде дорівнювати діапазону між 0 та 1.

```
MyNeurn::MyNeurn(unsigned numOutputs, unsigned myIndex) {
        for (unsigned c = 0; c < numOutputs; ++c) {
          outputWeights.push_back(Connection());
          outputWeights.back().weight = randomWeight();}
        m_myIndex = myIndex;}
static double randomWeight() { return rand() / double(RAND_MAX); }
```

Так само додаються до цього класу методи сеттер та геттер для вихідних нейронів.

```
void setOutputVal(double val) { outputVal = val; }
double getOutputVal() const { return outputVal; }
```

Створимо ще один метод, що отримувавпопереду, але вже для нейрона. Він потрібен для того, щоб підсумувати всі вхідні дані з попереднього шару і перемножувати їх з їхньою ж вагою.

```
void MyNeurn::feedForw(const Layer& prevLayer) {
    double sum = 0.0;
    for (unsigned n = 0; n < prevLayer.size(); ++n) {
      sum += prevLayer[n].getOutputVal() *
      prevLayer[n].outputWeights[m_myIndex].weight; }
    outputVal = MyNeurn::activationFunction(sum);}
```

Далі необхідно реалізувати саму функцію активації, щоб сформувати вихідні дані нейрону. Цією функцією, як говорилося раніше, можливо будь-що, але для прикладу береться гіпербалічний тангенс. Так само нам під час навчання знадобиться похідна цією функцією метод activationFunctionDerivative.

```
double MyNeurn::activationFunction(double x) {
    //вихідний діапазон [-1.0..1.0]
    return tanh(x); }
```

Повний код класу MyNeurn дивіться у додаткуБ.

Далі необхідно реалізувати алгоритм навчання нейронної мережі (backPropagetion). Для початку потрібен обчислювач помилки нейронної мережі. Обчислюється це за формулою

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

що й реалізується у коді. RMSE - Root Mean Square Error (Середньоквадратична помилка).

```
void Network::backPropagetion(const std::vector<double>& targValues) {
    Layer& outputLayer = layers.back();
    error = 0.0;
    for (unsigned n = 0; n < outputLayer.size() - 1; ++n) {
     double delta = targValues[n] - outputLayer[n].getOutputVal();
      error += delta * delta;}
    error /= outputLayer.size() - 1;
    error = sqrt(error);
```

Далі обчислюються градієнти, це протилежність похідним, тобто векторна величина. Обчислюються градієнти як вихідного шару так прихованого.

```
for (unsigned n = 0; n < outputLayer.size() - 1; ++n) {
    outputLayer[n].calcOutputGradients(targValues[n]); }
for (unsigned layerNum = layers.size() - 2; layerNum > 0; --layerNum) {
    Layer& hiddenLayer = layers[layerNum];
    Layer& nextLayer = layers[layerNum + 1];
    for (unsigned n = 0; n < hiddenLayer.size(); ++n) {
      hiddenLayer[n].calcHiddenGradients(nextLayer);}}
```

І наприкінці оновлюємо значення ваги.

```
for (unsigned layerNum = layers.size() - 1; layerNum > 0; --layerNum) {
    Layer& layer = layers[layerNum];
    Layer& prevLayer = layers[layerNum - 1];
    for (unsigned n = 0; n < layer.size() - 1; ++n){
      layer[n].updateInputWeights(prevLayer); }}
```

Повний код класів дивитись у додатках А та Б.

Далі потрібно створити клас, за допомогою якого нейронна мережа отримуватиме дані. До самої нейронної мережі цей клас не має жодного відношення, він потрібен для зручності використання створеної нейронної мережі. Для повного коду дивитись додаток В.

2.3 Зміна ваг для покращення роботи мережі

У прикладі створення нейронної мережі для початкових ваг використовують випадкові числа, які генеруються системою. У багатьох джерелах говориться, що початкові ваги можуть бути будь-якими в діапазоні від 0 до 1, головне щоб ваги були різними, інакше нейромережа не буде здатна до навчання. Замінимо випадкові числа на значення, що інкрементує. Перший нейрон буде з вагою 0.01 і кожен наступний з вагою на 0.01 більше попереднього. У класі MyNeurn заміняємо реалізацію методу randomweight. Тепер цей метод має такий вигляд:

```
double MyNeurn::randomWeight() {
    w = + 0.01;
    return w; }
```

