КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

«До захисту допущено»
Завідувач кафедри
Ю. В. Крак _____
(підпис)

«___» _____20___ р.

Дипломна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ЗНАХОДЖЕННЯ СТАЛИХ ВИРАЗІВ В ПЕРСОНАЛЬНИХ ТЕКСТАХ

Виконав студент 4 курсу
Гаврилов Сергій Дмитрович _____
(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Крак Юрій Васильович _____
(підпис)

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2021

РЕФЕРАТ

Робота складається зі вступу, 3 розділів, висновку, списку використаних джерел (86 найменувань). Робота містить 6 рисунків. Загальний обсяг становить 54 сторінки, основний текст роботи викладено на 44 сторінках.

Ключові слова: МАШИННЕ НАВЧАННЯ, ТЕКСТОВИЙ АНАЛІЗАТОР, МОВА ПРОГРАМУВАННЯ PYTHON, ДІАГРАМА КЛАСІВ, ГРАФІЧНИЙ ІНТЕРФЕЙС.

Об’єктом роботи є методи і технології машинного навчання. Предметом роботи є обробка текстових файлів методами машинного навчання. Мета роботи полягає в розробленні системи пошуку сталих виразів в текстах методами машинного навчання з використанням засобів мови C#.

В якості засобу розроблення системи було обрано Visual Studio 2019 та мову програмування C#.

В процесі роботи було проаналізовано методику пошуку сталих виразів в текстових файлах різними способами і виявлено механізм пошуку підрядка зі зміною регістру тексту на нижній та видаленням знаків пунктуації і табуляції.

Результатом роботи став програмний продукт, який може бути використаний в реальних умовах для пошуку сталих виразів в тексті, а при правильному навчанні може слугувати для пошуку плагіату або копіювань тексту, тощо.

ЗМІСТ

ВСТУП

В сучасному світі, зважаючи на хід діджиталізації і автоматизації усіх життєвих процесі людства, починаючи від побутових необхідностей, на кшталт замовлення продуктів харчування і закінчуючи серйозними промисловими процесами, можна помітити тенденцію розвитку явища машинного навчання і використання його в різного роду автоматизаційних системах.

Актуальність явища машинного навчання росте з кожним роком, інтерес до галуз підвищується, що зумовлює його біль часте використання.

Виходячи з описаної актуальності явища, метою роботи було обрано розробку системи пошуку сталих виразів в текстах з використанням методик машинного навчання.

Для досягнення поставленої мети слід виконати наступні завдання:

- Проаналізувати поняття нейронної мережі
- Проаналізувати поняття штучних нейронів
- Провести огляд парадигм навчання нейронних мереж
- Розглянути сфери використання нейронних мереж
- Проаналізувати поняття синапсу
- Проаналізувати правило корекції за помилкою
- Розглянути поняття функції активації
- Обрати мову програмування
- Обрати середовище розробки
- Побудувати діаграму використання
- Розробити діаграму класів
- Створити графічний інтерфейс користувача

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

1.1. Поняття машинного навчання

Машинне навчання (ML) - це вивчення комп’ютерних алгоритмів, які автоматично вдосконалюються завдяки досвіду та використанню даних. [1] Він розглядається як частина штучного інтелекту. Алгоритми машинного навчання будують модель на основі зразкових даних, відомих як "навчальні дані", для того, щоб робити прогнози або приймати рішення, не будучи явно запрограмованими для цього. [2] Алгоритми машинного навчання використовуються в найрізноманітніших додатках, таких як медицина, фільтрація електронної пошти та комп'ютерний зір, де важко або нездійсненно розробити звичайні алгоритми для виконання необхідних завдань. [3]

Підмножина машинного навчання тісно пов’язана з обчислювальною статистикою, яка фокусується на прогнозуванні за допомогою комп’ютерів; але не все машинне навчання є статистичним навчанням. Вивчення математичної оптимізації доставляє методи, теорію та сфери застосування в область машинного навчання. Інтелектуальний аналіз даних є суміжною галуззю досліджень, зосередженою на аналізі дослідницьких даних за допомогою неконтрольованого навчання. [5] [6] У своєму застосуванні до бізнес-проблем машинне навчання також називають прогнозною аналітикою.

Машинне навчання передбачає виявлення комп’ютерами того, як вони можуть виконувати завдання, не будучи явно запрограмованими для цього. Він передбачає навчання комп’ютерів на основі даних, що дозволяють виконувати певні завдання. Для простих завдань, покладених на комп’ютери, можна запрограмувати алгоритми, що повідомляють машині, як виконувати всі кроки,

необхідні для вирішення сучасної проблеми; з боку комп'ютера, навчання не потрібно. Для більш складних завдань для людини може бути складним завданням створювати вручну необхідні алгоритми. На практиці це може виявитись більш ефективним, щоб допомогти машині розробити власний алгоритм, а не вимагати від програмістів, що визначають кожен необхідний крок. [7]

Дисципліна машинного навчання використовує різні підходи для навчання комп’ютерів виконанню завдань, коли не існує повністю задовільного алгоритму. У випадках, коли існує величезна кількість потенційних відповідей, одним із підходів є позначення деяких правильних відповідей як дійсних. Потім це можна використовувати як навчальні дані для комп’ютера для вдосконалення алгоритму (алгоритмів), який він використовує для визначення правильних відповідей. Наприклад, для підготовки системи для завдання цифрового розпізнавання символів часто використовується набір даних від рукописних цифр MNIST [7].

Термін машинне навчання було введено в життя в 1959 році Артуром Самуелем, американським IBMer і піонером у галузі комп'ютерних ігор та штучного інтелекту [8] [9]. Репрезентативною книгою досліджень машинного навчання протягом 1960-х років була книга Нільссона про навчальні машини, що стосується переважно машинного навчання для класифікації шаблонів [10]. Інтерес, пов’язаний з розпізнаванням зразків, тривав і в 1970-х роках, як це описали Дуда та Харт у 1973 р. [11] У 1981 р. Було дано звіт про використання стратегій навчання, щоб нейронна мережа навчилася розпізнавати 40 символів (26 букв, 10 цифр і 4 спеціальні символи) з комп'ютерного терміналу. [12]

Том М. Мітчелл навів широко цитоване, більш офіційне визначення алгоритмів, що вивчаються в галузі машинного навчання: "Комп'ютерна програма, як кажуть, вчиться на досвіді E стосовно певного класу завдань T і міри продуктивності P, якщо її ефективність при виконанні завдань в T, як

вимірюється P, покращується з досвідом E. "[13] Це визначення завдань, що стосуються машинного навчання, пропонує принципово оперативне визначення, а не визначення сфери в когнітивному плані. Це слідує пропозиції Алана Тьюрінга в його роботі "Обчислювальні машини та інтелект", в якій питання "Чи можуть машини мислити?" замінюється запитанням "Чи можуть машини робити те, що ми (як мислячі особи) можемо робити?". [14]

Сучасне машинне навчання має дві цілі, одна - класифікувати дані на основі розроблених моделей, інша мета - робити прогнози щодо майбутніх результатів на основі цих моделей. Гіпотетичний алгоритм, специфічний для класифікації даних, може використовувати комп'ютерне бачення родимок у поєднанні з контрольованим навчанням, щоб навчити його класифікувати ракові родимки. Де як, алгоритм машинного навчання для торгівлі акціями може інформувати торговця про майбутні потенційні прогнози. [15]

1.2. Поняття штучного інтелекту

Як наукова діяльність, машинне навчання виросло із пошуків штучного інтелекту. У перші дні ШІ як навчальної дисципліни деякі дослідники були зацікавлені в тому, щоб машини навчалися на даних. Вони намагалися підійти до проблеми за допомогою різних символічних методів, а також того, що тоді називали "нейронними мережами"; це були переважно персептрони та інші моделі, які згодом виявились переосмисленнями узагальнених лінійних моделей статистики [18]. Також використовувались імовірнісні міркування, особливо в автоматизованій медичній діагностиці [19]: 488

Однак посилення акценту на логічному підході, заснованому на знаннях, спричинило розрив між ШІ та машинним навчанням. Імовірнісні системи страждали від теоретичних і практичних проблем збору та представлення даних [19]: 488. До 1980 року експертні системи стали домінувати над ШІ, а статистика була не в позі. [20] Робота над символічним / заснованим на знаннях

навчанням продовжувалась у межах ШІ, що призвело до індуктивного логічного програмування, але більш статистичний напрямок досліджень тепер знаходився поза сферою власне ШІ, щодо розпізнавання шаблонів та пошуку інформації [19]: 708–710; 755 Дослідження нейронних мереж були припинені ШІ та інформатикою приблизно в той же час. Цю лінію також продовжили поза сферою AI / CS, як "коннекціонізм", дослідники з інших дисциплін, включаючи Хопфілд, Румельхарт та Хінтон. Їхній головний успіх прийшов у середині 1980-х років із повторним винаходом зворотного розмноження [19]: 25

Машинне навчання (ML), реорганізоване як окрема сфера, почало процвітати в 1990-х. Поле змінило свою мету від досягнення штучного інтелекту до вирішення практичних проблем. Це змістило фокус від символічних підходів, які він успадкував від ШІ, до методів і моделей, запозичених із статистики та теорії ймовірностей. [20]

