МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

А. В. Калюжняк

ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ СИСТЕМИ

Навчальний посібник
для здобувачів ступеня вищої освіти магістра
спеціальності «Комп'ютерні науки»
освітньо-професійної програми «Комп'ютерні науки»

Затверджено
Вченою радою ЗНУ
Протокол №
Від _____ 2024 р.

Запоріжжя
2024

УДК 004.89(075.8)
К178

Калюжняк А. В. Інтелектуальні інформаційні системи : навчальний посібник для здобувачів ступеня вищої освіти магістра спеціальності «Комп'ютерні науки» освітньо-професійної програми «Комп'ютерні науки». Запоріжжя : Запорізький національний університет, 2024. 84 с.

У навчальному посібнику представлені теоретичні та практичні основи створення й використання інтелектуальних інформаційних систем, методи подання та обробки знань в інтелектуальних системах, а також технології проєктування й реалізації інтелектуальних систем.

Навчальне видання складається зі вступу, основної частини, додатків і списків використаної та рекомендованої літератури. В основній частині представлені 6 змістових модулі, в яких наведені теоретичні відомості з інтелектуальних інформаційних систем, нейронних мереж та штучного інтелекту. Для підготовки до контрольних заходів студентам пропонуються контрольні запитання.

Посібник призначений для здобувачів ступеня вищої освіти магістра, які навчаються за освітньо-професійною програмою «Комп'ютерні науки».

Рецензент
О. В. Кудін, кандидат технічних наук, доцент кафедри програмної інженерії

Відповідальний за випуск
Г. М. Шило, доктор технічних наук, доцент, завідувач кафедри комп'ютерних наук

ЗМІСТ

ВСТУП

Характерною особливістю сьогодення є процеси інформатизації, що інтенсивно розвиваються та впроваджуються практично у всіх сферах людської діяльності. Вони призвели до формування нової інформаційної інфраструктури, яка пов'язана з новим типом суспільних відносин та використанням нових інформаційних технологій.

Одним із напрямків розвитку штучного інтелекту є розробка комп'ютерних інтелектуальних систем, здатних виконувати функції, що традиційно вважаються інтелектуальними, – розуміння мови, логічний висновок, використання накопичених знань, навчання, розпізнавання образів, а також навчатися і пояснювати свої рішення. На сьогодні, інформаційні інтелектуальні системи є перспективними у своєму розвитку.

Метою вивчення навчальної дисципліни «Інтелектуальні інформаційні системи» є формування та узагальнення спеціальних знань та навичок у здобувачів ступеня вищої освіти із інтелектуальних системи на основі глибинного навчання, сфери його застосування, аналізу даних методами глибинного навчання.

Основними **завданнями** вивчення дисципліни «Інтелектуальні інформаційні системи» є:

– ознайомлення студентів з основами нейронних мереж;
– ознайомлення студентів з основами глибинного навчання;
– ознайомлення студентів з основами машинного навчання;
– ознайомлення студентів з нейромережевими бібліотеками.

Згідно з вимогами освітньої-професійної програми здобувачі освіти мають досягти наступних компетентностей та програмних результатів навчання:

– здатність формалізувати предметну область певного проєкту у вигляді відповідної інформаційної моделі;
– здатність збирати і аналізувати дані (включно з великими), для забезпечення якості прийняття проєктних рішень;
– здатність застосовувати існуючі і розробляти нові алгоритми розв'язування задач у галузі комп'ютерних наук;
– здатність розробляти та адмініструвати бази даних та знань;
– здатність ініціювати, планувати та реалізовувати процеси розробки інформаційних та комп'ютерних систем та програмного забезпечення, включно з його розробкою, аналізом, тестуванням, системною інтеграцією, впровадженням і супроводом.

Навчальний посібник містить 8 тем, під час опанування яких здобувачі ознайомляться з важливими теоретичними відомостями дисципліни. Наприкінці кожної теми знаходиться контрольний блок, до якого включені дослідницькі завдання (🔔) та контрольні запитання (?).

ЗМІСТОВИЙ МОДУЛЬ 1. ОСНОВИ ГЛИБИННОГО НАВЧАННЯ

Тема 1. Основи глибинного навчання. Машинне та глибинне навчання

1.1 Базові поняття штучного інтелекту

Ідея штучного інтелекту з'явилася в 1950-х, коли група ентузіастів з тільки зародженої галузі інформатики ставили собі питання, чи можна змусити комп'ютери «думати», питанням, наслідки якого ми вивчаємо дотепер. Коротко цю галузь можна визначити так: *автоматизація інтелектуальних завдань, зазвичай виконуваних людьми*. Відповідно, *штучний інтелект (ШІ*) – це галузь, що охоплює машинне навчання і глибоке навчання, а також містиить багато підходів, не пов'язаних з навчанням. Наприклад, перші програми для гри в шахи діяли за жорстко певними правилами, заданими програмістами, і не могли кваліфікуватися як машинне навчання. Довгий час багато експертів вважали, що ШІ рівня людини можна створити, якщо дати програмісту достатній набір явних правил для маніпулювання знаннями. Цей підхід, відомий як *символічний ШІ*, і був парадигмою, що домінує з 1950-х до кінця 1980-х. Пік його популярності припав на бум *експертних систем* в 1980-х [4].

Символічний ШІ прекрасно справлявся з розв'язуваням чітко визначених логічних задач, таких як гра в шахи, але, як виявилося, неможливо поставити строгі правила для розв'язання складніших, нечітких задач, таких як класифікація зображень, розпізнавання мови й переклад на інші мови. На зміну символічного ШІ з'явився новий підхід: *машинне навчання*.

Насамперед, **Штучний інтелект** – один з найперспективніших напрямків комп'ютерних наук, який вивчає методи розв'язання задач, для яких не існує способів вирішення. Системи штучного інтелекту можуть оперувати даними та самонавчатися. Сфери застосування таких систем є необмеженими – від створення роботів, які самостійно приймають рішення, до машин з автопілотом чи онлайн-перекладачів в реальному часі.

Загалом поняття «штучний інтелект» є досить розмитим. Практично вся сучасна техніка обладнується мікрочипами, а виробники переконують споживачів про наявність ШІ в їх виробах. Але, в більшості випадків це є просте копіювання людиноподібної лінії поведінки на штучно створеному об'єкті для зменшення витрат і часу людини.

Основні поняття, які використовуються у сфері штучного інтелекту, є поняття **інтелекту**, **штучного інтелекту** та **інтелектуальної задачі**. Наведемо їх визначення.

Рисунок 1.1 – Роботи, що працюють на базі штучного інтелекту

Термін **інтелект** (*Intelligence*) походить від латинського поняття intellectus – розум. **Інтелектом** вважається спроможність мозку до мисленнєвої діяльності, тобто до оперування знаннями для прийняття певних рішень стосовно конкретної задачі.

Більш широке визначення: **інтелект** – це здатність мозку розв'язувати інтелектуальні задачі шляхом набуття, запам'ятовування та цілеспрямованого перетворення знань у процесі навчання, отримання життєвого досвіду та адаптації до різноманітних зовнішніх та внутрішніх обставин. У цьому визначенні «інтелекту» під терміном «знання» мають на увазі не тільки ту інформацію, яка поступає в мозок через органи чуття; зазначена інформація важлива, але недостатня для інтелектуальної діяльності.

Річ у тому, що об'єкти навколишнього середовища володіють властивістю не тільки впливати на органи чуття, але і знаходитися один з одним в певних відносинах. Ясно, що для того, щоб здійснювати в навколишньому середовищі інтелектуальну діяльність (або хоча б просто існувати), необхідно мати знання щодо моделі цього середовища. У цій інформаційній моделі навколишнього середовища реальні об'єкти, їх властивості та відносини між ними не тільки відображаються і запам'ятовуються, але і можуть подумки цілеспрямовано перетворюватися. При цьому важливо, що формування моделі зовнішнього середовища відбувається «у процесі навчання, отриманні життєвого досвіду та адаптації до різноманітних обставин».

У будь-якому інтелекті закладені вихідні моделі, первинна структура зв'язків і основні критерії. Все це забезпечує початок діяльності. Подальший саморозвиток інтелекту залежить від кількості елементів та їх вихідних характеристик, що лежить в основі можливості їх тренування та здатності до утворення зв'язків. Такий інтелект, отримуючи ззовні інформацію, стає здатним до навчання і виховання під дією суспільства та його моделей. Виховання розуміється як зміна первинних критеріїв і формування нових.

Штучний інтелект (*Artificial Intelligence* – AI) визначається як здатність автоматичних систем брати на себе функції людини, вибирати й приймати

оптимальні рішення на основі раніше отриманого життєвого досвіду й аналізу зовнішніх впливів.

Штучний інтелект (ШІ) – характеризується властивістю інтелектуальних систем виконувати *творчі* функції, які традиційно вважаються прерогативою людини; наука і технологія створення інтелектуальних машин, особливо інтелектуальних комп'ютерних програм. ШІ пов'язаний з подібною задачею використання комп'ютерів для розуміння людського інтелекту, але не обов'язково обмежується біологічно правдоподібними методами. Будь-який інтелект спирається на діяльність.

Діяльність мозку – це **мислення**. Інтелект і мислення пов'язані багатьма цілями й завданнями: розпізнавання ситуацій, логічний аналіз, планування поведінки. Характерними особливостями інтелекту є здатність до навчання, узагальнення, накопичення досвіду, адаптація до умов, що змінюються в процесі вирішення завдань.

Виходячи з самого визначення ШІ виникає основна проблема у створенні інтелекту: можливість або неможливість моделювання мислення дорослої людини або дитини.

1.2 Чи може машина мислити?

Найгарячіші суперечки у філософії штучного інтелекту викликає питання можливості мислення як людина. Питання «Чи може машина мислити?», яке підштовхнуло дослідників до створення науки про моделювання людського розуму, було поставлено Аланом Тюрінгом у 1950 р. [5]. Дві основні думки на це питання носять назви гіпотез сильного і слабкого штучного інтелекту.

Термін «Сильний штучний інтелект» ввів Джон Серль, його ж словами підхід характеризується: «Що більше, така програма буде не тільки моделлю розуму; вона в буквальному розумінні слова сама і буде розумом, в тому ж розумінні, в якому людський розум – це розум…»[12].

З іншого боку, прихильники слабкого штучного інтелекту надають перевагу розгляду програми лише як інструменту, який дозволяє вирішувати ті чи інші задачі, які не потребують повному спектру людських пізнавальних здібностей.

У своєму уявному експерименті «Китайська кімната» Джон Серль демонструє, що проходження тесту Тьюринга не є критерієм наявності істинного процесу мислення. Мислення є процесом опрацювання інформації, яка перебуває у пам'яті: аналіз, синтез і само програмування. Аналогічну позицію займає і Роджер Пенроуз, який в своїй книзі «Новий розум короля» аргументує неможливість отримання процесу мислення на основі формальних систем [1].

Що вважати інтелектом?

Існують різні думки на це запитання. Аналітичний підхід допускає аналіз вищої нервової діяльності людини до нижчої, неподільного рівня (функція вищої нервової діяльності, елементарна реакція на зовнішні подразники (стимули), збудження синапсів сукупності зв'язаних функцією нейронів) і подальше відтворення цих функцій.

За визначенням спеціалістів за інтелект – це здатність раціонального, мотивованого вибору, в умовах недостатньої інформації. Тобто інтелектуальною вважається та програма діяльності (не обов'язково реалізована на сучасних комп'ютерах), яка зможе вибрати із визначеної множини альтернатив, наприклад, куди іти у випадку «наліво підеш …», «направо підеш …», «прямо підеш …».

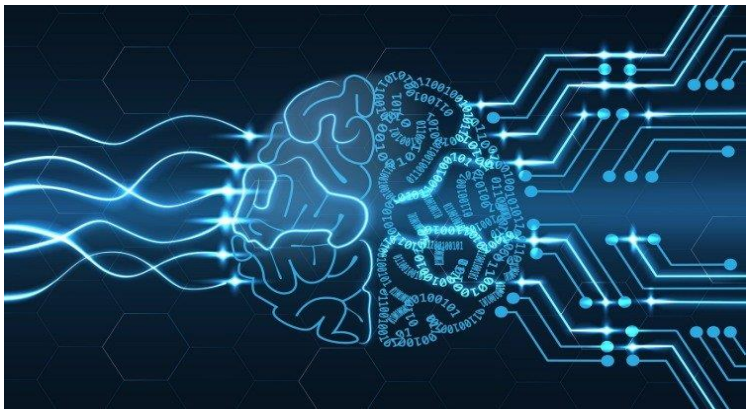


Рисунок 1.2 – Нейрон, який порівняли з нейромережею

Штучний інтелект, на відміну від психології і філософії, займається створенням інтелектуальних штучних істот (сутностей, об'єктів), які називають агентами або носіями. З іншого боку, що поняття «штучний інтелект» не можна зводити лише до створення пристроїв, які імітують людину в усій повноті її діяльності. Насправді ж, спеціалісти які працюють в цій галузі, розв'язують іншу задачу: виявити механізми, які лежать в основі діяльності людини, щоб застосувати їх при розв'язання науково-технічних задач. І це лише одна з можливих проблем.

Вищим проявом штучного інтелекту є продукування самого себе за допомогою технічних засобів та послідовне необмежене зростання його потужності.

Інтелектуальна задача – це процес знаходження алгоритму розв'язанням певного класу задач. Інтелектуальними задачами є задачі, для розв'язання яких немає чітко заданого алгоритму, який завжди приводить до потрібного результату, а інтелектуальною діяльністю можна назвати процес розв'язання інтелектуальних задач. Прикладом інтелектуальної задачі є розпізнавання образів, тобто визначення належності об'єкта, що спостерігається, до однієї із заздалегідь визначених категорій.

Інтелектуальним задачам *властиві* неповнота, неточність та суперечливість знань, а також велика розмірність простору рішень, що не дає змоги розв'язувати їх простим перебором. У таких задачах часто немає чітких критеріїв для вибору оптимального рішення, а сама задача не завжди цілком формалізується.

Основними властивостями інтелектуальних задач є:

– символьне подання умов задачі;

– відсутність чіткої постановки задачі;

– відсутність прийнятного для практичного використання алгоритму рішення, який завжди забезпечує правильний результат;

– неповнота, неточність та суперечливість знань;

– відсутність чітких однозначних критеріїв вибору оптимального рішення;

– велика розмірність простору рішень.

Інтелектуальні задачі часто називають *неформалізованими задачами*, багато з них які розв'язуються інтелектуальними системами.

1.3 Напрями досліджень в галузі штучного інтелекту

В дослідженнях у галузі штучного інтелекту склалося два головних напрями: біонічний і прагматичний.

Біонічний напрям досліджень в галузі штучного інтелекту засновано на припущенні про те, що якщо в штучній системі відтворити структури й процеси людського мозку, то й результати вирішення завдань такою системою будуть подібні до результатів, що отримує людина. В цьому напрямку досліджень виділяються:

– Нейромережні алгоритми. В основі лежать системи елементів, які подібно до нейронів головного мозку здатні відтворювати деякі інтелектуальні функції. Прикладні системи, розроблені на основі цього підходу, називаються нейронними мережами.

– Структурно-евристичний підхід. В його основі лежать знання щодо поведінку спостережувального об'єкта або групи об'єктів і міркування про ті структури, які могли б забезпечити реалізацію спостережуваних форм поведінки. Прикладом подібних систем служать мультиагентні системи.

– Еволюційні алгоритми. В цьому випадку можна вирішити завдання, яке формулюється в термінах, що еволюціонує популяції організмів – сукупності підсистем, що протидіють і співпрацюють, в результаті функціонування яких забезпечується необхідна рівновага (стійкість) всієї системи в умовах постійно змінних впливів середовища. Такого роду підхід реалізовано в прикладних системах на основі генетичних алгоритмів.