2.4 Зміна функції активації для покращення роботи мережі

Як вже відомо існують 4 види основних функцій активацій – сигмоїда, лінійна, ступінчаста, ReLu та гіперболічний тангенс. У прикладі простої

нейронної мережі використовується гіперболічний тангенс. Мета домогтися підвищення працездатності нейронної мережі, тобто збільшити швидкість навчання і зменшити ймовірність неправильного результату. Для цього змінимо роботу функції активації та її похідну. Замість гіперболічного тангенсу використовуватимемо звичайний arctg, формула якого

$$f(x) = tg^{-1}(x)$$

а його похідна відповідно дорівнює

$$f`(x) = \frac{1}{x^2 + 1}$$

Реалізація даних рівнянь у коді виглядатиме так.

```
double calcArctng(double x, int accuracy = 1000) {
    double a = x;
    double sum = a;
    double b = a;
    double c = 1 / accuracy;
    for(int i = 1; a > c; i++){
        b = b * ( - x * x);
        a = a * (b / (2 * i + 1));
        sum = sum + a;}
    return sum;}
```

У метод calcArctng передається два аргументи, де x - сам шуканий аргумент, а accuracy - це точність з якою буде обчислюватися. Так як у цьому випадку нам потрібна точність як можна вище, за замовчуванням значення дорівнює 1000.

Похідна для arctg в коді має наступний вигляд, зміни в методі activationfunction derivative.

```
double MyNeurn::activationFunctionDerivative(double x)
{
        return 1 / (pow(x, 2) + 1);
}
```

І другий спосіб зміни функції активації для підвищення ефективності нейронної мережі – це використання Softsign функції та її похідної. Функція Softsign має наступний вигляд

$$f(x) = \frac{x}{1 + |x|}$$

а її похідна виглядає так

$$f`(x) = \frac{1}{(1 + |x|)^2}$$

Реалізація даних рівнянь у коді виглядатиме так.

```
double MyNeurn::activationFunction(double x) {
return x / (1 + abs(x)); }
```

Функція abs обчислює абсолютне значення та повертає модуль значення. Функція pow зводить значення першого аргументу на ступінь значення другого аргументу. Для їх використання необхідно підключити заголовний файл #include <cmath> для C++ або #include <math.h> для C.

Похідна для Softsign в коді має наступний вигляд, зміни в методі activationfunction derivative.

```
double MyNeurn::activationFunctionDerivative(double x) {
return 1 / pow(1 + abs(x),2); }
```

Результати та тестування даних зміни ваг та функцій активація представлені у 3 розділі про тестування нейронної мережі.

2.5 Висновки

У цьому розділі було обрано основні елементи для підвищення ефективності нейронної мережі – нові ваги та функції активації. Запропоновано метод виставляти початкові ваги не випадковим образом, а завдяки інкриментації. А у функціях активації використовувати не гіперболічний тангенс, а тригонометричні функції Softsign та arctg. Описано принцип створення моделі простої нейронної мережі та запропоновано методи для підвищення ефективності роботи нейронної мережі. Вибрали мову програмування C++, через швидкість виконання коду, якою створювалася модель нейронної мережі. Була побудована архітектура та топологія моделі нейронної мережі, реалізовані функції активації і навчили мережу до самонавчання.

3 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ

Тестування програмного забезпечення (Software Testing) - перевірка відповідності між реальним і очікуваним поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином. У більш широкому сенсі, тестування - це одна з технік контролю якості, що включає в себе активності з планування робіт (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) і аналізу отриманих результатів (Test Analysis).

Для того, щоб упевнитися, що розробляється мережа повністю працездатна необхідно її протестувати. Тестується вся програма яка написана на мові C ++. В результаті тестування можна виявити неробочі елементи в системі і її недоліки. Тестування проходить поетапно. Спочатку проводиться перевірка прикладу простої нейронної мережі, вимірюється швидкість роботи програми і виглядає можливість помилки отриманого результату. Після чого проводиться тестування зі зміненими вагами та заміряються результати цієї нейромережі. І далі тестуються перероблені функції активації з гіперболічного тангенсу на арктангенс та Softsign.