Починаючи з 2020 року, багато джерел продовжують стверджувати, що машинне навчання залишається підполем ШІ. [21] [22] [16] Основна розбіжність полягає в тому, чи вся МЗ є частиною ШІ, оскільки це означало б, що кожен, хто використовує МЛ, може заявити, що він використовує ШІ. Інші дотримуються думки, що не вся МЗ є частиною ШІ [23] [24] [25], де лише „інтелектуальна“ підмножина МЛ є частиною ШІ. [26]

На запитання, в чому різниця між ML та AI, відповідає Юдея Перл у книзі "Чому". [27] Відповідно ML вивчає та передбачає на основі пасивних спостережень, тоді як AI передбачає взаємодію агента з навколишнім середовищем для навчання та вжиття дій, що максимізують його шанси на успішне досягнення своїх цілей. [30]

Машинне навчання та видобуток даних часто використовують однакові методи і суттєво накладаються, але в той час як машинне навчання фокусується на прогнозуванні, на основі відомих властивостей, отриманих з навчальних даних, видобуток даних зосереджується на виявленні (раніше) невідомих

властивостей даних (це етап аналізу виявлення знань у базах даних). Видобуток даних використовує багато методів машинного навчання, але з різними цілями; з іншого боку, машинне навчання також використовує методи видобування даних як "безконтрольне навчання" або як крок попередньої обробки для підвищення точності навчання. Велика частина плутанини між цими двома дослідницькими спільнотами (які часто мають окремі конференції та окремі журнали, основним винятком є ECML PKDD) походить від основних припущень, з якими вони працюють: у машинному навчанні продуктивність зазвичай оцінюється щодо здатності відтворювати відомі знання, тоді як при виявленні знань та аналізі даних (KDD) ключовим завданням є відкриття раніше невідомих знань. Оцінено щодо відомих знань, неінформований (неконтрольований) метод буде легко перевершений за іншими контрольованими методами, тоді як у типовому завданні KDD контрольовані методи не можуть бути використані через відсутність навчальних даних.

1.3. Використання машинного навчання

1.3.1. Розпізнавання зображень

Розпізнавання зображень - одне з найпоширеніших застосувань машинного навчання. Застосовується для ідентифікації об’єктів, людей, місць, цифрових зображень тощо. Популярним випадком розпізнавання зображень та розпізнавання обличчя є пропозиція автоматичного позначення друзів:

Facebook надає нам функцію автоматичної пропозиції позначення друзів. Щоразу, коли ми завантажуємо фотографію з друзями на Facebook, ми автоматично отримуємо пропозицію позначати тегами з іменем, а технологія, що лежить в основі цього, - це алгоритм розпізнавання та розпізнавання обличчя машинного навчання.

Він заснований на проекті Facebook під назвою "Deep Face", який відповідає за розпізнавання обличчя та ідентифікацію особи на знімку.

1.3.2. Розпізнавання мови

Використовуючи Google, ми отримуємо опцію "Шукати голосом", яка підпадає під розпізнавання мови, і це популярний додаток машинного навчання.

Розпізнавання мови - це процес перетворення голосових вказівок у текст, він також відомий як "Мова в текст", або "Комп'ютерне розпізнавання мови". В даний час алгоритми машинного навчання широко використовуються різними програмами розпізнавання мови. Асистент Google, Siri, Cortana та Alexa використовують технологію розпізнавання мовлення, щоб слідувати голосовим інструкціям.

1.3.3. Прогнозування руху

Якщо ми хочемо відвідати нове місце, ми звертаємось за допомогою до Google Maps, який показує нам правильний шлях із найкоротшим маршрутом та передбачає дорожній рух.

Він передбачає умови дорожнього руху, такі як: чи буде розмитнений, повільний рух чи сильно перевантажений за допомогою двох способів:

Розташування автомобіля в реальному часі з програми Google Map та датчиків

Усі, хто користується картою Google, допомагають цій програмі покращити її. Він отримує інформацію від користувача і надсилає назад до своєї бази даних для покращення продуктивності.

1.3.4. Рекомендації щодо продуктів

Машинне навчання широко використовується різними компаніями електронної комерції та розваг, такими як Amazon, Netflix тощо, для рекомендацій користувачеві щодо продуктів. Щоразу, коли шукаємо якийсь продукт на Amazon, ми починаємо отримувати рекламу того самого продукту під час серфінгу в Інтернеті в одному браузері, і це пов'язано з машинним навчанням. Google розуміє інтерес користувачів, використовуючи різні алгоритми машинного навчання, і пропонує товар відповідно до інтересів

клієнтів. Так само, коли використовуємо Netflix, ми знаходимо деякі рекомендації щодо розважальних серіалів, фільмів тощо, і це також робиться за допомогою машинного навчання.

1.3.5. Самохідні машини

Одним із найбільш захоплюючих застосувань машинного навчання є самокеровані машини. Машинне навчання відіграє значну роль у автомобілях, що самостійно керують автомобілем. Tesla, найпопулярніша компанія з виробництва автомобілів, працює над самокерованим автомобілем. Він використовує метод без нагляду для навчання моделей автомобілів для виявлення людей та предметів під час руху.

1.3.6. Фільтрування спаму та шкідливих програм

Щоразу, коли ми отримуємо нове електронне повідомлення, воно автоматично фільтрується як важливе, нормальне та спам. Ми завжди отримуємо важливу пошту у свою поштову скриньку з важливим символом та електронною поштою у спамі, а технологія, що лежить в основі цього, - це машинне навчання. Нижче наведено кілька фільтрів спаму, які використовує Gmail:

- Фільтр вмісту
- Фільтр заголовка
- Фільтр загальних чорних списків
- Фільтри на основі правил
- Фільтри дозволів

Деякі алгоритми машинного навчання, такі як багатошаровий персептрон, дерево рішень та Naive-Bayes-класифікатор, використовуються для фільтрації спаму електронної пошти та виявлення шкідливих програм.

РОЗДІЛ 2. ПІДХОДИ ДО МАШИННОГО НАВЧАННЯ

Підходи до машинного навчання традиційно поділяються на три широкі категорії, залежно від природи "сигналу" або "зворотного зв'язку", доступних для системи навчання:

Навчання під контролем: Комп’ютер представлений прикладами входів та їх бажаних результатів, наданими «вчителем», і мета полягає у вивченні загального правила, яке відображає входи на виходи.

Навчання без нагляду: алгоритму навчання не присвоюються мітки, що залишає його самостійно знаходити структуру у своєму введенні. Навчання без нагляду може бути самоціллю (виявлення прихованих закономірностей у даних) або засобом досягнення мети (вивчення особливостей).

Навчання підкріплення: комп’ютерна програма взаємодіє з динамічним середовищем, в якому вона повинна виконати певну мету (наприклад, керування транспортним засобом або гра в гру проти супротивника). Орієнтуючись у своєму проблемному просторі, програма отримує зворотний зв’язок, аналогічний винагородам, який вона намагається максимізувати. [4]

2.1. Навчання під наглядом

Керовані алгоритми навчання будують математичну модель набору даних, що містить як входи, так і бажані результати. [38] Дані відомі як навчальні дані і складаються з набору навчальних прикладів. Кожен навчальний приклад має один або кілька входів і бажаний вихід, також відомий як контрольний сигнал. У математичній моделі кожен приклад навчання представлений масивом або вектором, який іноді називають вектором ознак, а дані навчання представлені матрицею. Завдяки ітеративній оптимізації цільової функції керовані алгоритми навчання вивчають функцію, яка може бути використана для прогнозування результату, пов’язаного з новими входами. [39] Оптимальна функція дозволить

алгоритму правильно визначити результати для входів, які не були частиною навчальних даних. Кажуть, що алгоритм, який покращує точність своїх результатів або прогнозів з часом, навчився виконувати це завдання. [13]

Типи керованих алгоритмів навчання включають активне навчання, класифікацію та регресію. [40] Алгоритми класифікації використовуються, коли виходи обмежені обмеженим набором значень, а алгоритми регресії використовуються, коли виходи можуть мати будь-яке числове значення в межах діапазону. Як приклад, для алгоритму класифікації, який фільтрує електронні листи, вхідним сигналом буде вхідне повідоАмлення електронної пошти, а вихідним буде ім'я папки, в яку слід подати електронне повідомлення.

Навчання на схожості - це область керованого машинного навчання, тісно пов’язана з регресією та класифікацією, але мета полягає в тому, щоб навчитися на прикладах із використанням функції подібності, яка вимірює схожість або пов’язаність двох об’єктів. Він має додатки в рейтингу, системах рекомендацій, візуальному відстеженні особистості, верифікації обличчя та верифікації ораторів.

2.2. Навчання без вчителя

Алгоритми навчання без нагляду (вчителя) беруть набір даних, який містить лише вхідні дані, і знаходять у них структуру, наприклад, групування або кластеризацію точок даних. Отже, алгоритми вивчають дані тестів, які не були марковані, класифіковані чи класифіковані. Замість того, щоб відповідати на відгуки, алгоритми навчання без нагляду визначають спільність даних і реагують на основі наявності чи відсутності таких спільностей у кожному новому фрагменті даних. Центральне застосування неконтрольованого навчання - це область оцінки щільності в статистиці, наприклад, пошук функції щільності

ймовірності. [41] Хоча навчання без нагляду охоплює й інші сфери, що включають узагальнення та пояснення особливостей даних.