– Нечітка логіка. Найбільш разючою в людському інтелекті є здатність приймати правильні рішення в умовах неповної та нечіткої інформації. Побудова моделей «наближених роздумів людини» й використання їх в комп'ютерних системах представляє сьогодні одну з найважливіших проблем

науки. «Штучний інтелект», який легко вирішує завдання управління складними технічними комплексами, часто є безпорадним в простих ситуаціях повсякденного життя. Для створення інтелектуальних систем, здатних адекватно взаємодіяти з людиною, потрібно застосовувати новий математичний апарат, який переводить неоднозначні життєві твердження в мову чітких і формальних математичних формул.

Прагматичний напрям ґрунтується на припущенні про те, що розумова діяльність людини є «чорною скринькою». Але, якщо результат функціонування штучної системи збігається із результатом діяльності експерта, то таку систему можна визнати інтелектуальною незалежно від способів отримання цього результату. При такому підході не ставиться питання про адекватність використаних в комп'ютері структур і методів чи якими користується в аналогічних ситуаціях людина, а розглядається лише кінцевий результат вирішення конкретних завдань.

З погляду кінцевого результату в прагматичному напрямі можна виділити три цільові галузі:

розробка методів подання й обробки знань – є однією з основ сучасного періоду розвитку штучного інтелекту;

інтелектуальне програмування – розбивається на кілька груп. До них відносять ігрові програми, природномовні програми (системи машинного перекладу, автоматичного реферування, генерації текстів), розпізнавальні програми, програми створення творів живопису та графіки;

створення інструментарію. Інструментарій – мови для систем штучного інтелекту; дедуктивні та індуктивні методи автоматичного синтезу програм; лінгвістичні процесори; системи аналізу та синтезу мови; бази знань; оболонки, прототипи систем; системи когнітивної графіки;

Спільним для перелічених програм є широке використання пошукових процедур і методів вирішення завдань, пов'язаних з пошуком і переглядом великого числа варіантів. Ці методи застосовуються при машинному рішенні ігрових завдань, в задачах вибору рішень, при плануванні доцільної діяльності в інтелектуальних системах [2].

1.4 Суть реалізації ШІ в теорії і на практиці

Суть реалізації мислення досі до кінця не з'ясована і залишається таємницею для науки. Сьогодні комп'ютери обробляють здебільшого не саму інформацію, а лише вміст комірок пам'яті, які можна заповнити чим завгодно. Отже, комп'ютери не «осмислюють» вміст інформації на відміну від людей, для яких характерним є виключно осмислені поняття. Образно можна сказати, що в людей процес мислення відбувається в душі, в той час, як для машин її не існує.

З яких компонентів зазвичай будується система штучного інтелекту, та й будь-якого інтелекту взагалі?

У першу чергу ШІ – це сукупність «апаратного заліза» та відповідного програмного забезпечення. Першим зазвичай виступає комп'ютер певної

конфігурації й механізми, що обслуговують (маніпулятори, відеокамери, звукові та інші датчики). Значною мірою на «інтелектуальність» машини в цілому впливає програмна начинка, яка визначає ступінь «просунутості» даного ШІ.

В електронній начинці ШІ **в першу чергу** присутня величезна кількість пам'яті, на основі якої й будуються всі міркування та висновки. Зрозуміло, що всі знання з різних галузей в пам'ять ШІ закласти неможливо, але зробити інтелектуальну систему в певній галузі знання цілком можливо. Зазвичай, людина спочатку закладає в систему мінімальні знання про світ. Далі ці знання розширюються в процесі накопичення досвіду і вкладення його людиною (пасивний шлях) або самою системою (активний шлях) в результаті її адаптації до умов навколишнього середовища. Однак комп'ютерна пам'ять являє собою лише просту сукупність файлів і тек.

Пам'ять людини влаштовано набагато складніше – вона оперує не файлами, що є клаптиками інформації. Людська пам'ять – пам'ять образів. Людську пам'ять можна порівняти з кометою: позаду – довгий «хвіст» життєвого досвіду, який з часом автоматично забувається і затирається новим; сама комета – це шар реальної щосекундної пам'яті; тонкий перший шар – це туманні міркування (передбачення) людського майбутнього. І поки що пам'ять систем ШІ докорінно відрізняється від людської.

В другу чергу, сам логічний процес обчислення «ситуації» відбувається в пристрої обробки інформації. Найчастіше це певне програмне забезпечення та центральний процесор комп'ютера. Від можливостей цього центру обробки інформації безпосередньо залежить продуктивність і активність ШІ.

Найголовнішою відмінністю програмного забезпечення справжнього штучного інтелекту від простих додатків є можливість «мислити» образами. За допомогою образного мислення сьогодні стали доступними такі технології, як стиснення і кодування інформації, обробка біометричних образів, оптимізація гами передачі кольору, подібний пошук, аналіз сенсу зображень, автоматична каталогізація інформації, алгоритми розпізнавання та класифікації образів.

Для людини прикладами образів можуть бути небо, хмари, музика, море, вірші тощо. Здатність сприйняття зовнішнього світу у формі образів дозволяє людям дізнаватися нескінченно велику кількість об'єктів і розуміти один одного незалежно від національної приналежності.

Процес сприйняття об'єкта як образа для машини має деякі особливості. Зазвичай, перед виділенням образу (наприклад, графічного) заздалегідь вважається відомим лише те, що потрібно розділити множину точок деякого простору на два або більше образів, і що після поділу всі точки будуть належати до цих двох (або більше) образів. При цьому, заздалегідь відомо лише розташування точок вихідної галузі (їх приблизні координати). Далі, відбувається сам процес поділу точок на образи за певними критеріями (для зображення це буде зміна кольорів і контрастів). Іноді потрібно обробити зображення так, щоб точки були більш явними для розділення (наприклад,

перевести кольорове зображення в чорно-біле) – це зробить чутливість поділу вищою (так працює більшість програм для розпізнавання тексту).

Якщо система зможе самостійно класифікувати й фільтрувати не лише раніше відомі об’єкти, але і невідомі (не знаючи їх властивостей, за зовнішнім виглядом), то цей процес буде називатися самонавчанням. Сьогодні системи ШІ можуть розрізняти тільки нечисленні образи в невеликих заданих просторах.

Важливою особливістю ШІ має стати його навчання і над цією проблемою працюють численні вчені в усьому світі. Навчання, зазвичай, визначається як процес, в результаті якого система поступово набуває здатність відповідати потрібними реакціями на певні зовнішні впливи. На даний час існують прототипи обладнання, що здатні навчатися найпростішим механічним операціям (обробка деталей на верстаті, копіювання людської ходи). Однак, досягнення у сфері навчання ШІ поки просуваються досить низькими темпами і не встигають за розвитком електроніки.

Для вирішення тієї чи іншої задачі ШІ сьогодні необхідний алгоритм рішення (втім, як і будь-якій людині). Алгоритм – це точне розпорядження про виконання в певному порядку операцій для розв’язання певної задачі. Знаходження алгоритму для людини або машини пов’язано з тонкими й складними міркуваннями. Ці міркування часто вимагають винахідливості й творчого підходу, тому, машина постійно потребує взаємодії з людиною через брак вищевказаних якостей. Машині не властивий «метод тику» – вона лише шукає варіанти розв’язання проблеми за допомогою прописаних в базі даних відомостей.

Важливу роль у функціонуванні ШІ виконують функції аналізу інформації та накопичення життєвого досвіду. Спостерігаючи за дітьми, ми переконуємося, що більшу частину знань вони отримують шляхом навчання і спілкування з навколишнім світом, за допомогою якостей, що закладені в них заздалегідь. Винахід ефективного механізму самоаналізу та самостійного накопичення життєвого досвіду поставить ШІ на значно вищий рівень порівняно з сучасним.

1.5 Машинне навчання

Галузь машинного навчання виникла з питання: чи може комп’ютер вийти за рамки того, «що ми й самі знаємо, як виконувати», і самостійно навчитися розв’язувати певну задачу? Чи може комп’ютер здивувати нас? Чи може комп’ютер без допомоги програміста, що задає правила обробки даних, автоматично визначити ці правила, досліджуючи дані?

Ці питання відкривають двері в нову парадигму програмування. У класичному програмуванні, в парадигмі символічного ШІ, люди вводять правила (програму), дані для обробки відповідно до цих правил і отримують відповіді (рис.1.3). У машинному навчанні люди вводять дані й відповіді, які

відповідають цим даним, а на виході отримують правила. Ці правила потім можна застосувати до нових даних для отримання оригінальних відповідей.

Рисунок 1.3 – Принцип роботи машинного навчання та класичного програмування

У машинному навчанні система навчається, а не програмується явно. Їй передаються численні приклади, які стосуються розв’язуваної задачі, а вона знаходить в цих прикладах статистичну структуру, яка дозволяє системі виробити правила для автоматичного виконання задач. Наприклад, щоб автоматизувати задачу визначення фотографій, зроблених у відпустці, можна передати системі машинного навчання безліч прикладів фотографій, вже класифікованих людьми, і система вивчить статистичні правила класифікації конкретних фотографій.

Розквіт машинного навчання почався тільки в 1990-х, але ця галузь швидко перетворилася в найбільш популярний і успішний розділ ШІ, і ця тенденція була підкріплена появою більш швидкодійної апаратури й величезних наборів даних. Машинне навчання тісно пов’язане з математичною статистикою, але має кілька важливих відмінностей. На відміну від статистики, машинне навчання зазвичай має справу з великими й складними наборами даних (наприклад, що складаються з мільйонів фотографій, кожна з яких складається з десятків тисяч пікселів), до яких практично неможливо застосувати класичні методи статистичного аналізу, такі як «байєсівські методи» [3].

Як результат, машинне й особливо глибоке навчання, не мають потужної математичної платформи й ґрунтуються майже виключно на інженерних рішеннях. Це практична дисципліна, в якій ідеї частіше доводяться емпірично, а не теоретично.

Існує багато алгоритмів машинного навчання. Один з найпоширеніших – алгоритми класу С4. Ці алгоритми дозволяють вибудовувати та аналізувати складне дерево рішень. З кожною гілкою дерева асоціюється певний клас прикладів розв’язання проблеми. В процесі вирішення класи можуть розбиватися на підкласи. Завершенням роботи алгоритму є прийняття того чи

іншого рішення, що задовольняє потребам задач. Недоліком алгоритму є обмеженість прикладів розв'язання проблеми.

В останні роки поширення набувають технології Data Mining (видобуток знань) та KnowLedge Discovery (пошук закономірностей у представлених даних).

Інтелектуальний аналіз даних і обробка статистичної інформації. Порівняно новий напрямок застосування ШІ. Сюди відносять процес виявлення закономірностей у вихідній інформації, побудова певної моделі для аналізу інформації, прогнозування результатів дослідження на майбутнє і подання у вигляді графічної інформації. Це доволі перспективний напрям ШІ, який вже реально застосовується на різних біржах і в маркетинговій діяльності.

Розробка природномовних інтерфейсів та систем машинного перекладу. Комп'ютерна лінгвістика, зокрема машинний переклад є популярною темою ще з 50-х років. Ідея перекладу не є такою простою, як здавалося першим розробникам. Вони застосовували послідовний приклад слів у тексті, що було недоречним, бо перекласти текст можна лише базуючись на розумінні всього тексту і в контексті всієї інформації.

1.6 Глибинне навчання

Щоб дати визначення *глибинному навчанню* і зрозуміти різницю між глибоким навчанням та іншими методами машинного навчання, спочатку потрібно отримати деяке уявлення про те, що *роблять* алгоритми машинного навчання. Вище зазначалося, що машинне навчання виявляє правила розв'язання задач обробки даних за прикладами очікуваних результатів. Тобто для машинного навчання потрібні три складові:

– *Контрольні вхідні дані* – наприклад, якщо вирішується завдання розпізнавання мови, такими контрольними вхідними даними можуть бути файли із записом мови різних людей; якщо вирішується завдання класифікації зображень, такими даними можуть бути зображення.

– *Приклади очікуваних результатів* – в задачі розпізнавання мови це можуть бути транскрипції звукових файлів, складені людьми; в задачі класифікації зображень очікуваним результатом можуть бути теги, такі як «собака», «кішка» і ін.

– *Спосіб оцінки якості роботи алгоритму* – це необхідно для визначення, наскільки далеко відхиляються результати, які повертаються алгоритмом, від очікуваних. Оцінка використовується як сигнал зворотного зв'язку для коригування роботи алгоритму. Цей етап коригування ми й називають навчанням.

Модель машинного навчання трансформує вихідні дані в значущі результати, «навчаючись» на відомих прикладах вхідних даних і результатів. Тобто головним завданням машинного і глибокого навчання є *значуще перетворення даних*, або, іншими словами, навчання *поданням* вхідних даних, наближає нас до очікуваного результату.

🔔 *Дослідницьке завдання:*

1) Дослідити методи підвищення інтерпретованості глибинних моделей у системах штучного інтелекту, які використовуються в критично важливих галузях, таких як медицина, фінанси або автономні транспортні засоби.

2) Оцінити, як підвищення інтерпретованості впливає на довіру користувачів до системи прийняття рішень.

? *Контрольні питання:*

1) Надати визначення поняттям: штучний інтелект (ШІ). Основні цілі ШІ?

2) Пояснити різницю між вузьким (weak) та загальним (strong) штучним інтелектом.

3) Якими є основні напрямки досліджень в галузі ШІ?

4) Які переваги глибокого навчання порівняно з іншими методами машинного навчання?

Тема 2. Основи нейронних мереж. Нейронні мережі. Представлення даних для нейронних мереж. Механізми нейронних мереж

2.1 Основи нейронних мереж

Принцип створення штучних нейронних мереж запозичено з біології. Вони утворені з елементів, які відтворюють елементарні функції біологічного нейрона. Штучні нейронні мережі відтворюють певні властивості, які притаманні мозку людини. Вони навчаються на основі досвіду, узагальнюють свій досвід, здатні виділяти головне з інформації, що надходить.

Здатність нейронної мережі до навчання вперше була досліджена Дж. Маккалоком і У. Піттом в дослідах 1943 року на створеній ними моделі *нейрона.* Автори описали принципи побудови нейронних мереж. Пізніше, в 1962 році, Ф. Розенблат запропонував свою модель нейронної мережі – перцептрон, а в 1986 р. Дж. Хінтон і його колеги опублікували статтю з описом моделі нейронної мережі й алгоритмом її навчання, що дало поштовх до ефективного вивчення нейронних мереж [6].

Для моделей, побудованих за типом нейронних мереж людського мозку, характерно легке розпаралелювання алгоритмів і висока продуктивність. З людським мозком їх зближує важлива властивість, яка відсутня в простих електронних машинах: нейронні мережі працюють навіть за умови неповної інформації про навколишнє середовище, тобто, як і людина, вони можуть відповідати не тільки «так» або «ні», але і «не знаю точно, але скоріше так».

Нейронним мережам сьогодні під силу розпізнавання сигналів, мови, зображень, пошук даних, фінансове прогнозування, шифрування даних. Нейромережний підхід використовується у великій кількості завдань – для кластеризації інформації з Інтернету, для імітації та моделювання складно влаштованого людського мозку, для розпізнавання образів і ін. Зараз продовжується вдосконалення методів синхронної роботи нейронних мереж на паралельних пристроях.

До переваг нейронних мереж можна віднести самонавчання, самоналаштування, гнучкість конфігурації, високу ефективність. Серед найбільш відомих сьогодні нейронних мереж виділяють мережі Хопфілда, нейронні мережі зі зворотним поширенням похибки й самоорганізовані карти.

2.2 Нові архітектури комп’ютерів

Сучасні комп’ютери, як і комп’ютери I покоління базуються на традиційній послідовній архітектурі фон Неймана, яка є доволі неефективною для символьної обробки. Тому, зусилля науковців та розробників скеровані на розробку архітектур, що здатні обробляти символьні та логічні дані. Створюються ПРОЛОГ та LISP машини, комп’ютери баз даних, паралельні та векторні комп’ютери.