3.1 Тестування звичайної моделі нейронної мережі

Для того щоб протестувати модель нейронної мережі для початку потрібно створити клас який надаватиме нейронній мережі вхідні дані на яких вона буде навчатися. Клас називається TrainingSet і повний код цього класу надано в додатку 3. Також переробимо main (вхідна точка програми) в якому програма буде відкривати вихідний текстовий файл, який містить вхідні дані для навчання (див. рис 2.2).

```
int main() {
    TrainingSet trainingData("testData.txt");
```

```
std::vector<unsigned> myTopol;
trainingData.getTopology(myTopol);
Network net(myTopol);
std::vector<double> inpValues, targValues, resValues;
int trainingPass = 0;
while (!trainingData.isEOF()) {
    ++trainingPass;
    std::cout << std::endl << "Pass: " << trainingPass << std::endl;
    if (trainingData.getNextInputs(inpValues) != myTopol[0])
        break;
    showVectorVals("Input:", inpValues);
    net.feedForw(inpValues);
    trainingData.getTargetOutputs(targValues);
    showVectorVals("Targets:", targValues);
    assert(targValues.size() == myTopol.back());
    net.getRes(resValues);
    showVectorVals("Outputs", resValues);
    net.backPropagetion(targValues);
    std::cout << "Network average error: " << net.getAverageError() << std::endl; }
std::cout << std::endl << "Done" << std::endl;
return(0); }
```

Також додаємо логів для виведення в консоль, щоб побачити результат роботи нейронної мережі showVectorVals. Повний код надано у додатку Г.

Для вимірювання часу роботи нейронної мережі скористаємося стандартною бібліотекою ctime [16].

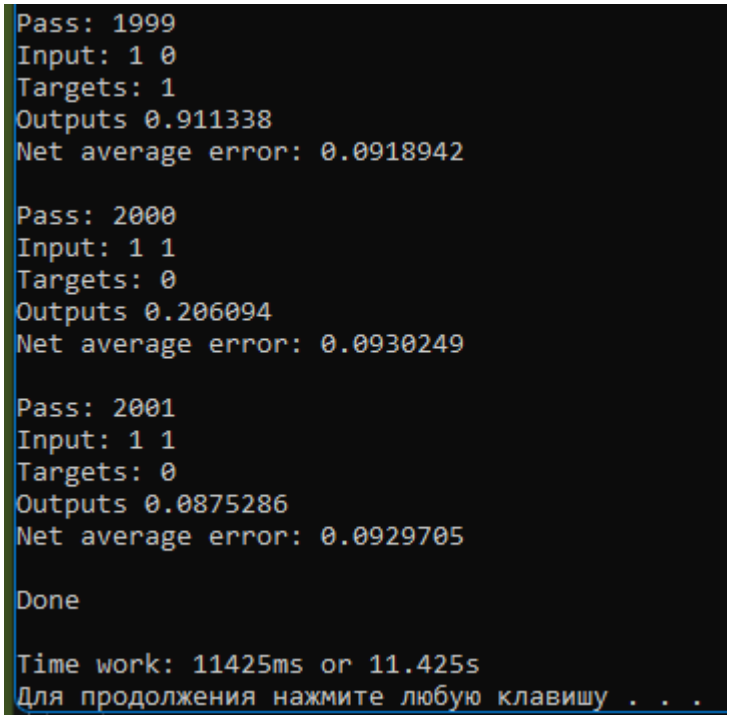
На початку програми запам'ятовуємо час старту програми, наприкінці знову визначаємо час на момент завершення програми та різниця цих значень покаже час роботи нейронної мережі.

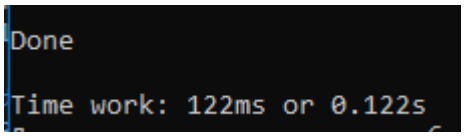
unsigned int start_time = clock(); // Початковий час

// тут має бути фрагмент коду, час виконання якого потрібно виміряти

unsigned int end_time = clock(); // Кінцевий час

unsigned int search_time = end_time - start_time; // шуканий час





Як видно з результату тестування простої нейронної мережі, мережа в результаті видає правильний результат. Для прикладу використовувалася логічна операція NAND, яка видає на виході 0 тільки в тому випадку, якщо обидва входи приймають 1, у всіх інших випадках на виході має бути 0, що і спостерігається. Час роботи програми з виведенням логів – 11.4с, без виведення логів – 122мс , ймовірність помилки ~ 0.093 або 9.3%.