Кластерний аналіз - це розподіл набору спостережень у підмножини (так звані кластери), щоб спостереження всередині одного кластеру були схожими за одним або кількома попередньо визначеними критеріями, тоді як спостереження, отримані з різних кластерів, відрізняються. Різні методи кластеризації роблять різні припущення щодо структури даних, які часто визначаються певною метрикою схожості та оцінюються, наприклад, внутрішньою компактністю або подібністю між членами одного кластера та розділенням, різницею між кластерами. Інші методи засновані на розрахунковій щільності та зв’язковості графіків.

2.3. Навчання з частковим наглядом

Навчання під наглядом розподіляється між навчанням без нагляду (без будь-яких позначених навчальних даних) та контрольованим навчанням (із повністю позначеними даними про навчання). У деяких прикладах навчання відсутні ярлики навчання, проте багато дослідників машинного навчання виявили, що немічені дані, використовуючи їх разом із невеликою кількістю мічених даних, можуть значно покращити точність навчання.

При слабко контрольованому навчанні ярлики тренувань бувають галасливими, обмеженими або неточними; однак ці ярлики часто дешевше отримувати, що призводить до більших ефективних навчальних наборів. [42]

РОЗДІЛ 3. ВИБІР І ОБГРУНТУВАННЯ ЗАСОБІ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ПРОДУКТУ

3.1. Мова програмування C#

C # (вимовляється як музична нота C♯, але написаний зі знаком числа) - це універсальна багатопарадігмальна мова програмування, що включає строгу типізацію, лексичну область видимості, імператив, декларативний, функціональний, універсальний об'єкт -орієнтовані (на основі класів) та компонентно-орієнтовані дисципліни програмування. Він був розроблений Microsoft приблизно в 2000 році в рамках ініціативи .NET, а потім затверджений в якості міжнародного стандарту Ecma (ECMA-334) і ISO (ISO / IEC 23270: 2018). Mono - це назва безкоштовного проекту з відкритим вихідним кодом для розробки компілятора і середовища виконання для мови. C # є одним з мов програмування, розроблених для Common Language Infrastructure (CLI).

C # був розроблений Андерсом Хейлсбергом, а його командою розробників в даний час керує Мадс Торгерсен. Остання версія - 8.0, випущена в 2019 році разом з Visual Studio 2019 версії 16.3.

Під час розробки .NET Framework бібліотеки класів спочатку були написані з використанням компілятора системи керованого коду під назвою «Simple Managed C» (SMC). У січні 1999 року Андерс Хейлсберг сформував команду для створення нової мови під назвою Cool, який позначав «C-подібна об'єктно-орієнтована мова». Microsoft розглядала збереження назви «Cool» в якості остаточної назви мови, але вирішила не робити цього з причин, пов'язаних з товарними знаками. До того часу, коли проект .NET був публічно оголошено на конференції професійних розробників в липні 2000 року, мову був перейменований в C #, а бібліотеки класів і середовище виконання ASP.NET були портований на C #.

Хейлсберг - головний дизайнер і головний архітектор C # в Microsoft, раніше займався проектуванням Turbo Pascal, Embarcadero Delphi (раніше CodeGear Delphi, Inprise Delphi і Borland Delphi) і Visual J ++. У своїх інтерв'ю і технічних статтях він заявив, що недоліки в більшості основних мов програмування (наприклад, C ++, Java, Delphi і Smalltalk) призвели до основ Common Language Runtime (CLR), що, в свою чергу, призвело до розробки сама мова C #.

Джеймс Гослінг, який створив мову програмування Java в 1994 році, і Білл Джой, співзасновник Sun Microsystems, творця Java, назвали C # «імітацією» Java; Далі Гослінг сказав, що «C # - це свого роду Java з надійністю, продуктивністю і безпекою видалений». Клаус Крефт і Анджеліка Лангер (автори книги про потоках C ++) заявили у своєму блозі, що «Java і C #» це майже ідентичні мови програмування. Нудне повторення без інновацій », « Навряд чи хто-небудь буде стверджувати, що Java або C # є революційними мовами програмування, які змінили спосіб написання програм », а« C # багато запозичив у Java - і навпаки Тепер, коли C # підтримує упаковки і розпаковування, у нас буде дуже схожа функція в Java. "У липні 2000 року Хейлсберг сказав, що C #" не є клоном Java "і" набагато ближче до C ++ "по своєму дизайну.

З часу випуску C # 2.0 в листопаді 2005 року мови C # і Java розвивалися за все більш і більш розбіжним траєкторіях, ставши двома зовсім різними мовами. Одним з перших основних відхилень стало додавання узагальнень до обох мов з абсолютно різними реалізаціями. C # використовує reification для надання «першокласних» універсальних об'єктів, які можна використовувати як будь-який інший клас, з генерацією коду, що виконується під час завантаження класу. Крім того, в C #, додано кілька основних функцій для програмування функціонального стилю, кульмінацією яких є розширення LINQ, випущені в C # 3.0, і підтримуюча його структура лямбда-виразів, методів розширення і

анонімних типів. Ці функції дозволяють програмістам C # використовувати методи функціонального програмування, такі як замикання, коли це вигідно для їх застосування. Розширення LINQ і функціональний імпорт допомагають розробникам скоротити обсяг стандартного коду, який включається в загальні завдання, такі як запити до бази даних, аналіз файлу XML або пошук в структурі даних, зміщуючи акцент на реальну логіку програми, щоб поліпшити читаність і ремонтопридатність.

По своєму дизайну C # є мовою програмування, який безпосередньо відображає основну інфраструктуру спільної мови (CLI). Більшість його внутрішніх типів відповідають типам значень, реалізованих середовищем CLI. Однак специфікація мови не містить вимог до генерації коду компілятора, тобто не вказує, що компілятор C # має орієнтуватися на середовище виконання спільної мови, або генерувати загальний проміжний мова (CIL), або генерувати будь-який інший конкретний формат. Теоретично, компілятор C # може генерувати машинний код, як традиційні компілятори C ++ та Fortran.

C # підтримує строге оголошення типу неявних змінних, використовуючи ключове слово var, і неявно набрані масиви, використовуючи нове ключове слово [] та ініціалізатор набору.

C # підтримує строгий логічний тип даних bool. Оператори, які приймають умови (наприклад, while та if), вимагають вирази типів, що реалізують істинні оператори (наприклад, логічні типи). Хоча C ++ також має логічні типи, його можна вільно перетворювати на цілі числа, цілі числа та вирази. Наприклад, якщо (a) вимагає лише перетворення a в bool, a може бути int або покажчиком. Причина, по якій C # забороняє такий тип "цілих чисел правильно і неправильно", полягає в тому, що примушування програмістів використовувати вирази, які повертають лише логічні значення, може запобігти певним типам помилок програмування, наприклад if (a = b) (використовувати assignment = замість рівного ==).

Типи C # безпечніші за C ++. Єдиними неявними перетвореннями за замовчуванням є ті, які вважаються безпечними перетвореннями за замовчуванням, наприклад цілі розширення. Використовується під час компіляції, під час JIT та в деяких випадках під час виконання. Не існує неявного перетворення між логічними значеннями та цілими числами, а також між членами перерахування та цілими числами (за винятком літералу 0, який можна неявно перетворити на будь-який тип перелічення). На відміну від конструкторів копіювання C ++ та операторів перетворення (які є неявними за замовчуванням), будь-яке кероване перетворення повинно бути чітко позначене як явне або неявне.

На відміну від C ++, C # явно підтримує загальні типи коваріації та коваріації, тоді як C ++ має лише певний ступінь підтримки коваріації завдяки семантиці типу повернення віртуальних методів.

Учасники перерахування розміщуються у своїй області видимості.

Мова C # не допускає глобальних змінних або функцій. Всі методи і члени повинні бути оголошені всередині класів. Статичні члени відкритих класів можуть замінювати глобальні змінні і функції.

Локальні змінні не можуть приховувати змінні вміщує блоку, на відміну від C та C ++.

C # має підтримку строго типізованих покажчиків на функції через ключове слово делегат. Як і псевдо-C ++ фреймворк Qt, в C # є семантика, зокрема навколишнє події стилю публікації-підписки, хоча C # використовує для цього делегати.

Керована пам'ять не може бути звільнена; замість цього він автоматично збирається. Збірка сміття вирішує проблему витоків пам'яті, позбавляючи програміста від відповідальності за звільнення пам'яті, яка більше не потрібна.

На відміну від C ++, C # не підтримує множинне успадкування, хоча клас може реалізовувати будь-яку кількість інтерфейсів. Це було дизайнерське

рішення провідного архітектора мови, щоб уникнути ускладнення і спростити архітектурні вимоги у всьому CLI. При реалізації декількох інтерфейсів, які містять метод з однаковою сигнатурою, і. тобто два методу з одним і тим же ім'ям, що приймають параметри одного й того ж типу в одному і тому ж порядку, C # дозволяє реалізовувати кожен метод в залежності від того, через який інтерфейс викликається цей метод, або, як Java, дозволяє реалізувати метод один раз і мати один виклик за викликом через будь-який з інтерфейсів класу.