Хоча існують гарні промислові зразки, але висока вартість, недостатнє програмне оснащення й апаратна несумісність з традиційними комп'ютерами відчутно гальмують широке використання нових архітектур.

Одна з найцікавіших і корисних сторін застосування ШІ – розробка ігор, розважальних програм і систем штучного спілкування з людиною. Велику частку тут займає моделювання соціальної поведінки, спілкування, людських емоцій, творчості. Це одне з найскладніших напрямів розробки ШІ й водночас – один з найперспективніших.

Сучасні системи штучного інтелекту здатні освоїти набагато більше спеціальностей, ніж проста людина, завдяки значному числу різноманітних передавачів інформації та пристроїв, які створюють подібно до будови органів чуття людини.

Розробки ШІ застосовується сьогодні у вигляді автономних секретарів, пошукових машин, планувальників робіт, професійних вчителів, продавців. Також передбачається використання надалі систем ШІ у всіляких побутових приладах: прибиральниках приміщень; агрегатах для приготування, доставлення та замовлення їжі; автоматичних водіях автомобілів тощо.

Однак не слід думати, що комп'ютери або роботи зможуть вирішувати будь-які завдання. Вченими доведено існування таких типів завдань, для вирішення яких є неможливим єдиний ефективний алгоритм (наприклад, складні життєві ситуації). Людина часто методом «наукового тику» розширює для себе зону пізнання про природу, відкриває нові закони. Комп'ютерному штучному інтелекту це абсолютно не властиво.

Сьогодні ми маємо можливість спостерігати постійне зростання обчислювальної потужності комп'ютерів, але це не означає появи в них ШІ. На жаль, навіть принципи роботи людської психіки сьогодні залишаються неясними. А оскільки ШІ спочатку замислювався як прообраз людини, то його створення пов'язане з невідомістю. Однак зростання продуктивності комп'ютерів у поєднанні з підвищенням якості алгоритмів обробки робить можливим застосування різних наукових методів на практиці в різних аспектах життя людства.

Розглянемо основні проблеми, пов'язані з розробкою ШІ на практиці.

Більшість сучасних розробок ШІ використовують кілька типів понять: ТАК (добре) і НІ (погано). В математиці й електроніці це нормально, але в житті точні поняття використовують рідко. Оскільки спочатку ШІ «замислювався» як людиноподібний інтелект, що слугує доповненням до людини, то догодити самій людині буде дуже нелегко. Як, наприклад, машині зрозуміти депресивний стан або ейфорію людини? Поняття «веселий» і «сумний» для машини тут ніяк не підходять.

Проблеми в розробці ШІ простежуються і на рівні формування образів і образної пам'яті. Оскільки образи в мисленні людини взаємопроникають один в одного, то формування образних ланцюжків у людей не представляє складності – воно асоціативно. Файли ж, на противагу до образів, є відокремленими пакетами машинної пам'яті. В пам'яті людини

пошук даних ведеться не за вмістом пам’яті, а вздовж готових ланцюжків асоціативних зв’язків. Комп’ютер же шукає тільки конкретні файли.

Приклад: для людини не буде проблемою впізнати обличчя друга на фотографії, навіть якщо він схудне або одужає, і це є яскравим прикладом асоціативної пам’яті. Для машини це практично неможливо. Вона не зможе відрізнити головне від другорядного. Для отримання результату ШІ використовує тільки певну базу відомих даних. Йому невластивий експеримент.

Проблема перекладу з однієї мови на іншу, а також навчання машини мові. Якщо запропонувати сучасним програмам-перекладачам (наприклад, Promt) перевести будь-який абзац з книги іншою мовою, то зрозуміло, що якості немає. В результаті отримаємо простий набір слів. Чому? Тому, що для перекладу цілих речень необхідно розуміти сенс речення, а не просто перекладати слова. Сучасні ШІ – програми не можуть поки виділяти сенс у тексті (ймовірно, тому, що посередником для перекладу, скажімо, з англійської на українську, є бездушна машинна мова – мова одиниць і нулів).

Простота математичних обчислень. Останнім часом багатьма провідними фахівцями в галузі ШІ внесено пропозицію щодо виключення зі списку високоінтелектуальних завдань простого алгебраїчного розв’язання рівнянь, оскільки для цього сьогодні є стандартні послідовні алгоритми обчислень. Це не вимагає складних, багатоетапних і часто непослідовних інтелектуальних здібностей. Розпізнавання тексту, гра в шахи та шашки, розпізнавання звуків на сьогодні успішно застосовуються на практиці.

Сучасні розробки, пов’язані зі штучним інтелектом, нездатні до самокопіювання (розмноження). На сучасному етапі розвитку кібернетики та електроніки абсолютно самостійне само копіювання роботів є неможливим, необхідно хоча б часткове (часто значне) втручання людини. Однак для програм цей процес є простим, наприклад, можливості утиліт самостійно копіюватися в іншу директорію. Яскравим прикладом є комп’ютерні та мобільні віруси, які здатні до безконтрольного розмноження і виконання руйнівних дій.

Ще одна проблема на шляху до створення ШІ – відсутність в нього всякого прояву волі. Як це не дивно звучить, але в сучасних комп’ютерів є колосальні можливості до складних розрахунків, але абсолютно відсутні будь-які бажання. Навіть якщо комп’ютер забезпечити мікрофоном і акустикою, це не означає, що він почне самостійно писати музику або мимовільно запускати будь-які додатки. Він не ледачий – просто у нього немає бажань. Комп’ютеру все одно, хто з ним працює, навіщо і з якою метою.

В сучасних прототипах ШІ відсутні стимули до подальшого вдосконалення. В природі на будь-який живий організм діє фактор природного відбору, який породжує постійне пристосування до умов навколишнього середовища. Голод, прагнення вижити й дати потомство – це фактори, що постійно діють на живий організм, як стимул до подальшого вдосконалення.

Мотивація більшості сучасних ШІ є дуже примітивною: людина задала задачу – машина її виконує без варіантів і емоцій. Теоретично на мотивацію і вдосконалення може вплинути введення зворотних зв'язків комп'ютер -> людина і створення покращеної системи самонавчання машини. Правда, це тільки теорія – на практиці ж все виявляється набагато складніше. Однак подібна робота вже проводиться. Як стимул, вибрано елементарне почуття голоду – провісник швидкого закінчення енергетичних ресурсів і, відповідно, існування машини. Американець С. Вілкінсон створив «гастроробота» на ім'я «Жуй – жуй». Машина харчується цукром, і основою її поведінки є дослідження навколишнього світу в пошуках їстівного. Тіло «Жуй – жую» складається з трьох візків, а відчуття голоду є його постійним супутником, оскільки акумулятори постійно вимагають перезаряджання. Проблемою є часті помилки машини у виборі харчових продуктів.

Деяка примітивність штучних нейронних мереж. Штучні нейронні мережі демонструють сьогодні дивовижні переваги, що властиві людському мозку. Вони навчаються на основі особистого досвіду, узагальнюють інформацію, самоконфігуруються, витягують головне з інформації з зайвими даними. Однак навіть найрозвиненіші штучні мережі не можуть дублювати функції людського мозку. Реальний інтелект, що демонструється складно влаштованими нейронними мережами, знаходиться нижче рівня розвитку інтелекту дощового хробака.

Неефективність штучного інтелекту у військових цілях. Останнім часом у ЗМІ досить часто з'являються новини про створення ШІ у військових цілях. Проте в реальності перед розробниками подібних машин-роботів стоять дуже складні й часто нерозв'язні завдання. Перш за все це недоліки систем автоматичного розпізнавання, нездатних самонавчатися й адекватно аналізувати інформацію в режимі реального часу (приймати потрібні рішення в потрібну хвилину). Таке бойовій машині дуже важко, а швидше за все – практично неможливо, буде відрізнити на полі бою своїх від чужих.

Також поки не розроблено алгоритмів роботи подібних пристроїв в умовах незнайомої місцевості. Подібні бойові одиниці здатні максимум до простого дистанційного керування. Більш видатні результати досягнуто військовими в прикладних напрямках: точне розпізнавання мови й тембру голосу, різноманітні «детектори брехні», створення консультаційних систем (зниження однотипних дій і навантаження на пілотів в режимі реального польоту), системи низькорівневого аналізу зображення, отриманого від відеокамери, тощо.

Крім цього, сьогодні створено досить велику кількість приладів з подобою ШІ, покликаних вдосконалити роботу збройних сил: різноманітні інтелектуальні сонари й радари для виявлення цілей, супутникова система позиціювання для точного координування локалізації військ та їх пересування, різноманітні системи навігації в судноплавстві.

Засновник SpaceX і Tesla Ілон Маск, який також цікавиться розробкою ШІ, побоюється, що штучний інтелект може стати фундаментальною загрозою

для людства. Тому дослідження в цій галузі потрібно обмежити й перевести під державний контроль. При цьому, за словами Маска, якщо зв'язати людський інтелект з машинним, це дозволить збалансувати ситуацію.

В 2019 році країни «великої двадцятки» домовилися про принципи поводження з ШІ.

Зокрема, у спільній заяві йшлося, що «розробники й користувачі технологій ШІ повинні поважати основні юридичні принципи, права людини та демократичні цінності… і для підвищення довіри до технологій ШІ та повної реалізації їх потенціалу необхідно, щоб у центрі використання штучного інтелекту стояла людина».

Крім того, один з принципів свідчить, що системи ШІ повинні бути «стійкі, захищені й надійні» протягом усього періоду їх використання і не повинні нести з собою «жодних неприйнятних ризиків».

Отже, сьогодні продовжується впровадження логіки в прикладні галузі та програми. Програм глобального масштабу, здатних хоч якоюсь мірою відповідати реальній людині, вести процес розумного мислення і спілкування, поки немає, і в найближчому часі не передбачається (існує занадто багато перешкод і нерозв'язних проблем).

Комп'ютер виконує тільки точні вказівки, які йому надає людина. При написанні будь-якого додатку програміст користується мовою високого рівня, потім програма – транслятор перекладає цей додаток на машинну мову директив, яку і розуміє процесор комп'ютера. Тому, стає зрозуміло, що сам по собі комп'ютер до мислення нездатний в принципі, але високорівневі програми роблять його відносно інтелектуальним.

Роблячи висновок зі всього сказаного, можна сказати, що високоінтелектуальне мислення – це властивість не високоорганізованої матерії, а властивість високоорганізованої ДУШІ. Тварини й людина здатні ставити й вирішувати завдання. Комп'ютери – пристрої неживі, сьогодні їх олюднюють програмісти, а машини лише слідують їх вказівками. На жаль, якою б не була складною сучасна програма, які б складні алгоритми не було в неї закладено, в кінцевому підсумку вона зможе зробити лише те, що передбачено її автором.

Вчені намагаються відкрити завісу віддаленого майбутнього. Чи можливе створення штучного інтелекту? Чи можна створити такі людиноподібні системи, які зможуть мислити абстрактними образами, будуть саморозмножуватися, самонавчатися, коректно реагувати на зміни навколишнього середовища, володіти почуттями, волею, бажаннями? Чи можна створити відповідні алгоритми? Чи зможе людство контролювати такі об'єкти? На жаль, відповідей на ці питання поки немає. Залишається сподіватися на те, що, якщо штучний інтелект можна створити в принципі, то рано чи пізно він буде створений.

🔔 *Дослідницьке завдання:*

1) Провести експерименти з різними типами даних, такими як текстові, числові та зображення, та оцінити, який підхід до представлення даних забезпечує найкращу точність і швидкість навчання.
2) Розглянути, як різні методи підготовки та представлення даних (наприклад, нормалізація, стандартизація) впливають на продуктивність нейронних мереж.

? *Контрольні питання:*

1) Надати визначення: штучний нейрон. Як штучний нейрон використовується в нейронних мережах?
2) Які основні компоненти складають нейронну мережу?
3) Як відбувається передача і обробка інформації в нейронній мережі?
4) Які проблеми розвитку ШІ?

ЗМІСТОВИЙ МОДУЛЬ 2. ПОЧАТОК РОБОТИ З НЕЙРОННИМИ МЕРЕЖАМИ

Тема 3. Початок роботи з нейронними мережами

3.1 Структура нейронної мережі

В останні роки, разом з нечіткими системами, великий інтерес викликає проблематика нейронних мереж і генетичних алгоритмів. Ці напрямки належать до наукової галузі, обумовленої в англомовній літературі терміном Computational Intelligence. На рис.3.1. видно, що задачі нейронних мереж, генетичних алгоритмів і нечітких систем можуть розглядатися без зв'язку між собою, однак їхня взаємозалежність виявляється надзвичайно важливою. Зокрема, генетичні алгоритми можна застосовувати для підбора ваг і топології нейронної мережі, а також для формування бази правил і функцій приналежності нечіткої системи

Рисунок 3.1 – Взаємозв'язки між нейронними мережами, генетичними алгоритмами й нечіткими системами

Нейронні мережі дозволяють вибирати відповідні параметри для самих генетичних алгоритмів (параметри схрещування і мутації); саму філософію нейронних мереж можна закласти у фундамент нечітких систем, що у результаті знаходять здатність до навчання.

Крім того, методи теорії нечітких множин дозволяють підбирати як згадані вище параметри генетичних алгоритмів, так і коефіцієнти, що визначають швидкість навчання нейронних мереж. Одним з популярних напрямків Artificial Intelligence є теорія нейронних мереж (neuron nets).

Людей завжди цікавило їхнє власне мислення. Це самопитання, мислення мозку самого про себе є, можливо, відмітною рисою людини. Нейробіологи й нейроанатоми досягли в цій галузі значного прогресу. Ретельно вивчаючи структуру і функції нервової системи людини, вони багато чого зрозуміли в «електропроводці» мозку, але мало довідалися про його функціонування. У процесі нагромадження ними знань з'ясувалося, що мозок має приголомшливу складність. Сотні мільярдів нейронів, кожний з яких з'єднаний із сотнями або тисячами інших, утворюють систему, що далеко перевершує наші самі сміливі мрії про суперкомп'ютери. На сьогодні існують дві мети нейронного моделювання: перша – зрозуміти функціонування нервової системи людини на рівні фізіології та психології, й друга – створити обчислювальні системи (штучні нейронні мережі), що виконують функції, подібні до функцій мозку.

Штучні нейронні мережі є моделями нейронної структури мозку, який здатен сприймати, обробляти, зберігати та продукувати інформацію. Особливістю мозку також є навчання та самонавчання на власному досвіді. Адаптивні системи на основі штучних нейронних мереж дозволяють з успіхом розв'язувати проблеми розпізнавання образів, виконання прогнозів, оптимізації, асоціативної пам'яті й керування.

Механізм природного мислення базується на збереженні інформації у вигляді образів. Штучні нейронні мережі дозволяють створення паралельних мереж, їх навчання та вирішення інтелектуальних завдань, не використовуючи традиційного програмування. В лексиконі розробників та користувачів нейромереж присутні слова «поводити себе», «реагувати», «самоорганізовувати», «навчати», «узагальнювати» та «забувати».

Штучний нейрон є базовим модулем нейронних мереж. Він моделює основні функції природного нейрона (рис. 3.2).

При функціонуванні нейрон одночасно отримує багато вхідних сигналів. Кожен вхід має свою власну синаптичну вагу, яка надає входу вплив, необхідний для функції суматора елемента обробки. Ваги є мірою сили вхідних зв'язків і моделюють різноманітні синаптичні сили біологічних нейронів. Ваги суттєвого входу підсилюються і, навпаки, вага несуттєвого входу примусово зменшується, що визначає інтенсивність вхідного сигналу. Ваги можуть змінюватись відповідно до навчальних прикладів, топології мережі та навчальних правил.