3.2 Тестування моделі нейронної мережі після зміни ваг

У пункті 2.3 змінили значення ваги з випадкових генерованих самою програмою на інкрементовані значення.

```
Pass: 1998
Input: 1 0
Targets: 1
Outputs 0.946511
Net average error: 0.0666913

Pass: 1999
Input: 1 0
Targets: 1
Outputs 0.953519
Net average error: 0.0664912

Pass: 2000
Input: 1 1
Targets: 0
Outputs 0.162929
Net average error: 0.067446

Pass: 2001
Input: 1 1
Targets: 0
Outputs 0.00332333
Net average error: 0.0668112

Done

Time work: 12067ms or 12.067s
```

```
Done

Time work: 115ms or 0.115s
```

Як видно з результату тестування моделі нейронної мережі у якій змінили початкові ваги, вона в результаті видає також правильний результат. Але змінився час роботи програми з виведенням логів, тепер він дорівнює ~ 12с, без виведення логів – 115мс ,при цьому знизилась ймовірність помилки ~ 0.066 або 6.6%.

3.3 Тестування моделі нейронної мережі після зміни функції активацій

У пункті 2.4 було змінено функцію активації з гіперболічного тангенсу на арктангенс, та був на Softsign. Протестуємо їх по черзі, спочатку почнемо зі звичайного арктангенсу.

```
Pass: 1998
Input: 1 0
Targets: 1
Outputs 0.976063
Net average error: 0.0169301

Pass: 1999
Input: 1 0
Targets: 1
Outputs 0.976147
Net average error: 0.0169986

Pass: 2000
Input: 1 1
Targets: 0
Outputs 0.0135094
Net average error: 0.0169641

Pass: 2001
Input: 1 1
Targets: 0
Outputs -0.00192451
Net average error: 0.0168152

Done

Time work: 12655ms or 12.655s
```

```
Done

Time work: 120ms or 0.12s
```

Як видно з результату тестування моделі нейронної мережі у якій змінили функцію активації на арктангенс, мережа в результаті видає також правильний результат. Також змінився час роботи програми з виведенням логів, тепер він дорівнює ~ 12.6с, без виведення логів – 120мс ,при цьому ймовірність помилки знизилась ще більше ~ 0.016 або 1.6%.

Тепер змінюємо функцію активації на Softsign.

```
Pass: 1998
Input: 1 0
Targets: 1
Outputs 0.954695
Net average error: 0.104023

Pass: 1999
Input: 1 0
Targets: 1
Outputs 0.954751
Net average error: 0.103441

Pass: 2000
Input: 1 1
Targets: 0
Outputs -0.281961
Net average error: 0.105209

Pass: 2001
Input: 1 1
Targets: 0
Outputs 0.238111
Net average error: 0.106525

Done

Time work: 12092ms or 12.092s
```

```
Done

Time work: 122ms or 0.122s
```

Як видно з результату тестування моделі нейронної мережі у якій змінили функцію активації на Softsign, мережа в результаті видає також правильний результат. Також змінився час роботи програми з виведенням логів, тепер він дорівнює ~ 12с, без виведення логів – 122мс ,при цьому ймовірність помилки знизилась ще більше ~ 0.01 або 1%.

3.4 Висновки

У цьому розділі проводилося тестування найпростішої моделі нейронної мережі. Для навчання цієї мережі використовували раніше створений файл з інформацією на базі якої мережа та проводила навчання. Результати тестування показали, що для підвищення ефективності роботи нейронної мережі потрібно використовувати не стандартні загальноприйняті рішення. При випадкових згенерованих вагах час роботи мережі був 122мс і можливість помилки при отриманні вихідних даних трохи більше 9%. Коли ваги змінили швидкість роботи мережі, трохи збільшилася 115мс і зменшилася ймовірність помилки, тепер вона дорівнює 6.6%. Після того, як змінили функцію активації з гіперболічного тангенсу на арктангенс, час роботи трохи збільшився 120мс, але при цьому ймовірність помилки знизилася до 1.6%. І потім, коли змінили функцію активацію з арктангенсу на Softsign, той час роботи звичайно не сильно змінилося 122мс, а ось ймовірність помилки знизилася ще до 1%. Тобто можна сказати, що нейронна мережа добре пройшла навчання і можливість правильного результату приблизно 99%. Загальні результати тестування моделі продемонстровані в таблиці 3.1.