Однак, на відміну від Java, C # підтримує перевантаження операторів. Тільки найбільш часто перевантажені оператори в C ++ можуть бути перевантажені в C #.

C # має можливість використовувати LINQ через .NET Framework. Розробник може запросити будь-який об'єкт IEnumerable <T>, документи XML, набір даних ADO.NET і бази даних SQL. [60] Використання LINQ в C # дає такі переваги, як підтримка Intellisense, потужні можливості фільтрації, безпека типів з можливістю перевірки помилок компіляції і узгодженість даних для запиту з різних джерел. Є кілька різних мовних структур, які можна використовувати з C # з LINQ, і вони є виразами запитів, лямбда-виразами, анонімними типами, неявно типізованими змінними, методами розширення і ініціалізаторами об'єктів

У серпні 2001 року корпорації Microsoft, Hewlett-Packard і Intel Corporation виступили співавторами в уявленні специфікації для C #, а також інфраструктури спільної мови (CLI) в організацію по стандартизації Ecma International. У грудні 2001 року ECMA випустила специфікацію мови ECMA-334 C #. C # став стандартом ISO у 2003 році (ISO / IEC 23270: 2003 Інформаційні технології. Мови програмування - C #). ECMA раніше прийняла еквівалентні специфікації як друге видання C # в грудні 2002 року.

У червні 2005 року ECMA схвалив видання 3 специфікації C # і оновило ECMA-334. Доповнення включали в себе часткові класи, анонімні методи, нульові типи і універсальні шаблони (щось схоже на шаблони C ++).

У липні 2005 року ECMA представила в ICO / МЕК JTC 1 за допомогою прискореного процесу останнього стандарти та відповідні ТЗ. Цей процес зазвичай займає 6-9 місяців.

Визначення мови C # і CLI стандартизовані у відповідності зі стандартами ISO і Ecma, які забезпечують розумну та недискримінаційну ліцензійну захист від патентних претензій.

Microsoft погодилася не пред'являти позов розробникам програмного забезпечення з відкритим вихідним кодом за порушення патентів в некомерційних проектах в частині структури, охоплюваній OSP. Microsoft також погодилася не застосовувати патенти на продукти Novell щодо платять клієнтів Novell, за винятком списку продуктів, в яких явно не згадується C#.NET або реалізація Novell .NET (The Mono Project). Тим не менш, Novell стверджує, що Mono не порушує жодних патентів Microsoft. Microsoft також уклала конкретну угоду про відмову у захисту патентних прав, пов'язаних з плагіном браузера Moonlight, який залежить від Mono, якщо він отриманий через Novell.

Microsoft очолює розробку еталонного компілятор C # з відкритим вихідним кодом і набору інструментів, що раніше носили кодова назва "Roslyn". Компілятор, який повністю написаний на керованому коді (C #), був відкритий, а функціональність відображена як API. Це дозволяє розробникам створювати інструменти рефакторінгу та діагностики.

Інші компілятори C # (деякі з яких включають реалізацію інфраструктури спільної мови і бібліотек класів .NET):

• Проект Mono надає компілятор C # з відкритим вихідним кодом, повну реалізацію загальномовної інфраструктури з відкритим вихідним кодом,

включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і майже повну реалізацію пропрієтарних бібліотек класів Microsoft .NET до .NET 3.5. Починаючи з Mono 2.6, планів по впровадженню WPF не існує; WF планується до більш пізнього випуску; і є лише часткові реалізації LINQ to SQL і WCF.

• Проект DotGNU (в даний час припинений) також надав компілятор C # з відкритим вихідним кодом, майже повну реалізацію інфраструктури спільної мови, включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і підмножина деяких залишилися пропрієтарних класів Microsoft .NET. бібліотеки .NET 2.0 (не документовані або не включені в специфікацію ECMA, але включені в стандартний дистрибутив Microsoft .NET Framework).

• Загальна мовна інфраструктура загального ресурсу Microsoft під кодовою назвою «Rotor» забезпечує реалізацію спільного джерела середовища CLR і компілятора C #, ліцензованого тільки для освітніх і дослідницьких цілей, а також підмножина необхідних бібліотек інфраструктури Common Language Infrastructure в специфікації ECMA (вгору в C # 2.0 і підтримується тільки в Windows XP).

C# була обрана основною мовою розробки даного проекту з огляду на усе вищеописане, а саме — на функціонал, який підтримує дана мова, операційні системи, які підтримуються та простота вивчення.

Загалом — функціонал C# найкраще підходить під дану розробку, за рахунок простої реалізації основних принципів ООП та зрозумілого інтерфейсу.

3.2. Visual Studio

Microsoft Visual Studio - це інтегроване середовище розробки Microsoft (IDE). Застосовується для розробки комп’ютерних програм, а також веб-сайтів, веб-додатків, веб-сервісів та мобільних додатків. Visual Studio використовує

платформу Microsoft для розробки програмного забезпечення, такого як Windows API, Windows Forms, Foundation Presentation Foundation, Windows Store та Microsoft Silverlight. Він може генерувати власний код та керований код.

Visual Studio включає редактор коду, який підтримує IntelliSense (компонент розширення коду) та рефакторинг коду. Вбудований налагоджувач може використовуватися як налагоджувач вихідного рівня, а також налагоджувач машинного рівня. Інші вбудовані інструменти включають аналізатор коду, конструктор графічного інтерфейсу, веб-дизайнер, дизайнер класів та конструктор схем баз даних. Він приймає плагіни, які можуть покращити функціональність майже на кожному рівні - включаючи додавання підтримки для систем керування джерелами (наприклад, Subversion і Git) та додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для мов, що задаються доменом або набори інструментів для інших аспектів розробки програмного забезпечення життєвий цикл (як клієнт Azure DevOps: Team Explorer).

Visual Studio підтримує 36 різних мов програмування та дозволяє редактору коду та налагоджувачу підтримувати (в різній мірі) майже будь-яку мову програмування за умови існування послуги, що залежить від мови. Вбудовані мови включають C, C ++, C ++ / CLI, Visual Basic .NET, C #, F #, JavaScript, TypeScript, XML, XSLT, HTML та CSS. Підтримка інших мов, таких як Python, Ruby, Node.js та M серед інших, доступна через плагіни. Java (і J #) підтримувалися в минулому.

Найбільш основне видання Visual Studio, спільноти, доступне безкоштовно. Гасло для видання Visual Studio Community - "Безкоштовне повнофункціональне IDE для студентів, відкритих джерел та індивідуальних розробників".

На даний момент підтримується версія Visual Studio - 2019 рік.

Visual Studio не підтримує жодних мов програмування, рішень чи інструментів. Натомість це дозволяє підключати функції, кодовані як VSPackage. Після встановлення цю функцію можна використовувати як послугу. IDE надає три послуги: SVsSolution, яка забезпечує функцію переліку проектів та рішень; SVsUIShell, що забезпечує функції вікон та користувацьких інтерфейсів (включаючи вкладки, панелі інструментів та вікна інструментів); і SVsShell для реєстрації VSPackages. Крім того, IDE також відповідає за координацію та забезпечення зв'язку між службами. Усі редактори, дизайнери, типи проектів та інші інструменти реалізовані як VSPackages. Visual Studio використовує COM для доступу до VSPackages. Visual Studio SDK також включає в себе Managed Package Framework (MPF), який являє собою набір керованих обгортки навколо інтерфейсу COM, що дозволяє писати пакети будь-якою мовою, сумісною з CLI. Однак MPF не забезпечує всю функціональність, що піддається інтерфейсам Visual Studio COM. Потім послуги можуть бути використані для створення інших пакетів, які додають функціональність Visual Studio IDE.

Підтримка мов програмування додається за допомогою спеціального VSPackage, який називається Language Service. Мовна служба визначає різні інтерфейси, які може реалізувати реалізація VSPackage, щоб додати підтримку різних функціональних можливостей. Функції, які можна додати таким чином, включають забарвлення синтаксису, завершення оператора, відповідність дужок, підказки інформації про параметри, списки членів та маркери помилок для складання фону. Якщо інтерфейс буде реалізований, функціонал буде доступний для мови. Мовні послуги реалізуються на мовній основі. Реалізації можуть повторно використовувати код з аналізатора або компілятора для мови. Мовні сервіси можуть бути реалізовані як у рідному коді, так і керованому коді. Для нативного коду можуть бути використані або рідні інтерфейси COM, або

Babel Framework (частина Visual Studio SDK). Для керованого коду MPF включає обгортки для написання сервісів керованої мови.

Visual Studio не включає вбудовану підтримку управління джерелами, але вона визначає два альтернативних способи інтеграції систем управління джерелами з IDE. VSPackage управління джерелом може забезпечити власний індивідуальний інтерфейс користувача. Навпаки, плагін управління джерелом за допомогою MSSCCI (Microsoft Source Code Control Interface) надає набір функцій, які використовуються для реалізації різних функцій управління джерелом, зі стандартним інтерфейсом користувача Visual Studio. MSSCCI вперше використовувався для інтеграції Visual SourceSafe з Visual Studio 6.0, але згодом був відкритий через SDK Visual Studio. Visual Studio .NET 2002 використовує MSSCCI 1.1, тоді як Visual Studio .NET 2003 використовує MSSCCI 1.2. Visual Studio 2005, 2008 та 2010 використовують MSSCCI версії 1.3, яка додає підтримку для перейменування та видалення дистрибутивів та асинхронного відкриття.