Вхідні сигнали x_n зважені ваговими коефіцієнтами з'єднання w_n додаються, проходять через передатну функцію, генерують результат і виводяться.

В програмних реалізаціях штучні нейрони називають «елементами обробки» або «процесорами» і вкладають в них більше можливостей, ніж в базовому штучному нейроні, що описаний вище.

На рис. 3.3 зображено детальну схему штучного нейрона.

Функція суматора може бути складнішою, наприклад, вибір мінімуму, максимуму, середнього арифметичного, добутку або обчислення за іншим

алгоритмом. Багато програмних реалізацій використовують власні функції суматора, що запрограмовані на мові високого рівня (C, C++) [7].

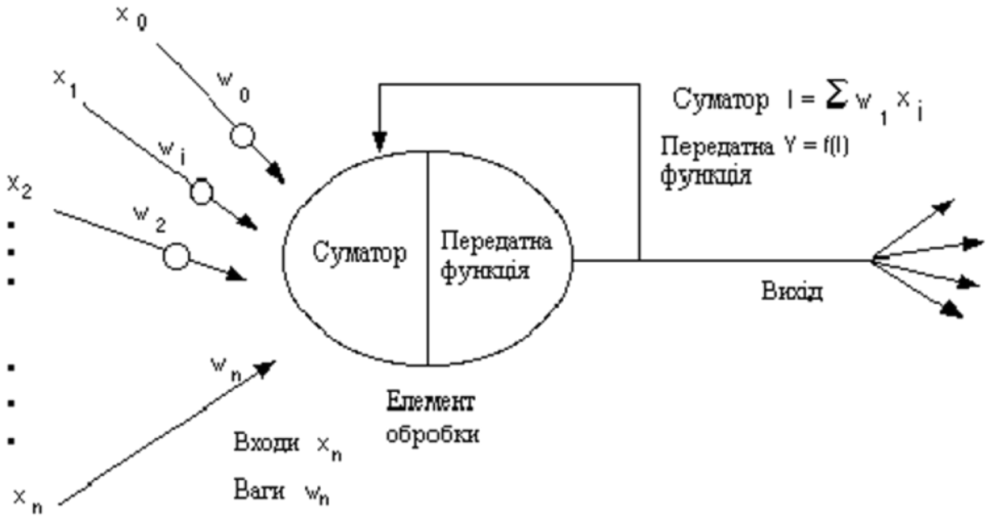


Рисунок 3.2 – Базовий штучний нейрон

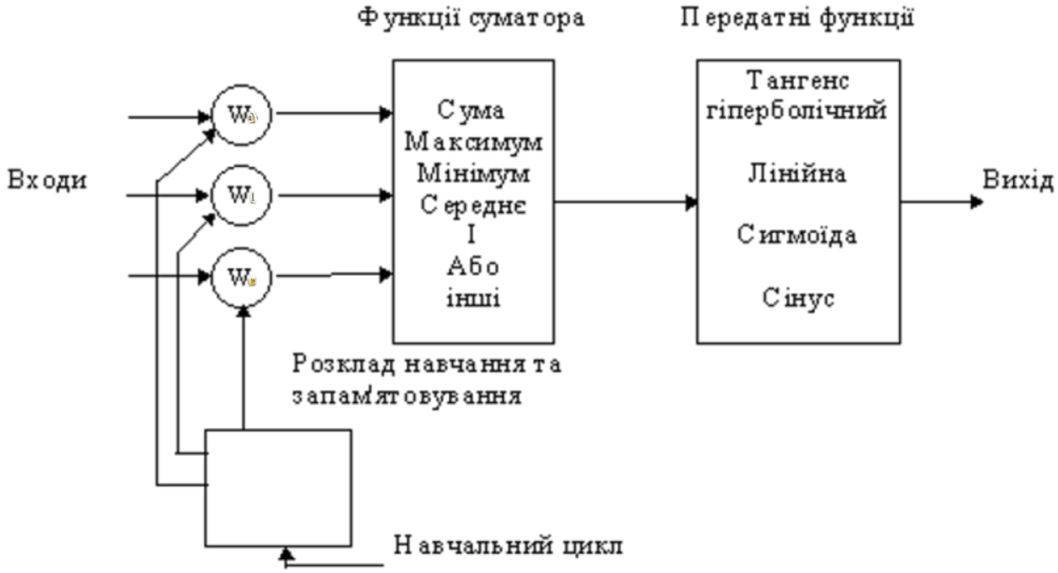


Рисунок 3.3 – Модель «елементу обробки»

Перед надходженням до передатної функції вхідні сигнали та вагові коефіцієнти можуть комбінуватись багатьма способами. Алгоритми для комбінування входів нейронів визначають відповідно до мережної архітектури та парадигми.

В деяких нейромережах суматор виконує додаткову обробку, так звану функцію активації, відбувається зміщення, вихід функції суматора в часі. Цю функцію найкраще використовувати як компоненту мережі в цілому, ніж як компоненту окремого нейрона. Часто, ця функція є відсутньою.

Результат функції суматора перетворюється у вихідний сигнал через передатну функцію. В передатній функції для визначення виходу нейрона загальна сума порівнюється з деяким порогом (зазвичай, це діапазон [0, 1] або [-1,1] або інше) за допомогою певного алгоритму.

Переважно застосовують нелінійну передатну функцію, оскільки лінійні (прямолінійні) функції є обмеженими і вихід є пропорційним до входу. Застосування лінійних передатних функцій було проблемою у ранніх моделях мереж, і їх обмеженість та недоцільність була доведена в книзі Мінскі та Пейперта «Перцептрони».

В нейромережах, що існують, в якості, передатної функції використовують сигмоїду, синус, гіперболічний тангенс тощо. На рис.3.4 зображені типові передатні функції.

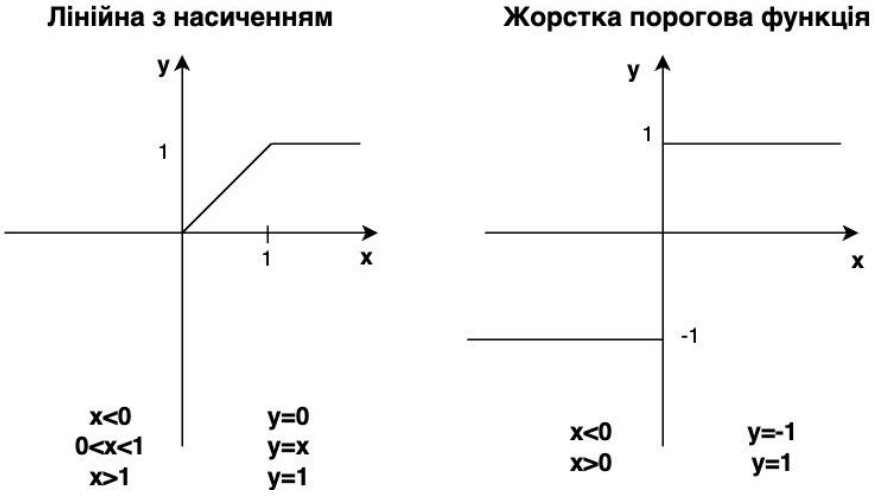


Рисунок 3.4 – Передатні функції

Для простої передатної функції нейромережа може видавати 0 чи 1, 1 чи -1 або інші числові комбінації. Передатна функція в таких випадках є пороговою або «жорстким обмежувачем».

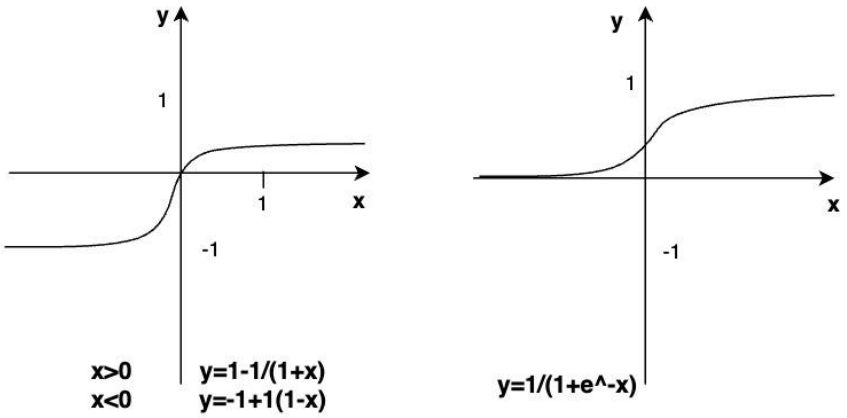


Рисунок 3.5 – Гіперболічний тангенс

Передатна функція, лінійна з насиченням, віддзеркалює вхід всередині заданого діапазону і діє як жорсткий обмежувач поза межами цього діапазону. Це лінійна функція, яка відсікається до мінімальних та максимальних значень, роблячи її нелінійною.

Сигмоїда або S-подібна крива наближує мінімальне та максимальне значення в асимптотах. Вона називається сигмоїдою, коли її діапазон [0, 1], або гіперболічним тангенсом (рис.3.5), при діапазоні [-1, 1]. Важливою рисою сигмоїд є неперервність функцій та їх похідних. Застосування сигмоїдних функцій надає гарні результати й має широке застосування.

Для різних нейромереж можуть вибиратись інші передатні функції. Після обробки сигналу, нейрон на виході має результат передатної функції, який надходить на входи інших нейронів або до зовнішнього з'єднання, як це передбачається структурою нейромережі [8].

3.2 Архітектура з'єднань штучних нейронів

Штучні нейромережі конструюються з базового блоку – **штучного нейрона**. Іншою властивістю нейромереж є величезна кількість зв'язків, які пов'язують окремі нейрони. Групування нейронів у мозку людини забезпечує обробку інформації динамічним, інтерактивним та самоорганізовуваним шляхом.

Біологічні нейронні мережі з мікроскопічних компонентів існують у тривимірному просторі й здатні до різноманітних з'єднань. Але для реалізації штучних мереж присутні фізичні обмеження.

Об'єднуючись у мережі, штучні нейрони утворюють систему обробки інформації, яка забезпечує ефективну адаптацію моделі до постійних змін з боку зовнішнього середовища. В процесі функціонування мережі відбувається перетворення вхідного вектора сигналів у вихідний. Конкретний вид перетворення визначається архітектурою нейромережі, характеристиками нейронних елементів, засобами керування та синхронізації інформаційних потоків між нейронами.

Важливим фактором ефективності мережі є встановлення оптимальної кількості нейронів та типів зв'язків між ними.

Для опису нейромереж використовують кілька усталених термінів, які в різних джерелах можуть мати різне трактування, зокрема:

Структура нейромережі – спосіб зв'язків нейронів у нейромережі.

Архітектура нейромережі – структура нейромережі та типи нейронів.

Парадигма нейромережі – спосіб навчання та використання, іноді містить поняття архітектури.

На базі однієї архітектури може бути реалізовано різні парадигми нейромережі й навпаки.

Серед відомих архітектурних рішень виділяють групу **слабозв'язаних** нейронних мереж, у випадку, коли кожний нейрон мережі зв'язаний лише із

сусідніми. В **повнозв'язних нейромережах** входи кожного нейрона зв'язані з виходами всієї решти нейронів.

Самим поширеним варіантом архітектури є багатошарові мережі. Нейрони в цьому випадку об'єднуються у прошарки з єдиним вектором вхідних сигналів. Зовнішній вхідний вектор подається на вхідний прошарок нейронної мережі (рецептори). Виходами нейронної мережі є вихідні сигнали останнього прошарку (ефектори). Окрім вхідного та вихідного прошарків, нейромережа має один або кілька прихованих прошарків нейронів, які не мають контактів із зовнішнім середовищем (рис 3.6.а, 3.6.б).

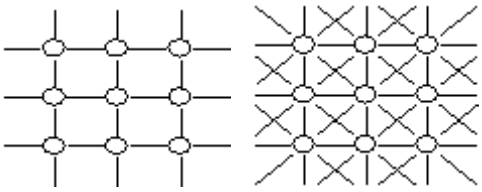


Рисунок 3.6.а – Слабозв'язані нейромережі

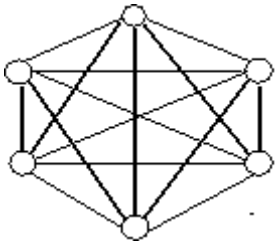


Рисунок 3.6.б – Повнозв'язні нейромережі

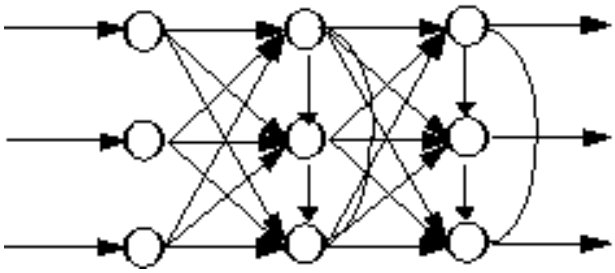


Рисунок 3.7 – Багатошаровий тип з'єднання нейронів

Зв'язки між нейронами різних прошарків називають *проєктивними*.

Зв'язки між нейронами одного прошарку називають *бічними (латеральними)*.

На рис.3.7 показана типова структура штучних нейромереж. Хоча існують мережі, які містять лише один прошарок, або навіть один елемент, більшість застосувань вимагають мережі, які містять як мінімум три типи прошарків – вхідний, прихований та вихідний. Прошарок вхідних нейронів отримує дані або з вхідних файлів, або безпосередньо з електронних датчиків. Вихідний прошарок пересилає інформацію безпосередньо до зовнішнього середовища, до вторинного комп'ютерного процесу, або до інших пристроїв. Між цими двома прошарками може бути багато прихованих прошарків, які містять багато нейронів в різноманітних зв'язаних структурах. Входи та виходи кожного з прихованих нейронів сполучені з іншими нейронами.

Важливим аспектом нейромереж є **напрямок зв'язку** від одного нейрону до іншого:

Зв’язки скеровані від вхідних прошарків до вихідних називаються **аферентними,** а зв’язки у зворотному напрямку називаються **еферентними**.

В більшості мереж кожен нейрон прихованого прошарку отримує сигнали від всіх нейронів попереднього прошарку чи від нейронів вхідного прошарку. Після виконання операцій над сигналами, нейрон передає свій вихід до всіх нейронів наступних прошарків, забезпечуючи передачу вперед (feedforward) на вихід.

При зворотному зв’язку, вихід нейронів прошарку скеровується до нейронів попереднього прошарку (рис.3.8).

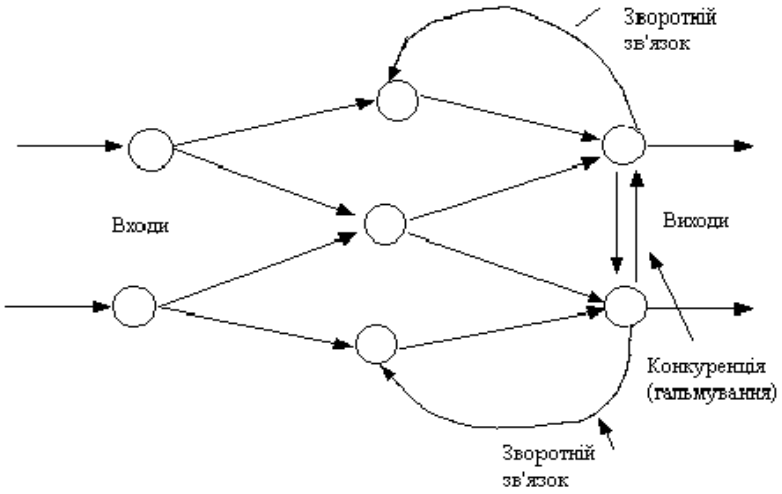


Рисунок 3.8 – Робота зворотного зв’язку нейронів

Напрямок зв’язків нейронів має значний вплив на роботу мережі. Більшість програмних нейромереж дозволяють користувачу додавати, вилучати та керувати з’єднаннями як завгодно. Корегуючи параметри, можна налаштувати зв’язки як на посилення, так і на послаблення величини сигналів.