Таблиця 3.1− Результати тестування

	Швидкість роботи моделі (з виводом інф)	Швидкість роботи моделі (без вивода інф)	Відсоток помилки
Стартова модель нейронної мережі з випадковими вагами та функцією активації гіпербалічний тангенс	11.4с	122мс	9.3%
Модель нейронної мережі з інкриментованими вагами	12с	115мс	6.6%
Модель нейронної мережі з функцією активації arctg	12.6с	120мс	1.6%
Модель нейронної мережі з функцією активації Softsign	12с	122мс	1%

ВИСНОВКИ

У кваліфікаційній роботі була розроблена проста нейронна мережа з архітектурою перцептрон, яка здатна до самого навчання. В роботі розповідалося про існуючи види нейронних мереж, принципи їхньої роботи, навіщо і де їх використовують. Було надано та обумовлено основні функції активації, методи навчання нейронних мереж. У результаті, як і було задумано, вдалося підвищити ефективність роботи нейронної мережі завдяки зміні ваг та функцій активацій на arctg та Softsign. Після змін функцій активацій та ваг мережа показувала кращі результати, ніж на початку зі звичайними вагами та функціями активації. Функції активації в яких використовувались arctg та Softsign відпрацювали краще ніж звичайні функція активації (в цій роботі це гіперболічний тангенс) завдяки тому, що ці тригонометричні функції розраховуються простіше та результат, як правило, точніше. Дана модель нейронної мережі має свої переваги та недоліки, якщо її масштабувати або дати інші варіанти навчання, то результати можуть трохи відрізнятися від отриманих, але на простих завданнях, такі як визначити шаблон і видати по ньому результат наша мережа цілком впорається показую хороші результати. Переваги цієї мережі в тому, що вона легка у використанні та реалізації, і за підсумком видає задовільні показники на виході (швидкість роботи та правильність результату). Але як говорилося раніше, якщо починати масштабувати мережу для складніших завдань, її вже використовувати не можна буде. Ця мережа не розрахована такі завдання як визначення образів, оптимізація даних чи прийняття рішень. Для створення таких нейромереж вже потрібно переробляти архітектуру. При цьому буде не один-два приховані рівні, а набагато більше. В результаті вдалося збільшити працездатність мережі, але потрібно пам'ятати що в першу чергу ефективність роботи нейронної мережі буде залежати від того, де ця мережа працює.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Алгоритми навчання нейронних мереж [Електронний ресурс] - Режим доступу до ресурсу: https://uadoc.zavantag.com/text/24469/index-1.html

2. Гослинг, Джеймс, Джой, Билл, Стил, Гай, Брача, Гилад, Бакли, Алекс. Язык программирования Java SE8. Подробное описание, 5-е изд: Пер. с англ – М. ООО «И.Д.Вильямс», 2015. – 672 с.

3. Нейронная сеть [Електронний ресурс] - Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Нейронная_сеть

4. Функції активації нейромережі: сигмоїда, лінійна, ступінчаста, ReLu, than [Електронний ресурс] - Режим доступу до ресурсу: https://neurohive.io/ru/osnovy-data-science/activation-functions/

5. Поліпшення продуктивності нейронної мережі [Електронний ресурс] - Режим доступу до ресурсу: https://www.machinelearningmastery.ru/how-to-increase-the-accuracy-of-a-neural-network-9f5d1c6f407d/

6. Нейронные сети, перцептрон [Електронний ресурс] - Режим доступу до ресурсу: https://neerc.ifmo.ru/wiki/index.php?title=Нейронные_сети_перцептрон

7. Что такое нейросеть [Електронний ресурс] - Режим доступу до ресурсу: https://secretmag.ru/enciklopediya/chto-takoe-neiroset-obyasnyaem-prostymi-slovami.htm

8. Функции активации [Електронний ресурс] - Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Функции_активации

9. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. — СПб.: Питер, 2018. — 480 с.

10. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение = Deep Learning. — М.: ДМК-Пресс, 2017. — 652 с.

11. Ясницкий Л. Н. Введение в искусственный интеллект. — М.: Издат. центр «Академия», 2005. — 176 с.

12. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Proceeding ICCV’15 Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV). — Washington: IEEE Computer Society, 2015. — С. 1026-1034.

13. Хайкин С. Нейронные сети: полный курс = Neural Networks: A Comprehensive Foundation. 2-е изд. — М.: Вильямс, 2006. — 1104 с.

14. Обучи себя сам. Что такое нейронные сети [Електронний ресурс] - Режим доступу до ресурсу: https://dit.urfu.ru/ru/blog/28689

15. Уоссермен, Ф. Нейрокомпьютерная техника: Теория и практика Neural Computing. Theory and Practice. — М.: Мир, 1992. — 240 с.