Visual Studio підтримує кілька екземплярів середовища виконання (кожен екземпляр має власну групу VSPackages). Екземпляр використовує різні вулики реєстру (див. Визначення терміну MSDN «вулик реєстру», що використовується тут) для зберігання стану конфігурації та розрізнення його AppId (ідентифікатор програми).). Екземпляр запускається специфічним для AppId .exe, який вибирає AppId, встановлює кореневий кущ та запускає IDE. Пакет програм VSP, зареєстрований для AppId, інтегрований з іншими пакетами програм VSP AppId. Різні версії продуктів Visual Studio були створені за допомогою різних програм. Продукти Visual Studio Express встановлюються із власними AppId, але стандартними продуктами, Professional та Team Suite мають однаковий AppId. Отже, видання Express можна встановити поряд з іншими виданнями, на відміну від інших видань, які оновлюють ту саму установку. Професійне видання включає в себе набір VSPackages у стандартному виданні, а командний набір

включає суперсет VSPackages в обох інших виданнях. Система AppId використовується Visual Studio Shell в Visual Studio 2008.

Дане середовище розробки було обрано по декількох критеріях:

- Підтримка мови C#
- Інтуїтивно зрозумілий інтерфейс
- Можливості графічного редактору, підходящі до задач

Так як усі критерії задовільнено — середовищем розробки було обрано саме Visual Studio 2019.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

4.1. Діаграма варіантів використання

Діаграма використання найпростіша - це представлення взаємодії користувача із системою, яка показує взаємозв'язок між користувачем та різними випадками використання, в яких користувач бере участь. Діаграма випадків використання може ідентифікувати різні типи користувачів системи та різні випадки використання, і часто вона супроводжується також іншими типами діаграм. Варіанти використання представлені кругами або еліпсами.

Незважаючи на те, що сам випадок використання може детально вивчити кожну можливість, діаграма прикладів використання може допомогти забезпечити огляд системи на більш високому рівні. Раніше вже було сказано, що "схеми використання - це принципи вашої системи".

Через їх спрощений характер, схеми використання можуть бути хорошим інструментом комунікації для зацікавлених сторін. Креслення намагаються імітувати реальний світ і дають зацікавленій стороні уявлення про те, як буде розроблена система. Сіау та Лі провели дослідження, щоб визначити, чи взагалі існувала дійсна ситуація для схем використання або вони були непотрібними. Було виявлено, що діаграми випадків використання передають намір системи більш спрощеним чином зацікавленим сторонам і що вони "інтерпретуються більш повно, ніж діаграми класів".

Метою діаграми використання є відображення динамічного аспекту системи. Додаткові схеми та документація можуть бути використані для забезпечення повного функціонального та технічного уявлення про систему. Вони забезпечують спрощене та графічне представлення того, що система насправді повинна робити.

Елементи:

- рамки системи (англ. system border) - прямокутник із назвою у верхніх частинах та еліпсами (прецедентами) всередині. Часто може бути опущено без корисної інформації про корисну інформацію,
- актор (англ. актор) - стилізований людський персонаж, що позначає набір ролей користувача (розуміється в широкому змісті: людина, зовнішня сутність, клас, інша система), взаємодіючого з деякою сутністю (системною, підсистемою, класом). Актори не можуть бути пов'язані між собою з іншим (за вимкнення відносин щодо обробки / дослідження),
- прецедент - еліпс із надписом, що позначає виконувану систематичну дію (може включати можливі варіанти), що призводить до результатів, які спостерігають актори. Надпис може бути ім'ям або описом (з точки зору актора) того, "що" робить система (а не "як"). Ім’я прецедента зв’язано з неперервним (атомарним) сценарієм - конкретною послідовністю дій, що ілюструє поведінку. Під час сценарію актори обмінюються із систематичними повідомленнями. Сценарій може бути приведений на діаграмі прецедентів у відео UML-коментарі. З одним прецедентом може бути пов'язано кілька різних сценаріїв

На рисунку 3.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

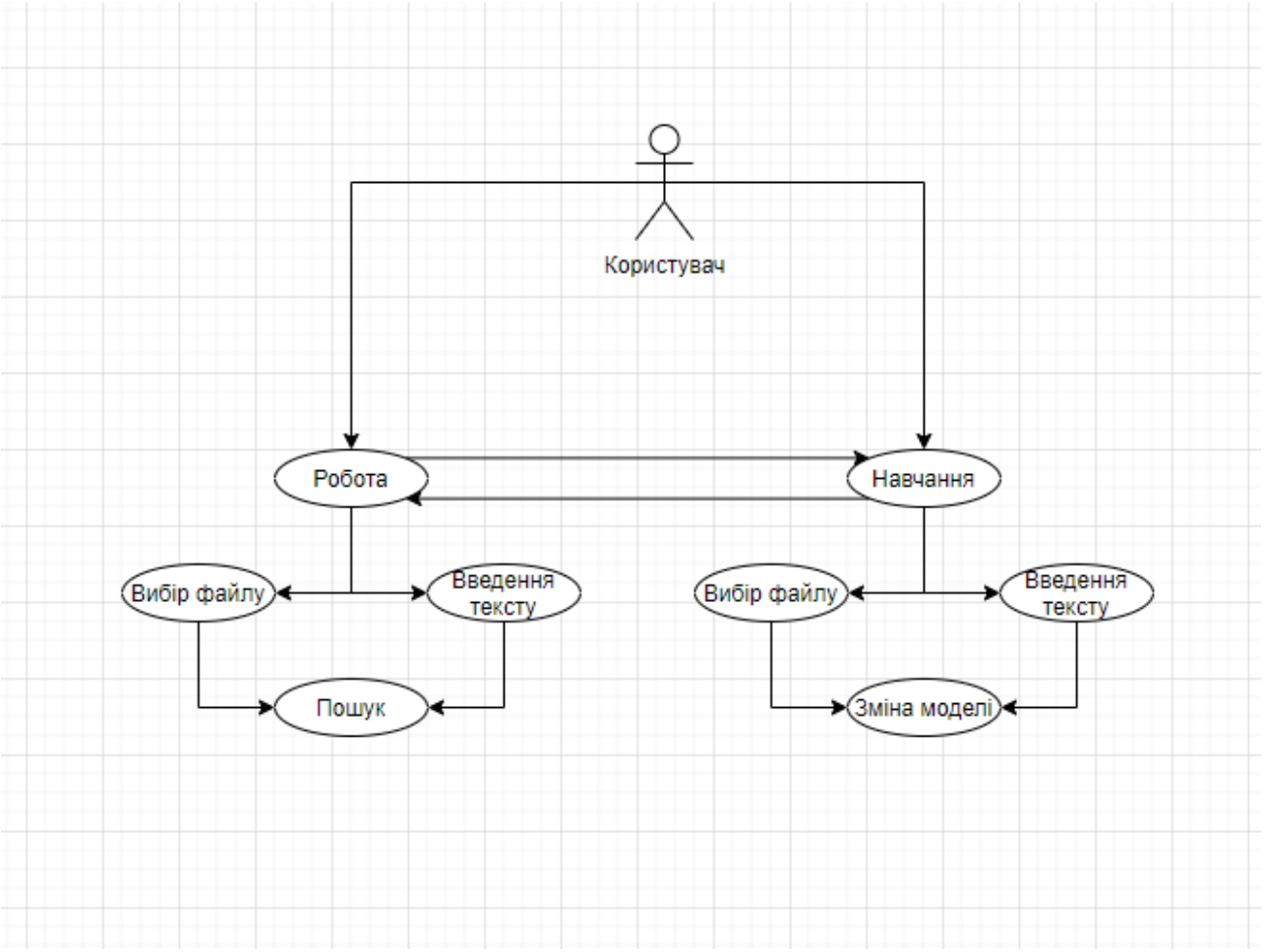

Рис. 4.1 — Діаграма варіантів використання

4.2. Діаграма класів

У програмній інженерії діаграма класів в Уніфікованій мові моделювання (UML) - це тип статичної структурної діаграми, що описує структуру системи, показуючи класи системи, їх атрибути, операції (або методи) та взаємозв'язки між об'єктами.

Діаграма класів є основним будівельним елементом об’єктно-орієнтованого моделювання. Він використовується для загального концептуального моделювання структури програми та для детального моделювання переведення моделей у програмовий код. Діаграми класів також

можуть бути використані для моделювання даних. Класи на діаграмі класів представляють як основні елементи, взаємодії в програмі, так і класи, що програмуються.

На схемі класи представлені вікнами, які містять три відділення:

- У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.
- Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква - мала.

При проектуванні системи ряд класів ідентифікується та згруповується у схему класів, яка допомагає визначити статичні відносини між ними. При детальному моделюванні класи концептуального проекту часто поділяються на ряд підкласів.