За архітектурою зв’язків, більшість відомих нейромереж можна згрупувати у два великих класи (рис.3.9).

Мережі прямого поширення (з одно скерованими послідовними зв’язками).

Мережі зворотного поширення (з рекурентними зв’язками).

Типові архітектури нейронних мереж:

Мережі прямого поширення	Рекурентні мережі
Перцептрони	Мережа Хопфілда
Мережа Back Propagation	Мережа Хемінга
Мережа зустрічного поширення	Мережа адаптивної резонансної теорії
Карта Кохонена	Двоскерована асоціативна пам’ять

Мережі ***прямого поширення*** відносять до ***статичних,*** тут на входи нейронів надходять вхідні сигнали, які не залежать від попереднього стану мережі.

Рекурентні мережі вважаються динамічними, оскільки внаслідок зворотних зв’язків (петель) входи нейронів модифікуються в часі, що призводить до зміни станів мережі.

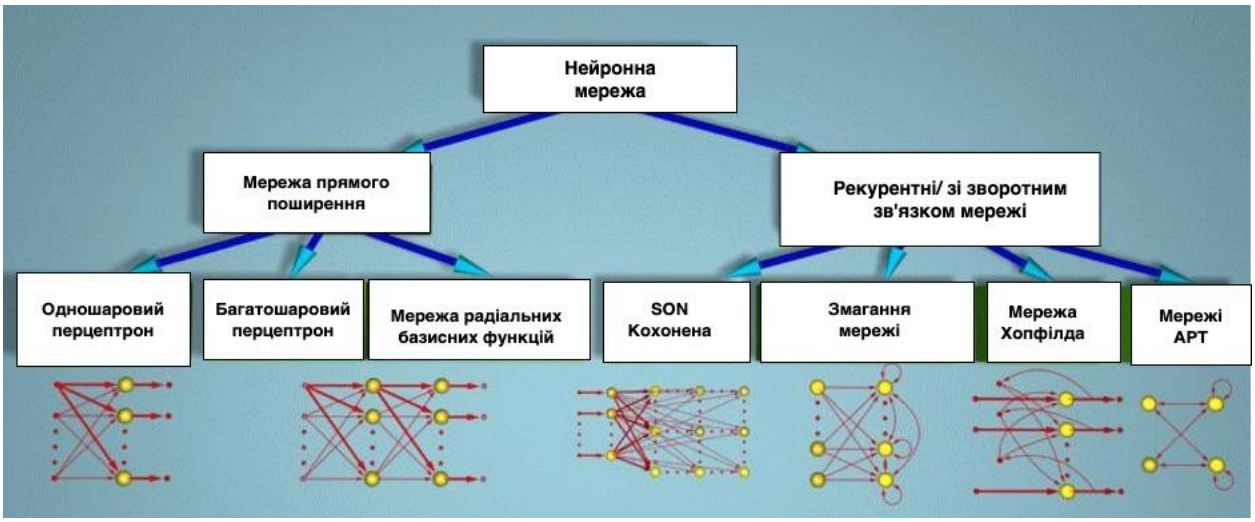


Рисунок 3.9 – Класи нейронних мереж

3.3 Навчання штучної нейронної мережі

Оригінальність нейромереж, як аналога біологічного мозку, полягає у здібності до навчання за прикладами, що складають навчальну множину. Процес навчання нейромереж розглядається як налаштування архітектури та вагових коефіцієнтів синаптичних зв’язків відповідно до даних навчальної множини для ефективного розв’язання поставленої задачі.

Для навчання нейромереж можливо:

Навчання з вчителем (контрольоване навчання)

Навчання без вчителя (неконтрольоване навчання)

Більшість реалізацій нейромереж використовують **контрольоване навчання**, де вихід, що змінюється, постійно порівнюється з бажаним виходом. Вагові коефіцієнти зв’язків на початку встановлюються випадково (ініціалізація мережі), але під час наступних ітерацій корегуються, щоб досягти близької відповідності між бажаним та вихідними результатами. Такі методи навчання націлені на мінімізацію вихідних похибок всіх елементів обробки, що відбувається завдяки неперервній зміні синаптичних ваг до досягнення прийнятної точності мережі.

Перед використанням, нейромережа з контрольованим навчанням повинна бути навченою. Фаза навчання займає певний час. Навчання вважається закінченим при досягненні нейромережею визначеного користувачем рівня ефективності й бажаної статистичної точності. Після навчання вагові коефіцієнти зв’язків фіксуються для подальшого застосування. Деякі типи мереж дозволяють під час використання продовжувати навчання, і це допомагає мережі адаптуватись до змінних умов.

Навчальні множини повинні бути достатньо великими, щоб містити всю необхідну інформацію для виявлення важливих особливостей і зв'язків. Навчальні приклади повинні містити широке різноманіття даних. Якщо мережа навчається лише для одного прикладу, вагові коефіцієнти, що старанно встановлено для цього прикладу, радикально змінюються у навчанні для наступного прикладу. Попередні приклади при навчанні наступних просто забуваються. В результаті система повинна навчатись всього разом, знаходячи найкращі вагові коефіцієнти для загальної множини прикладів.

Наприклад, у навчанні системи розпізнавання піксельних образів для десяти цифр, які представлені двадцятьма прикладами кожної цифри, всі приклади цифри «сім» не доцільно представляти послідовно. Краще надати мережі спочатку один тип представлення всіх цифр, потім другий тип і так далі.

Головною компонентою для успішної роботи мережі є представлення і кодування вхідних і вихідних даних. Штучні мережі працюють лише з числовими вхідними даними, отже, необроблені дані, що надходять із зовнішнього середовища повинні перетворюватись. Важливою є нормалізація даних, тобто приведення всіх значень даних до єдиного діапазону. Нормалізація виконується шляхом ділення кожної компоненти вхідного вектора на довжину вектора, що перетворює вхідний вектор в одиничний. Попередня обробка зовнішніх даних, отриманих за допомогою сенсорів, у машинний формат є спільною і легко доступною для стандартних комп'ютерів [9].

Якщо після контрольованого навчання нейромережа ефективно опрацьовує дані навчальної множини, важливим стає її ефективність при роботі з даними, які не використовувались для навчання. У випадку отримання незадовільних результатів для тестової множини, навчання продовжується. Тестування використовується для забезпечення запам'ятовування не лише даних заданої навчальної множини, але і створення загальних образів, що можуть міститись в даних.

Неконтрольоване навчання може бути великим надбанням у майбутньому. Воно проголошує, що комп'ютери можуть самонавчатись у справжньому роботизованому сенсі. На даний час, неконтрольоване навчання використовується у мережах, як самоорганізовані карти (self organizing maps). Мережі не використовують зовнішні впливи для корегування своїх ваг і внутрішньо контролюють свою ефективність, шукаючи регулярність або тенденції у вхідних сигналах, та здійснюють адаптацію відповідно до навчальної функції. Навіть без повідомлення правильності чи неправильності дій, мережа повинна мати інформацію відносно власної організації, яка закладена у топологію мережі та навчальні правила.

Алгоритм неконтрольованого навчання скеровано на знаходження близькості між групами нейронів, які працюють разом. Якщо зовнішній сигнал активує будь-який вузол в групі нейронів, дія всієї групи в цілому збільшується. Аналогічно, якщо зовнішній сигнал в групі зменшується, це приводить до гальмівного ефекту на всю групу.

Основу для навчання формує конкуренція між нейронами. Навчання нейронів, що конкурують, підсилює відгуки певних груп на певні сигнали. Це пов'язує групи між собою та відгуком. При конкуренції змінюються ваги лише нейрона-переможця.

Оцінка ефективності навчання нейромережі залежить від кількох керованих факторів, важливими з яких є: місткість, складність зразків і обчислювальна складність.

Місткість показує, скільки зразків може запам'ятати мережа, і які межі прийняття рішень можуть бути на ній сформовані.

Складність зразків визначає кількість навчальних прикладів, необхідних для досягнення здатності мережі до узагальнення.

Обчислювальна складність напряму пов'язана з потужністю комп'ютера.

Основні етапи розв'язання задач за допомогою нейромереж:

- Збір даних для навчання;
- Підготовка та нормалізація даних;
- Вибір топології мережі;
- Експериментальний підбір характеристик мережі;
- Експериментальний підбір параметрів навчання;
- Власне навчання;
- Перевірка адекватності навчання;
- Корегування параметрів, остаточне навчання;
- Вербалізація мережі з метою подальшого використання.

Розглянемо докладніше деякі з цих етапів.

Збір даних для навчання

Вибір даних для навчання мережі та їхня обробка є самим складним етапом розв'язання задачі. Набір даних для навчання повинний задовольняти декільком критеріям:

- Репрезентативність – дані повинні ілюструвати дійсне положення речей у предметній галузі;
- Несуперечність – суперечливі дані в навчальній вибірці призведуть до поганої якості навчання мережі;
- Обсяг – як правило, число записів у вибірці повинне на кілька порядків перевершувати кількість зв'язків між нейронами в мережі. В противному випадку мережа просто «запам'ятає» усю навчальну вибірку і не зможе виконати узагальнення.

Підготовка та нормалізація даних

Вихідні дані перетворяться до виду, у якому їх можна подати на входи мережі. Кожен запис у файлі даних називається навчальною парою або навчальним вектором. Навчальний вектор містить по одному значенню на кожен вхід мережі й, залежно від типу навчання (із вчителем або без), по одному значенню для кожного виходу мережі. Навчання мережі на «сирому» наборі, як правило, не дає якісних результатів. Існує ряд способів поліпшити «сприйняття» мережі.

• Нормування виконується, коли на різні входи подаються дані різної розмірності. Наприклад, на перший вхід мережі подаються величини зі значеннями від нуля до одиниці, а на другий – від ста до тисячі. При відсутності нормування, значення на другому вході будуть завжди робити істотно більший вплив на вихід мережі, чим значення на першому вході. При нормуванні розмірності усіх вхідних і вихідних даних зводяться воєдино;

• Квантування виконується над безперервними величинами, для яких виділяється кінцевий набір дискретних значень. Наприклад, квантування використовують для завдання частот звукових сигналів при розпізнаванні мови;

• Фільтрація виконується для «зашумлених» даних.

Крім того, велику роль грає саме представлення як вхідних, так і вихідних даних. Припустимо, мережа навчається розпізнаванням букв на зображеннях і має один числовий вихід – номер букви в алфавіті. У цьому випадку мережа одержить неправильне уявлення про те, що букви з номерами 1 і 2 більш схожі, чим букви з номерами 1 і 3, що, загалом, неправильно. Для того, щоб уникнути такої ситуації, використовують топологію мережі з великим числом виходів, коли кожен вихід має свій зміст. Чим більше виходів у мережі, тим більша відстань між класами й тим складніше їх поплутати.

Вибір топології мережі

Вибирати тип мережі необхідно виходячи з постановки задачі й наявних даних для навчання. Для навчання з учителем потрібна наявність для кожного елемента вибірки «експертної» оцінки. Іноді одержання такої оцінки для великого масиву даних просто неможливо. У цих випадках природним вибором є мережа, що навчається без учителя, наприклад, така як самоорганізована карта Кохонена або нейрона мережа Хопфілда. При розв’язанні інших задач, таких як прогнозування часових рядів, експертна оцінка вже утримується у вихідних даних і може бути виділена при їхній обробці. У цьому випадку можна використовувати багатошаровий перцептрон або мережу Ворда.

Експериментальний підбір характеристик мережі

Після вибору загальної структури потрібно експериментально підібрати параметри мережі. Для мереж, подібних перцептрону, це буде число шарів, число блоків у схованих шарах (для мереж Ворда), наявність або відсутність обхідних з’єднань, передатні функції нейронів. При виборі кількості шарів і нейронів у них, варто виходити з того, що здатності мережі до узагальнення тим вище, чим більше сумарне число зв’язків між нейронами. З іншого боку, число зв’язків обмежене зверху кількістю записів у навчальних даних. Експериментальний підбір параметрів навчання. Після вибору конкретної топології, необхідно вибрати параметри навчання нейронної мережі. Цей етап особливо важливий для мереж, які навчаються с учителем. Від правильного вибору параметрів залежить не тільки те, наскільки швидко відповіді мережі будуть збігатися до правильних відповідей. Наприклад, вибір низької швидкості навчання збільшить час збігу, однак іноді дозволяє уникнути паралічу мережі. Збільшення моменту навчання може привести як до збільшення, так і до зменшення часу збігу, залежно від форми поверхні

похибки. Виходячи з такого суперечливого впливу параметрів, можна зробити висновок, що їх значення потрібно вибирати експериментально, керуючись при цьому критерієм завершення навчання (наприклад, мінімізація похибки або обмеження за часом навчання).

Власне навчання мережі

У процесі навчання мережа у визначеному порядку переглядає навчальну вибірку. Порядок перегляду може бути послідовним, випадковим тощо. Мережі, які навчаються без учителя, переглядають вибірку тільки один раз. При навчанні з учителем мережа переглядає вибірку множину разів, при цьому один повний прохід по вибірці називається **епохою навчання**.

Зазвичай набір вихідних даних поділяють на дві частини – власне навчальну вибірку і тестові дані; принцип поділу може бути довільним. Навчальні дані подаються мережі для навчання, а тестові використовуються для розрахунку похибки мережі (тестові дані ніколи для навчання мережі не застосовуються). Таким чином, якщо на тестових даних похибка зменшується, то мережа дійсно виконує узагальнення.

Якщо похибка на навчальних даних продовжує зменшуватися, а похибка на тестових даних збільшується, виходить, мережа перестала виконувати узагальнення і просто «запам’ятовує» навчальні дані. Це явище називається **перенавчанням мережі** або **оверфіттінгом**. У таких випадках навчання зазвичай припиняють. У процесі навчання можуть проявитися інші проблеми, такі як параліч або влучення мережі в локальний мінімум поверхні похибок. Неможливо заздалегідь прогнозувати прояв тієї або іншої проблеми, так само як і дати однозначні рекомендації до їх розв’язання.

Перевірка адекватності навчання

Навіть у випадку успішного, на перший погляд навчання мережа, не завжди навчається саме тому, чого від неї хотів творець. Відомий випадок, коли мережа навчалася розпізнаванням зображень танків по фотографіях, однак пізніше з’ясувалося, що всі танки були сфотографовані на тому самому тлі. У результаті мережа «навчилася» розпізнавати цей тип ландшафту, замість того, щоб «навчитися» розпізнавати танки.

Таким чином, мережа «розуміє» не те, що від неї було потрібно, а те, що найпростіше узагальнити.

3.4 Нейромережеві бібліотеки. Keras. TensorFlow. Theano. CNTK

Шари є фундаментальною структурою даних в нейронних мережах. **Шар** – це модуль обробки даних, що приймає на вході і повертає на виході один або кілька тензорів. Деякі шари не зберігають стану, але частіше це не так: ваги шару, один або кілька тензорів, навчаючись із застосуванням алгоритму стохастичного градієнтного спуску, які разом зберігають знання мережі.

Різним шарам відповідають тензори різних форматів і різні види обробки даних. Наприклад, прості векторні дані, що зберігаються у двовимірних тензорах з формою (зразки, ознаки), часто обробляються щільно пов’язаними

шарами, які також називають повнозв'язними, або щільними, шарами (клас Dense в Keras). Ряди даних зберігаються в тривимірних тензорах з формою (зразки, мітки часу, ознаки) і зазвичай обробляються рекурентними шарами, такими як LSTM. Зображення зберігаються в чотиривимірних векторах і зазвичай обробляються двовимірними згортковими шарами (Conv2D).