16. Измерение времени с ctime [Електронний ресурс] - Режим доступу до ресурсу: http://cppstudio.com/post/468/

17. Сайт для вивчення С++ [Електронний ресурс] - Режим доступу до ресурсу: http://cppstudio.com/

18. Прата Стивен,П70 Язык Программирования С++, Лекции и упражнения, 6-е узд.: Пер. С анг. – М. : ООО «И.Д.Вильямс», 2018. – 1248 с.: ил. – Парал. тит. Англ.

19. Нейронные сети для начинающих. Часть 1 [Електронний ресурс] - Режим доступу до ресурсу: https://habr.com/ru/post/312450/

20. Bengio, Yoshua. Learning Deep Architectures for AI // Foundations and Trends in Machine Learning. 2 (1): 1–127. — 2009.

21. Schmidhuber, J. Deep Learning in Neural Networks: An Overview // Neural Networks. 61: 85–117. — 2015. — doi:10.1016/j.neunet.2014.09.003. — arXiv:1404.7828. — PMID 25462637.

22. Szegedy, Christian; Toshev, Alexander; Erhan, Dumitru. Deep neural networks for object detection // Advances in Neural Information Processing Systems. — 2013. — С. 2553–2561.

23. Hof, Robert D. Is Artificial Intelligence Finally Coming into Its Own? // MIT Technology Review. Retrieved 2018-07-10.

24. Глубокое обучение [Електронний ресурс] - Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Глубокое_обучение

25. Хайкин Саймон. Нейронные сети: полный курс, 2-е изд./Пер. с англ. М. :ООО: «И.Д. Вильямс», 2016. — 1104 с.

26. Методи розпізнавання обличчя людини нейронними мережами [Електронний ресурс] - Режим доступу до ресурсу: https://science.donntu.edu.ua/sii/klimenko/diss/indexu.htm

27. ВИВЧАЄМО НЕЙРОННІ МЕРЕЖІ ЗА ЧОТИРИ КРОКИ [Електронний ресурс] - Режим доступу до ресурсу: https://videoplays.ru/uk/what-is-spam/izuchaem-neironnye-seti-za-chetyre-shaga/

28. Моделювання систем за допомогою нейронних мереж. Вирішувані задачі. [Електронний ресурс] - Режим доступу до ресурсу: https://infopedia.su/17xc019.html

29. Глубокие нейронные сети: пути применения [Електронний ресурс] - Режим доступу до ресурсу: https://postnauka.ru/longreads/155983

Додаток А

Клас створення мережі

Файл Network.h

```
#pragma once

#include <vector>

class MyNeurn;

typedef std::vector<MyNeurn> Layer;


class Network
{
public:
	Network(const std::vector<unsigned>& myTopol);
	void feedForw(const std::vector<double>& inpValues);
	void backPropagetion(const std::vector<double>& targValues);
	void getRes(std::vector<double>& resValues) const;
	double getAverageError() const { return averageError; }
private:
	std::vector<Layer> layers;
	double error;
	double averageError;
	static double averageFactor;

private:

};
```