Залежність - це семантичний зв’язок між залежними та незалежними елементами моделі. Він існує між двома елементами, якщо зміни у визначенні одного елемента (сервера або цілі) можуть спричинити зміни для іншого (клієнта або джерела). Ця асоціація є односпрямованою. Залежність відображається у вигляді штрихової лінії з відкритою стрілкою, яка вказує від клієнта до постачальника.

Для подальшого опису поведінки систем ці діаграми класів можуть бути доповнені діаграмою стану або машиною стану UML.

Асоціація представляє родину посилань. Двійкова асоціація (з двома кінцями) зазвичай представляється у вигляді рядка. Асоціація може пов’язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціацію можна назвати, а кінці асоціації можна прикрасити

іменами ролей, показниками власності, кратністю, видимістю та іншими властивостями.

Існує чотири різні типи асоціацій: двонаправлена, односпрямована, агрегаційна (включає агрегацію композиції) та рефлексивна. Двонаправлені та односпрямовані асоціації є найбільш поширеними.

Наприклад, клас польоту асоціюється з класом літака двонаправлено. Асоціація представляє статичне відношення, яке ділиться між об'єктами двох класів.

Агрегація є варіантом взаємозв'язку "має"; агрегація є більш конкретною, ніж асоціація. Це асоціація, яка представляє частково цілі або часткові стосунки. Як показано на зображенні, професор "має" клас для викладання. Як тип асоціації, агрегація може бути названа та мати ті самі прикраси, що і асоціація. Однак агрегація не може включати більше двох класів; це має бути бінарна асоціація. Крім того, навряд чи існує різниця між агрегаціями та асоціаціями під час реалізації, і діаграма може взагалі пропустити відносини агрегування. [7]

Агрегація може відбуватися, коли клас є колекцією або контейнером інших класів, але вміщені класи не мають сильної залежності життєвого циклу від контейнера. Вміст контейнера все ще існує, коли контейнер знищений.

В UML він графічно представлений у вигляді порожнистої форми ромба на вміщуючому класі одним рядком, що зв’язує його із вміщеним класом. Сукупність - це семантично розширений об’єкт, який у багатьох операціях трактується як одиниця, хоча фізично він складається з декількох менших об’єктів.

Приклад: Бібліотека та студенти. Тут студент може існувати без бібліотеки, зв'язок між студентом і бібліотекою є агрегацією.

Це вказує на те, що один із двох пов’язаних класів (підклас) вважається спеціалізованою формою іншого (супер тип), а суперклас - узагальненням підкласу. На практиці це означає, що будь-який екземпляр підтипу є також

екземпляром суперкласу. Зразкове дерево узагальнень цієї форми зустрічається в біологічній класифікації: людина - це підклас приматів, які є підкласом ссавців тощо. Зв’язок найлегше зрозуміти за допомогою фрази „А - це В” (людина - це ссавець, ссавець - тварина).

Графічне представлення UML узагальнення - це форма порожнистого трикутника на кінці суперкласу рядка (або дерева рядків), що зв’язує його з одним або кількома підтипами.

Відносини узагальнення також відомі як спадщина або відносини "є".

Суперклас (базовий клас) у відносинах узагальнення також відомий як "батьківський", суперклас, базовий клас або базовий тип.

Підтип у відносинах спеціалізації також відомий як "дочірній", підклас, похідний клас, похідний тип, клас успадкування або тип успадкування.

Зверніть увагу, що ці стосунки нічим не схожі на біологічні стосунки батьків та дітей: використання цих термінів надзвичайно поширене, але може ввести в оману.

А - це тип В

Наприклад, "дуб - це тип дерева", "автомобіль - це тип транспортного засобу"

Узагальнення може бути показано лише на діаграмах класів та на діаграмах використання.

При моделюванні UML взаємозв'язок реалізації - це взаємозв'язок між двома елементами моделі, в яких один елемент моделі (клієнт) реалізує (реалізує або виконує) поведінку, яку вказує інший елемент моделі (постачальник).

Графічне представлення UML реалізації - це порожниста форма трикутника на кінці інтерфейсу штрихової лінії (або дерева рядків), яка з'єднує її з одним або кількома реалізаторами. Проста головка стрілки використовується на кінці інтерфейсу штрихової лінії, що з'єднує її з користувачами. У діаграмах

компонентів використовується графічна умова «м'яч і сокет» (реалізатори виставляють кульку або льодяник, тоді як користувачі показують сокет). Реалізації можна показати лише на діаграмах класів або компонентів. Реалізація - це взаємозв'язок між класами, інтерфейсами, компонентами та пакетами, що з'єднує елемент клієнта з елементом постачальника. Зв'язок реалізації між класами / компонентами та інтерфейсами показує, що клас / компонент реалізує операції, пропоновані інтерфейсом.

Залежність - це слабша форма зв'язку, яка вказує на те, що один клас залежить від іншого, оскільки він використовує його в певний момент часу. Один клас залежить від іншого, якщо незалежний клас є змінною параметра або локальною змінною методу залежного класу. Це відрізняється від асоціації, де атрибут залежного класу є екземпляром незалежного класу. Іноді відносини між двома класами дуже слабкі. Вони взагалі не реалізовані зі змінними-членами. Швидше вони можуть бути реалізовані як аргументи функції-члена.

інший. Ці відносини зазвичай описуються як "А має В" (у матері-кота є кошенята, у кошенят - мати-кішка).

Представлення UML асоціації - це лінія, що з'єднує два пов'язані класи. На кожному кінці рядка є додаткові позначення. Наприклад, ми можемо вказати, використовуючи наконечник стрілки, що загострений кінець видно з хвоста стрілки. Ми можемо вказати власність шляхом розміщення кульки, ролі, яку відіграють елементи цього кінця, вказавши ім'я ролі та множинність екземплярів цієї сутності (діапазон кількості об'єктів, які беруть участь в асоціації з точки зору іншого кінця).

Класи сутності моделюють довгоживучу інформацію, якою обробляє система, а іноді і поведінку, пов'язану з цією інформацією. Їх не слід ідентифікувати як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Як варіант, їх можна намалювати як звичайні класи із позначенням стереотипу «сутність» над назвою класу.

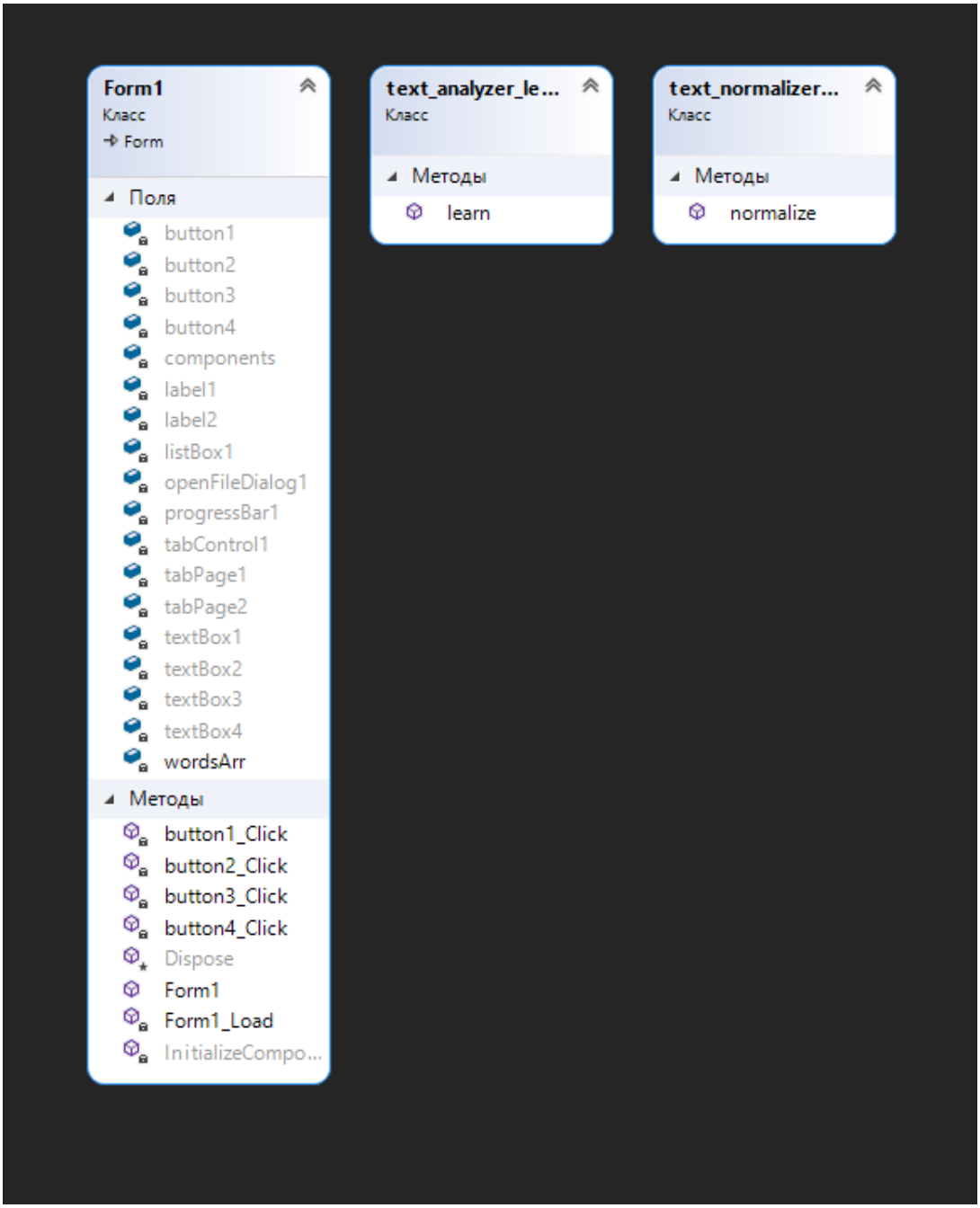

Рис. 4.2 — Діаграма класів

4.3. Графічний інтерфейс користувача

Графічний інтерфейс користувача складається з однієї віконної форми, яка розбита на дві вкладки:

- Вкладка роботи (рис. 4.3, 4.4)
- Вкладка навчання (рис. 4.5 4.6)

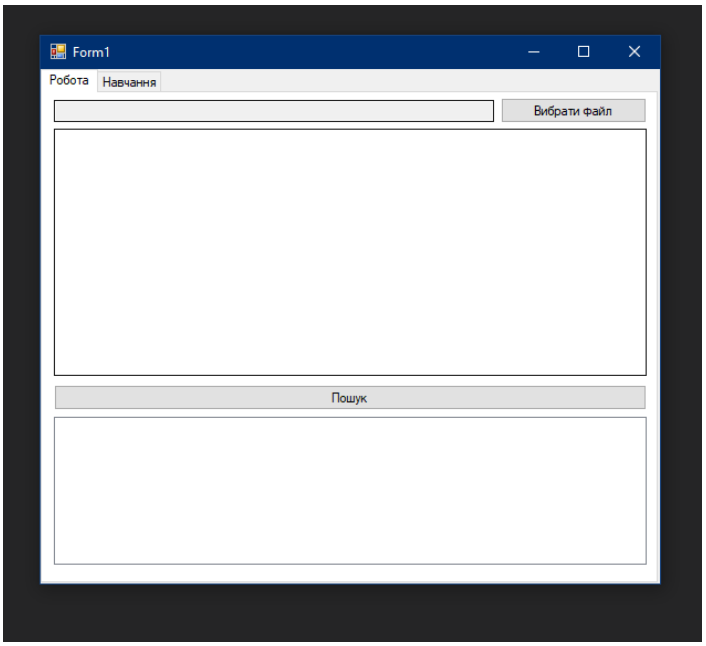

Рис. 4.3 — Порожня вкладка роботи

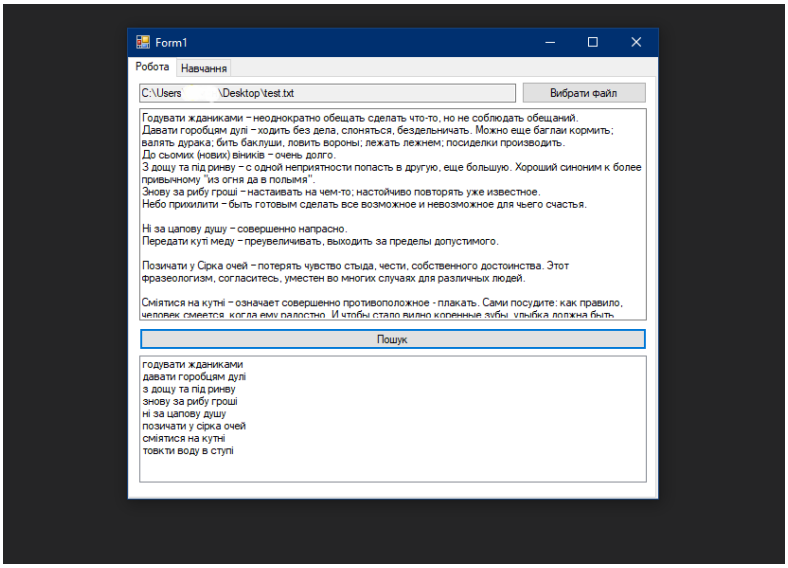

Рис. 4.4 — Вкладка роботи

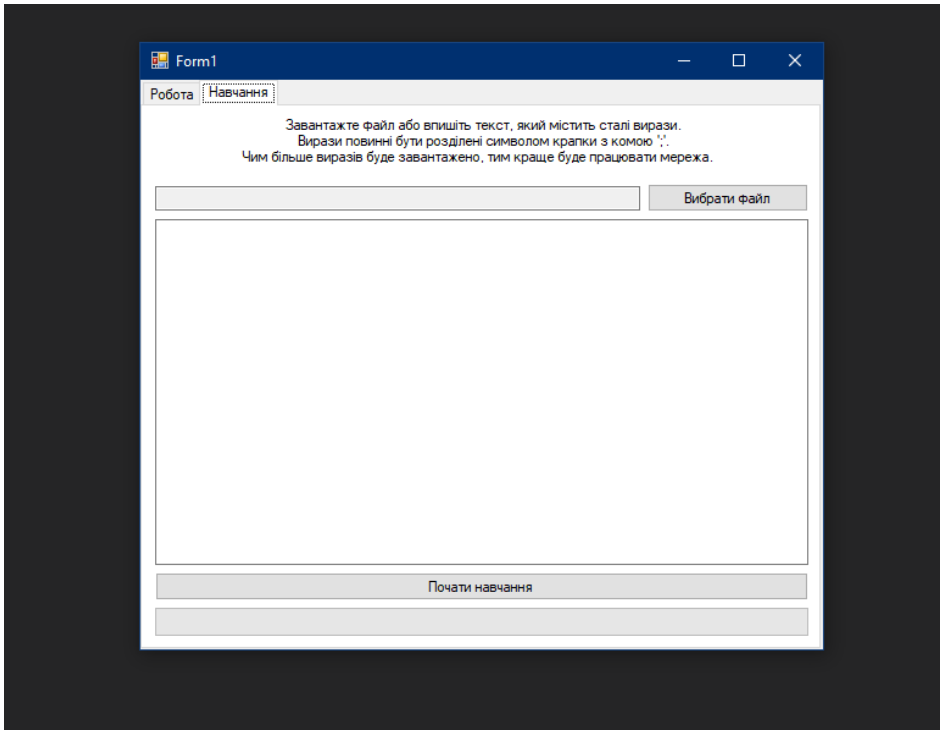

Рис. 4.5 — Вкладка навчання (порожня)

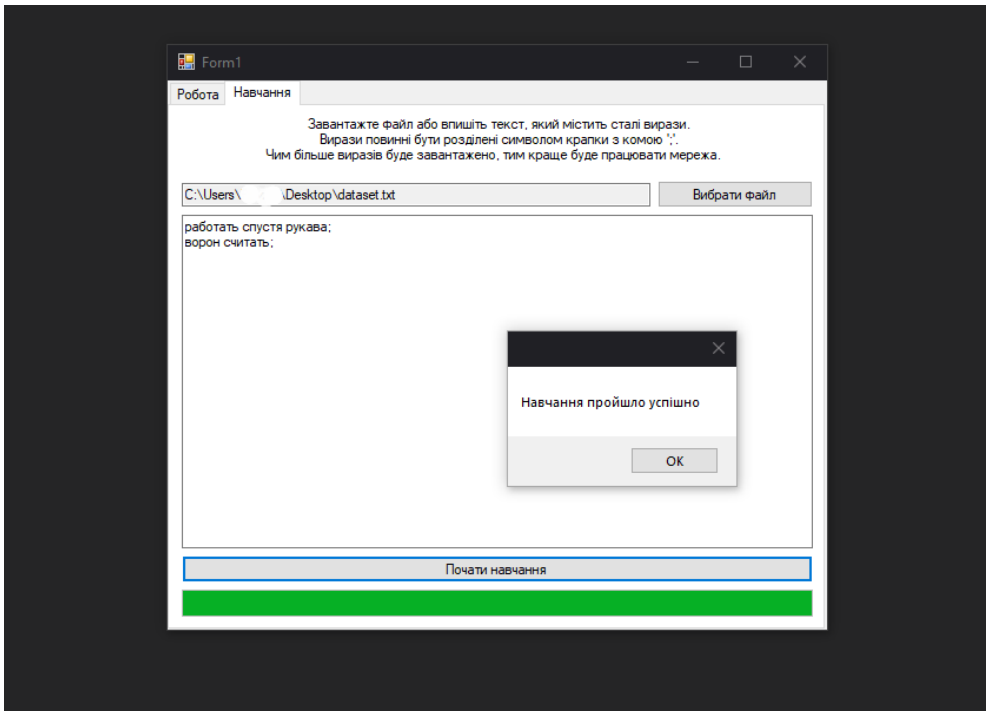

Рис. 4.6 — Вкладка навчання під час роботи

ВИСНОВКИ

В процесі роботи над даним проектом було виконано наступні завдання:

- Проаналізувати поняття нейронної мережі
- Проаналізувати поняття штучних нейронів
- Провести огляд парадигм навчання нейронних мереж
- Розглянути сфери використання нейронних мереж
- Проаналізувати поняття синапсу
- Проаналізувати правило корекції за помилкою
- Розглянути поняття функції активації
- Обрати мову програмування
- Обрати середовище розробки
- Побудувати діаграму використання
- Розробити діаграму класів
- Створити графічний інтерфейс користувача

За рахунок чіткого виконання завдань, поставлених на початку роботи, в результаті отримано повноцінну функціонуючу систему, яка здатна виконувати функції, закладені в неї та готова до використання в реальних умовах за деяких доробок.