Шари можна вважати кубиками LEGO глибокого навчання. Фреймворки зразка Keras роблять це порівняння ще більш явним. Створення моделей глибокого навчання в Keras здійснюється шляхом об'єднання сумісних шарів в конвеєри обробки даних. Поняття сумісності шарів в такому випадку відображає лише той факт, що кожен шар приймає і повертає вектори певної форми.

Розглянемо наступний приклад:

```
rom keras import layers
layer = layers.Dense(32, input_shape=(784,))
```

повнозв'язний шар з 32 вихідними нейронами

Тут створюється шар, який бере тільки двовимірні вектори, перший вимір яких дорівнює 784 (вісь 0 – вимір пакетів – не задано, тому допустимо будь-яке значення). Цей шар повертає вектор, перший вимір якого дорівнює 32.

Іншими словами, цей шар можна пов'язати з шаром нижче, тільки якщо той приймає двовимірні вектори. Фреймворк Keras позбавляє необхідності турбуватися про сумісність, тому що шари, що додаються в моделі, автоматично конструюються так, щоб відповідати формі вхідного шару. наприклад, розглянемо наступний код:

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(32, input_shape=(784,)))
model.add(layers.Dense(32))
```

Другий шар створюється без явного значення для аргументу input_shape, тому форма вхідних даних буде автоматично виведена з форми вихідних даних попереднього шару.

Модель глибокого навчання є орієнтованим, ациклічним графом шарів. Найчастіше на практиці використовується лінійний стек шарів, що відображають єдиний вхід в єдиний вихід. Але спектр топологій мереж може бути більш широким:

- мережі з двома гілками (two-branch networks);
- багатоголові мережі (multihead networks);
- вхідні блоки (inception blocks).

Топологія мережі визначає простір гіпотез. Вибираючи топологію мережі, обмежується простір можливостей (простір гіпотез) певною послідовністю операцій з тензорами, що відображають вхідні дані у вихідні. Подальша задача – знайти хороший набір значень для вагових тензорів, залучених в ці операції з векторами.

Вибір правильної архітектури мережі – це більше мистецтво, ніж наука; і хоча є деякі методи й принципи, на які можна покластися, тільки практика може допомогти стати досвідченим архітектором нейронних мереж.

Keras – це бібліотека рівня моделі, що надає високорівневі будівельні блоки для конструювання моделей глибокого навчання. Вона не реалізує низькорівневі операції, такі як маніпуляції з вектрами й диференціювання, для цього використовується спеціалізована й оптимізована бібліотека підтримки векторів.

При цьому Keras не покладається на якусь одну бібліотеку підтримки векторів, а використовує модульний підхід (рис. 3.10); тобто до фреймворку Keras можна під'єднати декілька різних низькорівневих бібліотек. В цей час підтримуються три такі бібліотеки: TensorFlow, Theano і Microsoft Cognitive Toolkit (CNTK). В майбутньому Keras, швидше за все, буде розширений ще декількома низькорівневими механізмами підтримки глибокого навчання.

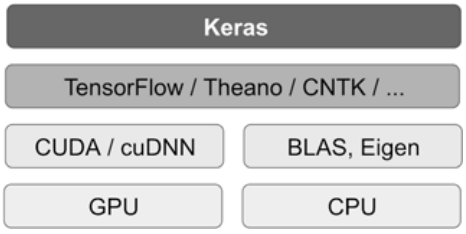


Рисунок 3.10 – Програмно-апаратний стек підтримки глибокого навчання

TensorFlow, CNTK і Theano – це одні з провідних платформ глибокого навчання. Theano (*http://deeplearning.net/software/theano*) розроблена в лабораторії MILA Монреальського університету, TensorFlow (*www.tensorflow.org*) розроблена в Google, а CNTK (*https://github.com/Microsoft/CNTK*) в Microsoft.

Будь-який код, який використовує Keras, можна запускати з будь-яких бібліотек без необхідності змінювати щось в коді: можна легко перемикатися між ними в процесі розробки, що часто виявляється корисним, наприклад, якщо одна з бібліотек показує вищу продуктивність при вирішенні даного конкретного завдання. За замовчуванням рекомендується використовувати бібліотеку TensorFlow як найбільш поширену, масштабовану і високоякісну.

Використовуючи TensorFlow (Theano або CNTK), Keras може виконувати обчислення і на CPU, і на GPU. При виконанні на CPU TensorFlow сама використовує низькорівневу бібліотеку спеціалізованих операцій з тензорами, яка називається Eigen (*http://eigen.tuxfamily.org*). При виконанні на GPU

TensorFlow використовує оптимізовану бібліотеку під назвою NVIDIA CUDA Deep Neural Network (cuDNN).

Вигляд типового процесу використання Keras:

1. Визначаються навчальні дані: вхідні та цільові тензори.

2. Визначаються верстви мережі (модель), що відображають вхідні дані в цільові.

3. Настроюється процес навчання вибором функції втрат, оптимізатора і деяких параметрів для моніторингу.

4. Виконуються ітерації по навчальних даних викликом методу fit () моделі.

Модель можна визначити двома способами: з використанням класу Sequential (Тільки для лінійного стека шарів – найбільш популярна архітектура мереж в цей час) або функціонального API (для орієнтованого ациклічного графа шарів, що дозволяє конструювати довільні архітектури).

Нижче наводиться визначення двошарової моделі з використанням класу **Sequential** (слід звернути увагу, що першому шару передається очікувана форма вхідних даних):

```
from keras import models
from keras import layers

model = models.Sequential ()
model.add (layers.Dense (32, activation = 'relu', input_shape = (784,)))
model.add (layers.Dense (10, activation = 'softmax'))
```

А ось та ж модель, але сконструйована із застосуванням функціонального API:

```
input_tensor = layers.Input (shape = (784,))
x = layers.Dense (32, activation = 'relu') (input_tensor)
output_tensor = layers.Dense (10, activation = 'softmax') (x)
model = models.Model (inputs = input_tensor, outputs = output_tensor)
```

Функціональний API дозволяє маніпулювати даними в тензори, які обробляють модель, і застосовувати шари до цих тензорів, оскільки вони були функціями.

3.5 Налаштування та встановлення бібліотеки

Для практичного використання рекомендується наступні два варіанти:

- Використовувати офіційні віртуальні машини EC2 Deep Learning AMI (*Https://aws.amazon.com/amazonai/amis*) і виконувати експерименти з Keras в блокнотах ***Jupyter Notebook*** на *EC2*. Цей варіант рекомендується всім, у кого на локальному комп'ютері немає GPU.
- Встановити все з нуля на локальну робочу станцію, що діє під керуванням UNIX. В цьому випадку можна виконувати експерименти в блокнотах ***Jupyter Notebook*** локально або запускати звичайний код на *Python*.

Цей варіант рекомендується тим, у кого на локальному комп’ютері є високопродуктивний NVIDIA GPU.

🔔 *Дослідницьке завдання:*

1) Ознайомитися з компонентами Keras, які дозволяють створювати різноманітні архітектури нейронних мереж, а також забезпечують гнучкість у виборі методів навчання і оцінки моделей.
2) Навести приклади використання Callback-функції в бібліотеці Keras.

❓ *Контрольні питання:*

1) Надати визначення: штучний нейрон. Як штучний нейрон використовується в нейронних мережах?
2) Які основні компоненти нейронної мережі і як вони взаємодіють між собою?
3) Охарактеризуйте типи штучних нейронних мереж існують і як вони відрізняються один від одного?
4) Які етапи включає процес навчання нейронної мережі? Описати кожен етап.
5) Які функції активації використовуються в нейронних мережах?
6) Пояснити, які стратегії представлення даних є ефективними для навчання нейронних мереж? Порівняти їхні переваги та недоліки.

ЗМІСТОВИЙ МОДУЛЬ 3. ОСНОВИ МАШИННОГО НАВЧАННЯ

Тема 4. Розпізнавання цифр, написаних від руки, за допомогою нейронної мережі

4.1 Розпізнавання цифр за допомогою нейронної мережі

Для реалізації даної нейронної мережі пропонується використовувати систему програмування на мові Python 3.6.4 для Windows, можна завантажити з офіційного сайту: *https://www.python.org/downloads/windows/* (рекомендується використовувати **Windows x86 executable installer**). Перед встановленням необхідно обрати пункт «Додати Python до PATH».

В якості самовчителя на мові Python можна використовувати ресурс: *https://pythonworld.ru/samouchitel-python*:

1) Встановлення бібліотеки **NumPy.**
2) Запустіть додаток «*Командний рядок*» (виберати cmd у вікні пошуку панелі завдань Windows) (рис 4.1).

```
pip3 install numpy
```

Рисунок 4.1 – Команда встановлення бібліотеки

3) Запуск команди для встановлення пакету.
4) Встановлення робочої директорії проєкту.
5) Створення каталогу **NeuralNetwork**, в якій будуть зберігатись вихідні тексти програми, створені під час виконання практичних завдань (наприклад, C:\NeuralNetwork). У каталозі **NeuralNetwork** створити підкаталог *Network1*, у якому будуть зберігатися вихідні коди завдання.

Запустити середовище розробки (для запуску середовища розробки IDLE, виберіть idle у вікні панелі пошуку задач Windows). Створити новий файл для програми (меню Файл/Новий файл). Зберегти цей файл у каталозі Network1 під іменем мережі (меню File/Save). Розширення **.py** буде встановлено за умовчанням.

Скопіювати у вікні програми **network.py** наступні команди та впишіть дані з (рис 4.2)

Збережіть файл network.py і виконати програму network. Щоб запустити програму виконання, вибрати Run/Run Module (або натисніть F5). У результаті буде створено об’єкту класу Мережа, що задає трирівневу нейронну мережу з окремими параметрами. При створенні об’єкту класу мережі ваги і розміщення ініціалізуються випадковим чином. Для ініціалізації цієї величини використовується функція np.random.randn з бібліотеки NumPy. Дана функція генерує числа з нормальним розподілом для масиву заданої розмірності.

```
"""""""""""""""""""""""""""""""""""""""""
network.py
Модуль створення та навчання нейронної
мережі для розпізнавання рукописних
цифр з використанням методу
градієнтного спуску.
Группа:<Указати номер групи>
ФИО:<Указати піб
студента>""""""""""""""""""""""""""""""""
"""""""""

import random # библиотека функцій для генерації випадкових величин


# Сторонні бібліотеки

import numpy as np # бібліотека функцій для роботи з матрицями

""" ---Розділ опису--- """
""" -Опис класу  Network--"""
class Network(object): # використовується для опису нейронної мережі
    def __init__(self, sizes): # конструктор класу
                               # self - вказівник на об'єкт класу
                               # sizes - список розмірів шарів нейронної
мер.
        self.num_layers = len(sizes) # кількість шарів нейронної мережі

        self.sizes = sizes # задаємо список розмірів шарів нейр. мер.
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]] #
випадкові навчальні зміщення
        self.weights = [np.random.randn(y, x) for x, y in zip(sizes[:-1],
sizes[1:])] # задаємо випадкові навчальні ваги зв'язків
""" -Кінець опису класу Network--""" """ -
-- Кінець розділу опису--- """

""" ---Тіло програми--- """
net = Network([2, 3, 1]) # створюємо нейронну мережу з трьох
шарів""" ---Кінець тіла програми--- """

""" Вивід результату на екран: """
print('Сеть net:')
print('Кількість шарів:', net.num_layers)
for i in range(net.num_layers):
    print('Кількість нейронів в шарі', i,':',net.sizes[i])
for i in range(net.num_layers-1):
    print('W_',i+1,':')
    print(np.round(net.weights[i],2))
    print('b_',i+1,':')
    print(np.round(net.biases[i],2))
```

Рисунок 4.2 – Приклад програми

Функцією активації для мережі нейронів використана сигмоїдна функція, що обчислює вихідний сигнал штучного нейрона. Нижче представлений код для визначення функції (рис 4.3). Необхідно додати цей код у розділ описаної програми **network.py**:

```
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))
```

Рисунок 4.3 – Визначення сигмоїдальної функції активації

Для опису сигмоїдальної функції активації використовується функція для вирахування експонентів з бібліотеки NumPy, що дозволяє передавати масив у вигляді вхідного параметра сигмоїдальної функції. У цьому випадку функція експонентів застосовується поелементно, тобто є у векторизованій формі.

Для реалізації механізму навчання створюваної нейронної мережі додаємо метод SGD, який реалізує стохастичний градієнтний спуск. Метод має такі параметри:

«**Training_data**» – навчальна вибірка, яка складається з пар виду ($\vec{x}$, $\vec{y}$), де $\vec{x}$ – вектор вхідних сигналів, а $\vec{y}$ – очікуваний вектор вихідних сигналів;

«**epochs**»– кількість епох навчання;

«**mini_batch_size**» – розмір підвибірки;

«**eta**» – швидкість навчання;

«**test_data**» – (необов'язковий параметр); якщо цей аргумент не пустий, то програма після кожної епохи навчання здійснює оцінку роботи мережі та показує досягнутий прогрес.

Програмний код методу SGD описаний в **Додатку А**.

Розглянемо конкретний приклад нейронної мережі, яка навчається класифікації рукописних цифр і створена за допомогою бібліотеки Keras для Python: реалізувати класифікацію чорно-білих зображень рукописних цифр (28×28 пікселів) по 10 категоріям (від 0 до 9). Будемо використовувати набір даних **MNIST**, популярний в спільноті дослідників глибокого навчання, який існує практично стільки ж, скільки сама галузь машинного навчання, і широко використовується для навчання. Цей набір містить 60 000 навчальних зображень і 10 000 контрольних зображень, зібраних Національним інститутом стандартів і технологій США (National Institute of Standards and Technology – частина NIST в абревіатурі MNIST) в 1980-х. «Розв'язок» завдання MNIST можна розглядати як своєрідний аналог «Hello World» в глибокому навчанні – часто це перша дія, що виконується, щоб переконатися, що алгоритми діють в точності як очікувалося. Деякі зразки зображень з набору MNIST можна бачити на рис 4.4.

У машинному навчанні категорія в задачі класифікації називається класом. Елементи вихідних даних називаються абзацами. Клас, пов'язаний з конкретним зразком, називається міткою.

Рисунок 4.4 – Зразки зображень MNIST

Щоб випробувати цей приклад, потрібно спочатку встановити бібліотеку Keras.

Набір даних MNIST вже входить до складу Keras в формі набору з чотирьох масивів Numpy.

Лістинг 1. Завантаження набору даних MNIST в Keras

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data ()
```

Тут train_images і train_labels – це тренувальний набір, тобто дані, необхідні для навчання. Після навчання модель буде перевірятися тестовим (або контрольним) набором, test_images і test_labels.

Зображення зберігаються в масивах Numpy, а мітки – в масиві цифр від 0 до 9. Зображення та мітки знаходяться в прямій відповідності один до одного.

Розглянемо навчальні дані:

```
>>> train_images.shape
(60000, 28, 28)
>>> len (train_labels)
60000
>>> train_labels
array ([5, 0, 4, ..., 5, 6, 8], dtype = uint8)
```

І контрольні дані:

```
>>> test_images.shape
(10000, 28, 28)
>>> len (test_labels)
10000
>>> test_labels
```

array ([7, 2, 1, ..., 4, 5, 6], dtype = uint8)

Спочатку нейронній мережі передаються навчальні дані, train_images і train_labels. В результаті цього мережа навчиться зіставляти зображення з мітками. Тепер сконструюємо мережу.