Файл Network.cpp

```
#include "Network.h"
#include "MyNeurn.h"
#include "TrainingSet.h"

double Network::averageFactor = 100.0;

Network::Network(const std::vector<unsigned>& myTopol)
{
	unsigned numLayers = myTopol.size();
	for (unsigned layerNum = 0; layerNum < numLayers; ++layerNum) {
		layers.push_back(Layer());
		unsigned numOutputs = layerNum == myTopol.size() - 1 ? 0 : myTopol[layerNum
+ 1];


		for (unsigned MyNeurnNum = 0; MyNeurnNum <= myTopol[layerNum];
++MyNeurnNum) {
			layers.back().push_back(MyNeurn(numOutputs, MyNeurnNum));
		}

		layers.back().back().setOutputVal(1.0);
	}}

void Network::getRes(std::vector<double>& resValues) const

{
	resValues.clear();
```

```
	for (unsigned n = 0; n < layers.back().size() - 1; ++n)
	{
		resValues.push_back(layers.back()[n].getOutputVal());
}}

void Network::backPropagetion(const std::vector<double>& targValues)
{

	Layer& outputLayer = layers.back();
	error = 0.0;

	for (unsigned n = 0; n < outputLayer.size() - 1; ++n)
	{
		double delta = targValues[n] - outputLayer[n].getOutputVal();
		error += delta * delta;
	}
	error /= outputLayer.size() - 1;
	error = sqrt(error);


	averageError =
		(averageError * averageFactor + error)
		/ (averageFactor + 1.0);


	for (unsigned n = 0; n < outputLayer.size() - 1; ++n)
	{
		outputLayer[n].calcOutputGradients(targValues[n]);
	}

	for (unsigned layerNum = layers.size() - 2; layerNum > 0; --layerNum)
	{
		Layer& hiddenLayer = layers[layerNum];
		Layer& nextLayer = layers[layerNum + 1];

		for (unsigned n = 0; n < hiddenLayer.size(); ++n)
		{
			hiddenLayer[n].calcHiddenGradients(nextLayer);
		}}


	for (unsigned layerNum = layers.size() - 1; layerNum > 0; --layerNum)
	{
		Layer& layer = layers[layerNum];
		Layer& prevLayer = layers[layerNum - 1];

		for (unsigned n = 0; n < layer.size() - 1; ++n)
		{
			layer[n].updateInputWeights(prevLayer);
		}
	}
}

void Network::feedForw(const std::vector<double>& inpValues)
{
	assert(inpValues.size() == layers[0].size() - 1);

for (unsigned i = 0; i < inpValues.size(); ++i) {
		layers[0][i].setOutputVal(inpValues[i]);
	}

	for (unsigned layerNum = 1; layerNum < layers.size(); ++layerNum) {
		Layer& prevLayer = layers[layerNum - 1];
		for (unsigned n = 0; n < layers[layerNum].size() - 1; ++n) {
```

```
        layers[layerNum][n].feedForw(prevLayer);
}}}
```

Додаток Б

Клас створення нейрона

Файл MyNeurn.h

```
#pragma once
#include <vector>

class MyNeurn;

typedef std::vector<MyNeurn> Layer;

struct Connection
{
	double weight;
	double deltaWeight;
};

class MyNeurn
{
public:
	MyNeurn(unsigned numOutputs, unsigned myIndex);
	void setOutputVal(double val) { outputVal = val; }
	double getOutputVal() const { return outputVal; }
	void feedForw(const Layer& prevLayer);
	void calcOutputGradients(double targValues);
	void calcHiddenGradients(const Layer& nextLayer);
	void updateInputWeights(Layer& prevLayer);
private:
	static double eta;
	static double alpha;
	static double randomWeight() { return rand() / double(RAND_MAX); }
	static double activationFunction(double x);
	static double activationFunctionDerivative(double x);
	double sumDOW(const Layer& nextLayer) const;
	double outputVal;
	std::vector<Connection> outputWeights;
	unsigned m_myIndex;
	double gradient;
};
```

Файл MyNeurn.cpp

```
#include "MyNeurn.h"
#include <cmath>


double MyNeurn::eta = 0.15; // чиста швидкість навчання
double MyNeurn::alpha = 0.5; // динамика


MyNeurn::MyNeurn(unsigned numOutputs, unsigned myIndex)
{
	for (unsigned c = 0; c < numOutputs; ++c) {
		outputWeights.push_back(Connection());
outputWeights.back().weight = randomWeight();
	}

	m_myIndex = myIndex;
```

```
}

void MyNeurn::updateInputWeights(Layer& prevLayer)
{
	for (unsigned n = 0; n < prevLayer.size(); ++n)
	{
		MyNeurn& MyNeurn = prevLayer[n];
		double oldDeltaWeight = MyNeurn.outputWeights[m_myIndex].deltaWeight;

		double newDeltaWeight = eta * MyNeurn.getOutputVal() * gradient + alpha *
oldDeltaWeight;
		MyNeurn.outputWeights[m_myIndex].deltaWeight = newDeltaWeight;
		MyNeurn.outputWeights[m_myIndex].weight += newDeltaWeight;
	}
}

double MyNeurn::sumDOW(const Layer& nextLayer) const
{
	double sum = 0.0;

	for (unsigned n = 0; n < nextLayer.size() - 1; ++n)
	{
		sum += outputWeights[n].weight * nextLayer[n].gradient;
	}

	return sum;
}

void MyNeurn::calcHiddenGradients(const Layer& nextLayer)
{
	double dow = sumDOW(nextLayer);
	gradient = dow * MyNeurn::activationFunctionDerivative(outputVal);
}
void MyNeurn::calcOutputGradients(double targValues)
{
	double delta = targValues - outputVal;
	gradient = delta * MyNeurn::activationFunctionDerivative(outputVal);
}

double MyNeurn::activationFunction(double x)
{
	//output range [-1.0..1.0]
	return tanh(x);
}

double MyNeurn::activationFunctionDerivative(double x)
{
	return 1.0 - x * x;
}

void MyNeurn::feedForw(const Layer& prevLayer)
{
	double sum = 0.0;

	for (unsigned n = 0; n < prevLayer.size(); ++n)
	{
		sum += prevLayer[n].getOutputVal() *
			prevLayer[n].outputWeights[m_myIndex].weight;
	}
	outputVal = MyNeurn::activationFunction(sum);}
```