ПЕРЕЛІК ВИКОРИСТАНХ ДЖЕРЕЛ

1. McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. 5 (4): 115–133. doi:10.1007/BF02478259.
2. Kleene, S.C. (1956). "Representation of Events in Nerve Nets and Finite Automata". Annals of Mathematics Studies (34). Princeton University Press. pp. 3–41. Retrieved 17 June 2017.
3. Hebb, Donald (1949). The Organization of Behavior. New York: Wiley. ISBN 978-1-135-63190-1.
4. Farley, B.G.; W.A. Clark (1954). "Simulation of Self-Organizing Systems by Digital Computer". IRE Transactions on Information Theory. 4 (4): 76–84. doi:10.1109/TIT.1954.1057468.
5. Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization in the Brain". Psychological Review. 65 (6): 386–408. CiteSeerX 10.1.1.588.3775. doi:10.1037/h0042519. PMID 13602029.
6. Werbos, P.J. (1975). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.
7. Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". Neural Networks. 61: 85–117. arXiv:1404.7828. doi:10.1016/j.neunet.2014.09.003. PMID 25462637. S2CID 11715509.
8. Ivakhnenko, A. G. (1973). Cybernetic Predicting Devices. CCM Information Corporation.
9. Ivakhnenko, A. G.; Grigor′evich Lapa, Valentin (1967). Cybernetics and forecasting techniques. American Elsevier Pub. Co.
10. Schmidhuber, Jürgen (2015). "Deep Learning". Scholarpedia. 10 (11): 85–117. Bibcode:2015SchpJ..1032832S. doi:10.4249/scholarpedia.32832.

11. Dreyfus, Stuart E. (1 September 1990). "Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure". Journal of Guidance, Control, and Dynamics. 13 (5): 926–928. Bibcode:1990JGCD...13..926D. doi:10.2514/3.25422. ISSN 0731-5090.
12. Mizutani, E.; Dreyfus, S.E.; Nishio, K. (2000). "On derivation of MLP backpropagation from the Kelley-Bryson optimal-control gradient formula and its application". Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium. IEEE: 167–172 vol.2. doi:10.1109/ijcnn.2000.857892. ISBN 0-7695-0619-4. S2CID 351146.
13. Kelley, Henry J. (1960). "Gradient theory of optimal flight paths". ARS Journal. 30 (10): 947–954. doi:10.2514/8.5282.
14. "A gradient method for optimizing multi-stage allocation processes". Proceedings of the Harvard Univ. Symposium on digital computers and their applications. April 1961.
15. Linnainmaa, Seppo (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors (Masters) (in Finnish). University of Helsinki. pp. 6–7.
16. Linnainmaa, Seppo (1976). "Taylor expansion of the accumulated rounding error". BIT Numerical Mathematics. 16 (2): 146–160. doi:10.1007/bf01931367. S2CID 122357351.
17. Dreyfus, Stuart (1973). "The computational solution of optimal control problems with time lag". IEEE Transactions on Automatic Control. 18 (4): 383–385. doi:10.1109/tac.1973.1100330.
18. Werbos, Paul (1982). "Applications of advances in nonlinear sensitivity analysis" (PDF). System modeling and optimization. Springer. pp. 762–770.
19. Minsky, Marvin; Papert, Seymour (1969). Perceptrons: An Introduction to Computational Geometry. MIT Press. ISBN 978-0-262-63022-1.

20. Mead, Carver A.; Ismail, Mohammed (8 May 1989). Analog VLSI Implementation of Neural Systems (PDF). The Kluwer International Series in Engineering and Computer Science. 80. Norwell, MA: Kluwer Academic Publishers. doi:10.1007/978-1-4613-1639-8. ISBN 978-1-4613-1639-8.
21. David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams , "Learning representations by back-propagating errors ," Nature', 323, pages 533–536 1986.
22. J. Weng, N. Ahuja and T. S. Huang, "Cresceptron: a self-organizing neural network which grows adaptively," Proc. International Joint Conference on Neural Networks, Baltimore, Maryland, vol I, pp. 576–581, June, 1992.
23. J. Weng, N. Ahuja and T. S. Huang, "Learning recognition and segmentation of 3-D objects from 2-D images," Proc. 4th International Conf. Computer Vision, Berlin, Germany, pp. 121–128, May, 1993.
24. J. Weng, N. Ahuja and T. S. Huang, "Learning recognition and segmentation using the Cresceptron," International Journal of Computer Vision, vol. 25, no. 2, pp. 105–139, Nov. 1997.
25. J. Schmidhuber., "Learning complex, extended sequences using the principle of history compression," Neural Computation, 4, pp. 234–242, 1992.
26. Smolensky, P. (1986). "Information processing in dynamical systems: Foundations of harmony theory.". In D. E. Rumelhart; J. L. McClelland; PDP Research Group (eds.). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. 1. pp. 194–281. ISBN 978-0-262-68053-0.
27. Ng, Andrew; Dean, Jeff (2012). "Building High-level Features Using Large Scale Unsupervised Learning". arXiv:1112.6209 [cs.LG].
28. Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016). Deep Learning. MIT Press.
29. Cireşan, Dan Claudiu; Meier, Ueli; Gambardella, Luca Maria; Schmidhuber, Jürgen (21 September 2010). "Deep, Big, Simple Neural Nets for Handwritten

Digit Recognition". Neural Computation. 22 (12): 3207–3220. arXiv:1003.0358. doi:10.1162/neco_a_00052. ISSN 0899-7667. PMID 20858131. S2CID 1918673.

30. Dominik Scherer, Andreas C. Müller, and Sven Behnke: "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition," In 20th International Conference Artificial Neural Networks (ICANN), pp. 92–101, 2010. doi:10.1007/978-3-642-15825-4_10.
31. 2012 Kurzweil AI Interview Archived 31 August 2018 at the Wayback Machine with Jürgen Schmidhuber on the eight competitions won by his Deep Learning team 2009–2012
32. "How bio-inspired deep learning keeps winning competitions | KurzweilAI". www.kurzweilai.net. Archived from the original on 31 August 2018. Retrieved 16 June 2017.
33. Graves, Alex; and Schmidhuber, Jürgen; Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks, in Bengio, Yoshua; Schuurmans, Dale; Lafferty, John; Williams, Chris K. I.; and Culotta, Aron (eds.), Advances in Neural Information Processing Systems 22 (NIPS'22), 7–10 December 2009, Vancouver, BC, Neural Information Processing Systems (NIPS) Foundation, 2009, pp. 545–552.
34. Graves, A.; Liwicki, M.; Fernandez, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (2009). "A Novel Connectionist System for Improved Unconstrained Handwriting Recognition" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868. CiteSeerX 10.1.1.139.4502. doi:10.1109/tpami.2008.137. PMID 19299860. S2CID 14635907.
35. Graves, Alex; Schmidhuber, Jürgen (2009). Bengio, Yoshua; Schuurmans, Dale; Lafferty, John; Williams, Chris editor-K. I.; Culotta, Aron (eds.). "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks". Neural Information Processing Systems (NIPS) Foundation. Curran Associates, Inc: 545–552.

36. Graves, A.; Liwicki, M.; Fernández, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (May 2009). "A Novel Connectionist System for Unconstrained Handwriting Recognition". IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868. CiteSeerX 10.1.1.139.4502. doi:10.1109/tpami.2008.137. ISSN 0162-8828. PMID 19299860. S2CID 14635907.
37. Ciresan, Dan; Meier, U.; Schmidhuber, J. (June 2012). Multi-column deep neural networks for image classification. 2012 IEEE Conference on Computer Vision and Pattern Recognition. pp. 3642–3649. arXiv:1202.2745. Bibcode:2012arXiv1202.2745C. CiteSeerX 10.1.1.300.3283. doi:10.1109/cvpr.2012.6248110. ISBN 978-1-4673-1228-8. S2CID 2161592.
38. Zell, Andreas (2003). "chapter 5.2". Simulation neuronaler Netze [Simulation of Neural Networks] (in German) (1st ed.). Addison-Wesley. ISBN 978-3-89319-554-1. OCLC 249017987.
39. Artificial intelligence (3rd ed.). Addison-Wesley Pub. Co. 1992. ISBN 0-201-53377-4.
40. "The Machine Learning Dictionary". www.cse.unsw.edu.au. Archived from the original on 26 August 2018. Retrieved 4 November 2009.
41. Abbod, Maysam F (2007). "Application of Artificial Intelligence to the Management of Urological Cancer". The Journal of Urology. 178 (4): 1150–1156. doi:10.1016/j.juro.2007.05.122. PMID 17698099.