Лістинг 2.. Архітектура мережі

```
from keras import models
from keras import layers
network = models.Sequential ()
network.add (layers.Dense (512, activation = 'relu', input_shape = (28 * 28,)))
network.add (layers.Dense (10, activation = 'softmax'))
```

Основним будівельним блоком нейронних мереж є шар (або рівень), модуль обробки даних, який можна розглядати як фільтр для даних. Він приймає деякі дані і виводить їх у більш корисній формі. Фактично методика глибокого навчання полягає в об'єднанні простих верств, що реалізують деяку форму поетапного очищення даних. Модель глибокого навчання можна порівняти з ситом, що складається з послідовності фільтрів більш тонкого очищення даних – шарів.

В такому випадку наша мережа складається з послідовності двох шарів Dense, які є тісно пов'язаними (їх ще називають повнозв'язаними) нейронними шарами. Другий (і останній) шар – це 10-змінний шар втрат (softmax layer), який повертає масив з 10 оцінками ймовірностей (в сумі дають 1). Кожна оцінка визначає ймовірність приналежності поточного зображення до одного з 10 класів цифр.

Щоб підготувати мережу до навчання, потрібно налаштувати ще три параметри для етапу *компіляції*:

– *функцію втрат*, яка визначає, як мережа повинна оцінювати якість своєї роботи на навчальних даних і, відповідно, як коригувати її в правильному напрямку;

– *оптимізатор* – механізм, за допомогою якого мережа буде оновлювати себе, спираючись на дані, які спостерігаються й функцію втрат;

– *метрики для моніторингу на етапах навчання і тестування* – тут цікавить тільки точність (частка правильно класифікованих зображень).

Призначення функції втрат і оптимізатора з'ясуємо в наступних розділах [5].

Лістинг 3. Етап компіляції

```
network.compile (optimizer = 'rmsprop',
loss = 'categorical_crossentropy',
metrics = [ 'accuracy'])
```

Перед навчанням виконується попередня обробка даних для перетворення їх у форму, яку очікує отримати нейронна мережа, і масштабується їх так, щоб всі значення виявилися в інтервалі [0, 1]. Вихідні дані – навчальні зображення – зберігаються в тривимірному масиві (60000, 28, 28) тип uint8, значеннями в якому є числа в інтервалі [0, 255]. Перетворимо його в масив (60000, 28*28) типу float32 зі значеннями в інтервалі [0, 1].

Лістинг 4. Підготовка вихідних даних

```
train_images = train_images.reshape ((60000, 28 * 28))
train_images = train_images.astype ( 'float32') / 255

test_images = test_images.reshape ((10000, 28 * 28))
test_images =test_images.astype ( 'float32') / 255
```

Також потрібно закодувати мітки категорій.

Лістинг 5. Підготовка міток

```
from keras.utils import to_categorical
train_labels = to_categorical (train_labels)
test_labels = to_categorical (test_labels)
```

Тепер можна починати навчання мережі, для чого в разі використання бібліотеки Keras досить викликати метод fit мережі – він намагається адаптувати (fit) модель під навчальні дані:

>>> Network.fit (train_images, train_labels, epochs = 5, batch_size = 128)
>>> Epoch 1/5
60000/60000 [==============================] - 9s - loss: 0.2524 - acc: 0.9273
Epoch 2/5
51328/60000 [============================> .....] - ETA: 1s - loss: 0.1035 - acc:
0.9692

У процесі навчання можна побачити дві величини: втрати мережі на навчальних даних і точність мережі на навчальних даних.

В такому випадку було досягнуто точності 0,989 (98,9%) на навчальних даних. Тепер перевіримо, як модель розпізнає контрольний набір:

>>> test_loss, test_acc = network.evaluate (test_images, test_labels)
>>> Print ('test_acc:', test_acc)
test_acc: 0.9785

Точність на контрольному наборі склала 97,8% – трохи менше, ніж на тренувальному наборі. Ця різниця між точністю на тренувальному і контрольному наборах демонструє приклад перенавчання (overfitting), коли моделі машинного навчання показують гіршу точність на новому наборі даних в порівнянні з тренувальним.

🔔 *Дослідницьке завдання:*

1) Розглянути покращення точності розпізнавання рукописного тексту з використанням гібридної архітектури CNN-LSTM.

2) Дослідити методи навчання та стратегії обробки даних для розпізнавання рукописного тексту з нестандартними або рідкісними стилями письма, такими як курсив, рукописні шрифти, або текст, написаний дітьми.

❓ *Контрольні питання:*

1) Назвати етапи, які включає процес розпізнавання цифр за допомогою нейронної мережі?

2) Які дані необхідні для тренування нейронної мережі для розпізнавання цифр?

3) Які функції активації можуть бути використані для розпізнавання цифр, і чому вибір необхідної функції активації є важливим?

4) Які існують підходи для покращення точності розпізнавання цифр за допомогою нейронної мережі?

5) Навести практичні приклади застосування систем розпізнавання цифр в реальному житті?

6) Як розпізнавання цифр за допомогою нейронної мережі відрізняється від інших підходів, наприклад, класичних методів машинного навчання?

Тема 5. Види машинного навчання. Оцінка моделей машинного навчання. Налагодження машинного навчання. Перенавчання. Недонавчання.

5.1 Види машинного навчання. Оцінка моделей машинного навчання. Налагодження машинного навчання.

Мета машинного навчання – передбачити результат за вхідними даними. Чим різноманітніші вхідні дані, тим простіше машині знайти закономірності й тим точніший результат.

Отже, якщо необхідно навчити машину, потрібні три речі: дані, ознаки, алгоритми (рис 5.1).

Рисунок 5.1 – Типи машинного навчання

Термін: «Кинь робота в лабіринт і нехай шукає вихід» використовують для автопілотів автомобілів, роботів, ігор, автоматизованої торгівлі, управління ресурсами підприємств.

Популярні алгоритми:

- Q-Learning,
- SARSA,
- DQN,
- A3C,
- Генетичний Алгоритм.

Навчання з підкріпленням використовують там, де задача полягає не в аналізі даних, а у виживанні в реальному середовищі.

Середовищем може бути навіть відеогра. Роботи, які грають в Маріо, були популярні ще років п’ять тому.

Середовищем може бути навіть реальний світ. Як приклад – автопілот Тесли, який вчиться не збивати пішоходів, або роботи-порохотяги, головне завдання яких – налякати вашого кота з максимальною ефективністю.

Знання про навколишній світ такому роботу можуть бути корисні, але лише для довідки. Не важливо скільки даних він збере, у нього все одно не вийде передбачити всі ситуації. Тому його мета – мінімізувати помилки, а не передбачати всі ходи. Робот вчиться виживати в просторі з максимальною вигодою: зібраними монетками в Маріо, тривалістю поїздки в Теслі.

Саме виживання в середовищі і є ідеєю навчання з підкріпленням. Киньмо бідного робота в реальне життя, будемо штрафувати його за помилки й нагороджувати за правильні вчинки.

Розумні моделі роботів-порохотягів і самокеровані автомобілі навчаються саме так: їм створюють віртуальне місто (часто на основі мап справжніх міст), населяють випадковими пішоходами й відправляють вчитися нікого там не вбивати. Коли робот починає добре себе почувати в штучному GTA, його випускають тестувати на реальні вулиці [10-11].

Запам’ятовувати саме місто машині не потрібно – такий підхід називається **Model-Free**. Звичайно, тут є і класичний **Model-Based**, але в ньому нашій машині довелося б запам’ятовувати модель всієї планети, всіх можливих ситуацій на всіх перехрестях світу. Таке просто не працює. У навчанні з підкріпленням машина не запам’ятовує кожен рух, а намагається узагальнити ситуації, щоб виходити з них з максимальною вигодою (рис 5.2).

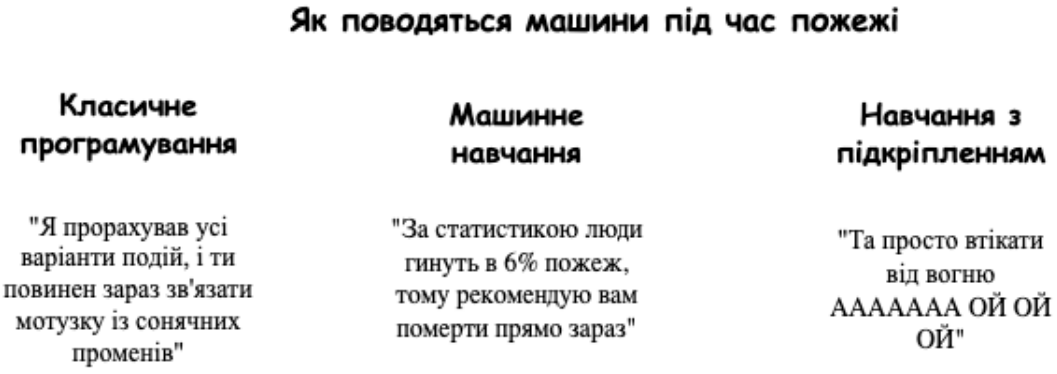


Рисунок 5.2 – Приклад поведінки інформаційної системи

Пам’ятаєте новину кількарічної давнини, коли машина обіграла людину в Ґо? Хоча незадовго до цього було доведено, що кількість комбінацій фізично неможливо прорахувати, адже вона перевищує кількість атомів у всесвіті. Тобто, якщо в шахах машина реально може прорахувати усі майбутні комбінації й перемагати, з Ґо така ідея нездійсненна. Тому вона просто вибирала найкращий вихід з кожної ситуації й робила це досить точно, щоб обіграти якогось «гравця».

Ця ідея лежить в основі алгоритму Q-learning і його похідних (SARSA і DQN). Буква Q в назві означає слово Quality, тобто робот вчиться робити найякісніші кроки в будь-якій ситуації, а всі ситуації він запам’ятовує як простий «марківський процес» (рис 5.3) .

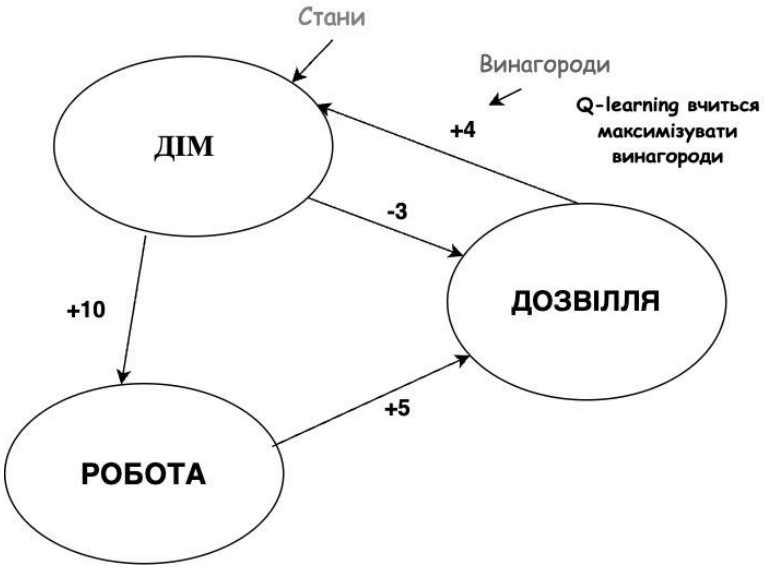


Рисунок 5.3 – «Марківський процес»

Машина проганяє мільйони симуляцій в середовищі, запам’ятовуючи всі сформовані ситуації й виходи з них, які принесли максимальну винагороду. Але як зрозуміти, коли у нас склалася знайома ситуація, а коли абсолютно нова? Ось самокеруючий автомобіль стоїть біля перехрестя і загоряється зелений – значить можна їхати? А якщо справа мчить швидка допомога з мигалками?

Одні прописують всі ситуації руками, що дозволяє їм обробляти виняткові випадки виду проблеми вагонетки. Інші йдуть глибше і віддають цю роботу нейромережам, нехай самі все знайдуть. Так замість Q-learning’а з’являється Deep Q-Network (DQN).

Reinforcement Learning для простого користувача виглядає як справжній інтелект. Тому що «wow, машина сама приймає рішення в реальних ситуаціях!»

Генетичні алгоритми теж належать до навчання з підкріпленням.

Ансамблі та нейромережі – це головні алгоритми на шляху до неминучої сингулярності. Сьогодні вони дають найточніші результати і використовуються усіма великими компаніями в продуктивному середовищі.

Разом з усією їхньою ефективністю, ідея досить проста. Виявляється, якщо взяти декілька не дуже ефективних методів навчання і навчити виправляти помилки один одного, якість такої системи буде набагато вище, ніж кожного з методів окремо.

Причому навіть краще, коли взяті алгоритми максимально нестабільні й сильно плавають стосовно вхідних даних. Тому частіше беруть Регресію і Дерева Рішень, яким достатньо однієї сильної аномалії в даних, щоб поїхала вся

модель. А ось метод Баєса і K-NN не беруть ніколи – вони хоч і недостатньо «розумні», але дуже стабільні.

Ансамбль можна зібрати як завгодно, але хоч випадково покласти у класифікатори й додати регресію. Є три перевірених способи робити ансамблі.

Стекінг. Навчаємо кілька різних алгоритмів і передаємо їх результати на вхід останньому, який приймає остаточне рішення (рис 5.4).

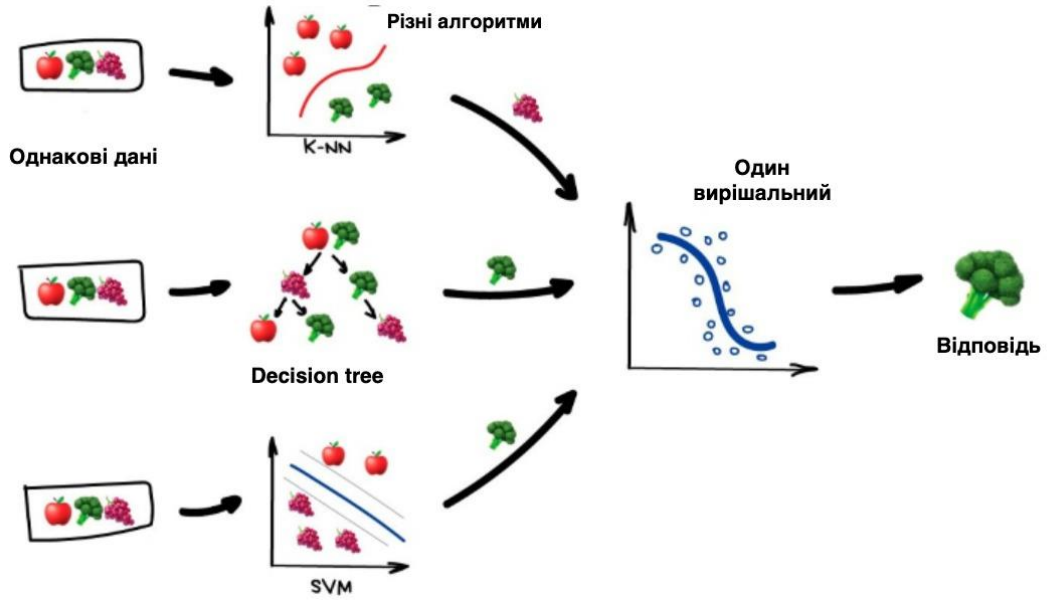


Рисунок 5.4 – Стекінг

Ключові слова з різних алгоритмів, адже один і той же алгоритм, навчений на одних і тих же даних не має сенсу. Яких – справа розробника, хіба що в ролі вирішального алгоритму частіше беруть регресію.

Стекінг на практиці застосовується рідко, тому що два інших методи зазвичай точніші.

Бегінг. Він же Bootstrap AGGregatING. Навчаємо один алгоритм багато разів на випадкових вибірках з вихідних даних. В кінці усереднюємо відповіді.

Дані в випадкових вибірках можуть повторюватися. Тобто з набору 1-2-3 ми можемо робити вибірки 2-2-3, 1-2-2, 3-1-2 тощо. На них можна навчити один і той самий алгоритм кілька разів, а в кінці знаходимо відповідь простим голосуванням.