Додаток В

Клас для навчання

Файл TrainingSet.h

```
#pragma once
#include <vector>
#include <iostream>
#include <cstdlib>
#include <cassert>
#include <cmath>
#include <fstream>
#include <sstream>

class TrainingSet
{
public:
	TrainingSet(const std::string filename);
	bool isEOF() { return trainingDataFile.eof(); }
	void getTopology(std::vector<unsigned>& myTopol);
	unsigned getNextInputs(std::vector<double>& inpValues);
	unsigned getTargetOutputs(std::vector<double>& targetOutputVals);
private:
	std::ifstream trainingDataFile;
};
```

Файл TrainingSet.cpp

```
#include "TrainingSet.h"

TrainingSet::TrainingSet(const std::string filename)
{
	trainingDataFile.open(filename.c_str());
}

void TrainingSet::getTopology(std::vector<unsigned>& myTopol)
{
	std::string line;
	std::string label;

	std::getline(trainingDataFile, line);
	std::stringstream ss(line);
	ss >> label;
	if (this->isEOF() || label.compare("myTopol:") != 0)
	{
		abort();
	}

	while (!ss.eof())
	{
		unsigned n;
		ss >> n;
		myTopol.push_back(n);
	}
	return;
}

unsigned TrainingSet::getNextInputs(std::vector<double>& inpValues)
{
	inpValues.clear();
```

```
        std::string line;
        std::getline(trainingDataFile, line);

std::stringstream ss(line);
        std::string label;
        ss >> label;
        if (label.compare("in:") == 0) {
                double oneValue;
                while (ss >> oneValue) {
                        inpValues.push_back(oneValue);
                }
        }

        return inpValues.size();
}

unsigned TrainingSet::getTargetOutputs(std::vector<double>& targetOutputVals)
{
        targetOutputVals.clear();

        std::string line;
        std::getline(trainingDataFile, line);
        std::stringstream ss(line);

        std::string label;
        ss >> label;
        if (label.compare("out:") == 0) {
                double oneValue;
                while (ss >> oneValue) {
                        targetOutputVals.push_back(oneValue);
                }
        }

        return targetOutputVals.size();
      }
```

Додаток Г

Початок виконання коду

Файл main.cpp

```
#include "TrainingSet.h"
#include "MyNeurn.h"
#include "Network.h"
#include <fstream>

void showVectorVals(std::string label, std::vector<double>& v)
{
	std::cout << label << " ";
	for (unsigned i = 0; i < v.size(); i++)
	{
		std::cout << v[i] << " ";
	}
	std::cout << std::endl;
}

int main()
{
	TrainingSet trainingData("testData.txt");
	std::vector<unsigned> myTopol;
	trainingData.getTopology(myTopol);
	Network net(myTopol);

	std::vector<double> inpValues, targValues, resValues;
	int trainingPass = 0;
	out.open("hello.txt");
	while (!trainingData.isEOF())
	{
		++trainingPass;
		std::cout << std::endl << "Pass: " << trainingPass << std::endl;
		if (out.is_open())
		{
			out << "Pass: " << trainingPass << std::endl;
		}

		if (trainingData.getNextInputs(inpValues) != myTopol[0])
			break;
		showVectorVals("Input:", inpValues);
		net.feedForw(inpValues);

		trainingData.getTargetOutputs(targValues);
		showVectorVals("Targets:", targValues);
		assert(targValues.size() == myTopol.back());
		net.getRes(resValues);
		showVectorVals("Outputs", resValues);

		net.backPropagetion(targValues);

		std::cout << "Network average error: " << net.getAverageError() <<
std::endl;
		if (out.is_open())
		{
			out << "Network average error: " << net.getAverageError() <<
std::endl;
			out << std::endl;
		}}
	std::cout << std::endl << "Done" << std::endl;
```

```
#if defined(_MSC_VER) || defined(_WIN32)
system("PAUSE");
#endif
       return(0);}
```