Найпопулярніший приклад бегінга – алгоритм Random Forest, бегінг на деревах, який намальований на рисунку вище. Коли ви відкриваєте камеру на телефоні і бачите як вона окреслила обличчя людей в кадрі жовтими прямокутниками – швидше за все це дана робота. Нейромережа буде занадто повільна в реальному часі, а бегінг ідеальний, адже він може обчислювати свої дерева паралельно на всіх шейдерах відеокарти (рис.5.6).

Здатність запаралелитися дає бегінгу перевагу навіть над наступним методом, який працює точніше, але тільки в один потік. Хоча можна розбити на сегменти, запустити кілька (рис 5.5).

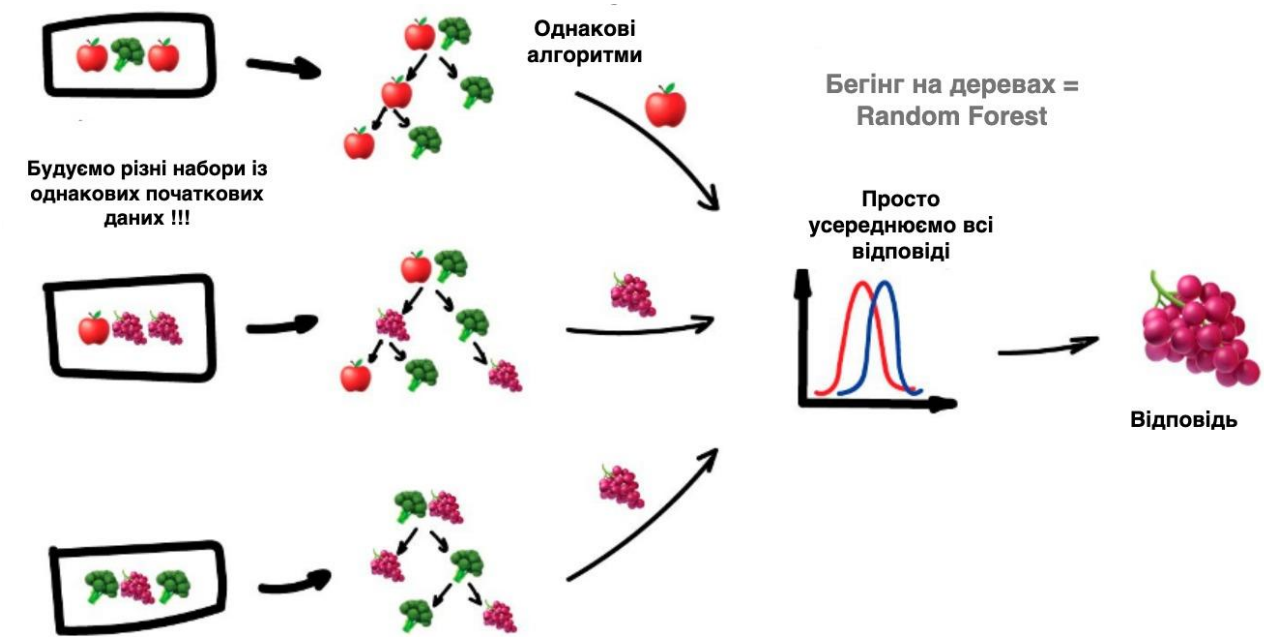


Рисунок 5.5 – Бегінг

Рисунок 5.6 – Розпізнавання облич

Бустинг. Навчаємо алгоритми послідовно, кожен наступний приділяє особливу увагу тим випадкам, на яких помилився попередній.

Як і в бегінгу, робимо вибірки з вихідних даних, але тепер не зовсім випадково. У кожну нову вибірку беремо частину тих даних, на яких попередній алгоритм відпрацював неправильно. Тобто, так би мовити, донавчаємо новий алгоритм на помилках попереднього (рис 5.7).

Плюси – висока точність класифікації. Мінуси вже названі – не запаралелиться.

Сьогодні є три популярних методи бустингу, відмінності між якими добре висвітлено у статті CatBoost vs. LightGBM vs. XGBoost [7].

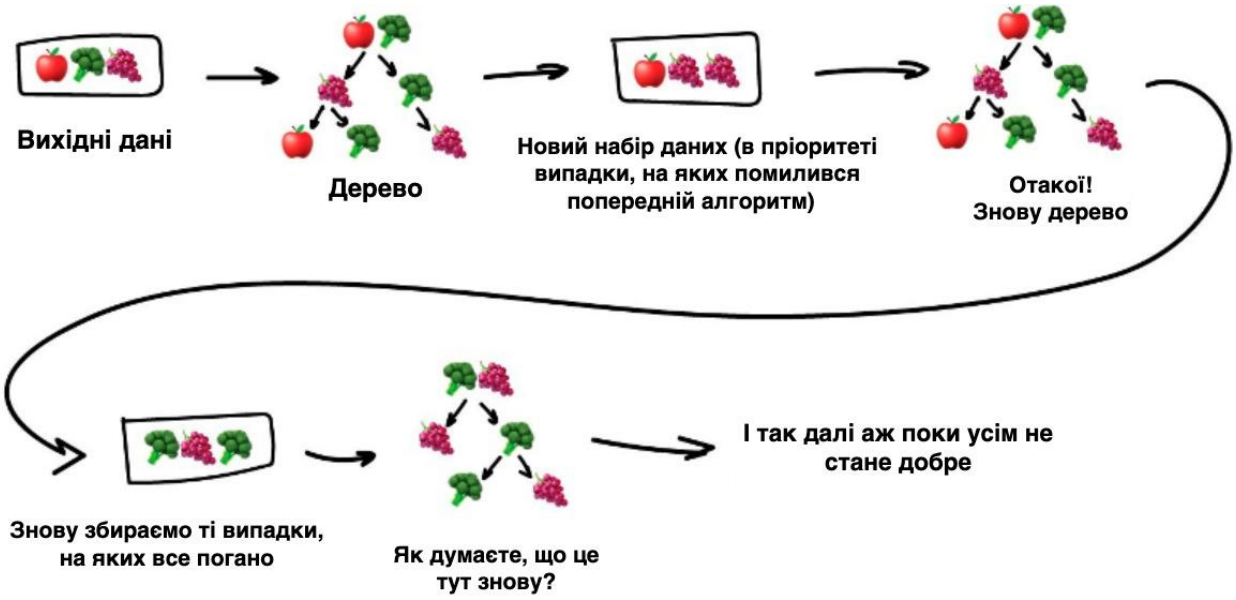


Рисунок 5.7 – Бустинг

Рекурентні нейромережі (RNN). Друга за популярністю архітектура на сьогодні. Завдяки рекурентним мережам є такі корисні речі, як машинний переклад текстів і комп'ютерний синтез мови. На них розв'язують усі завдання, пов'язані з послідовностями – голосові, текстові або музичні.

Це відбувається тому, що сучасні голосові помічники навчають говорити не буквами, а фразами. Але одразу змусити нейромережу повністю видавати фрази не вийде, адже тоді їй треба буде запам'ятати всі фрази в мові і їх розмір буде велетенським. Тут на допомогу приходить той факт, що текст, мова або музика – це послідовності. Кожне слово або звук – це, так би мовити, самостійна одиниця, але яка залежить від попередніх [12].

Можна достатньо легко навчити мережу вимовляти окремі слова або букви. Беремо деяку кількість розмічених на слова аудіофайлів і навчаємо по вхідному слову видавати нам послідовність сигналів, схожих на вимову. Порівнюємо з оригіналом від диктора і намагаємося максимально наблизитися до ідеалу. Для такого підійде навіть перцептрон.

Ось тільки з послідовністю знову проблема, адже перцептрон не запам'ятовує те, що він генерував раніше. Для нього кожен запуск як перший раз. З'явилася ідея додати до кожного нейрона пам'ять. Так були придумані рекурентні мережі, в яких кожен нейрон запам'ятовував всі свої попередні відповіді й при наступному запуску використовував їх як додатковий вхід.

Була лише одна проблема – коли кожен нейрон запам'ятовував всі минулі результати, в мережі виникала така велика кількість входів, що навчити таку кількість зв'язків ставало неможливо.

Коли нейромережа не вміє забувати – її не можна навчити (рис 5.8).

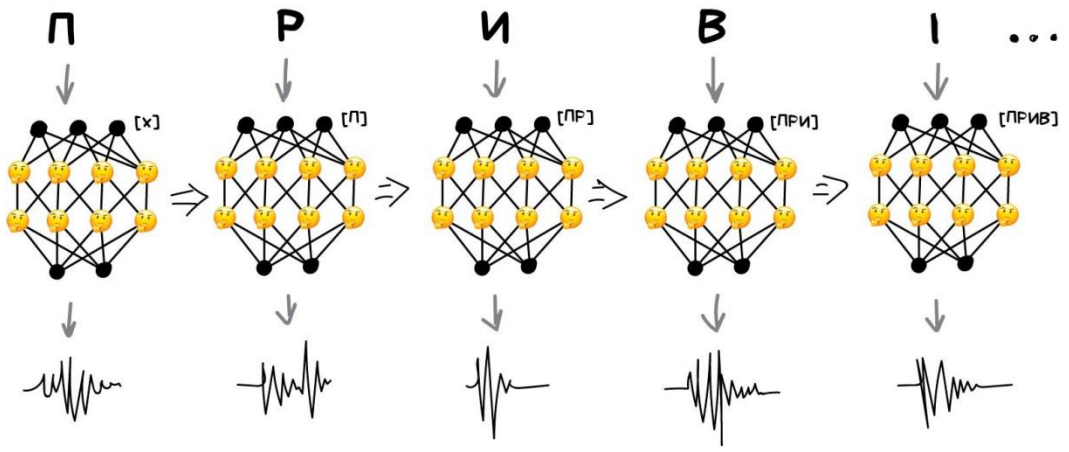


Рисунок 5.8 – Рекурентна нейронна мережа

Коли нейрону було потрібно поставити собі нагадування на майбутнє – він писав це в комірку, коли навпаки вся історія ставала непотрібною (речення, наприклад, закінчилося) – комірки видалялися, залишаючи тільки «довгострокові» зв’язки, як в класичному перцептроні. Іншими словами, мережа навчалася не тільки встановлювати поточні зв’язки, а й ставити нагадування.

Озвучені тексти для навчання почали брати звідки завгодно. На цьому прикладі видно, що імітувати голос – досить просте завдання для сьогоднішніх машин (рис 5.9).

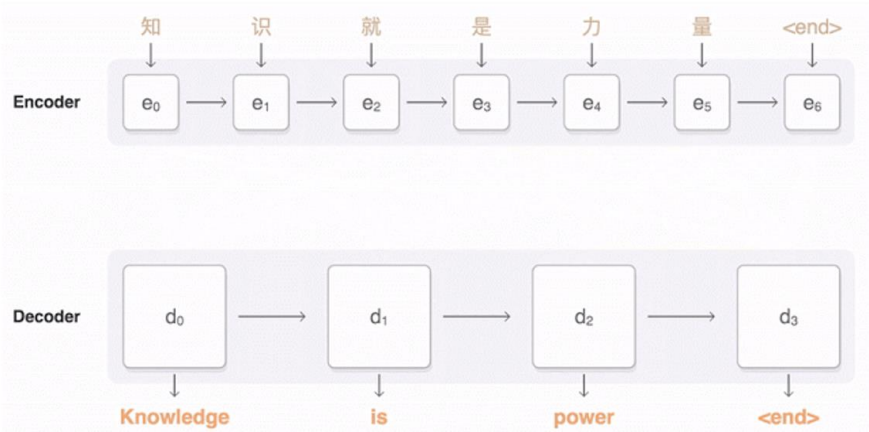


Рисунок 5.9 – Принцип імітації голосу

5.2 Перенавчання. Недонавчання

При навчанні нейронних мереж часто виникає проблема перенавчання (overfitting). **Перенавчання,** або надмірно близька підгонка – зайва точна відповідність нейронної мережі до конкретного набору навчальних прикладів, при якому мережа втрачає здатність до узагальнення.

Перенавчання виникає в разі занадто довгого навчання, недостатньої кількості навчальних прикладів або переускладненої структури нейронної мережі.

Перенавчання пов’язано з тим, що вибір навчальної (тренувальної) множини є випадковим. З перших кроків навчання відбувається зменшення похибки. На наступних кроках, з метою зменшення похибки (цільової функції) параметри підлаштовуються під особливості навчальної множини. Однак при цьому відбувається «підлаштування» не під загальні закономірності ряду, а під особливості його частини – навчальної підмножини. При цьому точність прогнозу зменшується.

Один з варіантів боротьби з перенавчанням мережі – поділ навчальної вибірки на дві множини (*навчальну і тестову*).

На навчальній множині відбувається навчання нейронної мережі. На тестовій множині здійснюється перевірка побудованої моделі. Ці множини не повинні перетинатися.

З кожним кроком параметри моделі змінюються, однак постійне зменшення значення цільової функції відбувається саме на навчальній множині. При розбитті множини на дві можна спостерігати зміну похибки прогнозу на тестовій множині паралельно зі спостереженнями над навчальною множиною. Через певну кількість кроків похибка прогнозу зменшується на обох множинах. Однак на певному етапі похибка на тестовій множині починає зростати, при цьому похибка на навчальній множині продовжує зменшуватися. Цей момент вважається кінцем реального або справжнього навчання, з нього і починається перенавчання [13].

Недонавчання (underfitting) нейронної мережі – це ситуація, коли модель не може добре відтворювати дані як на тренувальній, так і на тестовій вибірках. Це означає, що модель є занадто простою, щоб виявити закономірності в даних. Основними причинами недонавчання можна виділити:

– **занадто проста модель**: модель має недостатню кількість параметрів або шарів для відтворення складних закономірностей у даних;

– недостатня кількість навчальних даних: маленький обсяг даних може призвести до того, що модель не зможе навчитися необхідним паттернам;

– **неправильна підготовка даних:** неправильне масштабування або нормалізація даних можуть перешкоджати моделі вивчати правильні зв’язки між змінними;

– **неправильно обрані гіперпараметри:** наприклад, занадто великий коефіцієнт регуляризації або занадто велика швидкість навчання.

Щоб уникнути недонавчання, можна вжити наступних заходів:

– **ускладнити модель:** додати більше шарів або нейронів до нейронної мережі;

– **зібрати більше даних:** додаткові дані можуть допомогти моделі краще зрозуміти закономірності;

– **поліпшити підготовку даних**: використовувати правильні методи масштабування та нормалізації;

– **налаштувати гіперпараметри:** підібрати оптимальні значення для коефіцієнтів регуляризації, швидкості навчання тощо.

Загалом, для досягнення балансу між недонавчанням та перенавчанням (overfitting) потрібне ретельне налаштування моделі та робота з даними.

🔔 *Дослідницьке завдання:*

1) Дослідити методи автоматизованого налагодження гіперпараметрів (AutoML) для різних моделей машинного навчання, а також оцінити вплив різних підходів (наприклад, байєсівської оптимізації, випадкового пошуку, grid search) на кінцеву продуктивність моделей у завданнях прогнозування часового ряду або класифікації.

❓ *Контрольні питання:*

1) Яке явище називають перенавчанням в машинному навчанні і чому воно виникає?
2) Які стратегії можна використовувати для запобігання перенавчанню?
3) Яке явище називають недонавчанням?
4) Назвати типи машинного навчання.
5) Охарактеризувати рекурентні нейромережі.

ЗМІСТОВИЙ МОДУЛЬ 4. ОБРОБКА ДАНИХ. КОНСТРУЮВАННЯ ОЗНАК. НАВЧАННЯ ОЗНАК

Тема 6. Згорткові Нейромережі (CNN) та деякі типи прямого поширення

6.1 Принцип роботи CNN

Згорткові мережі зараз на піку популярності. Вони використовуються для пошуку об’єктів на фото і відео, розпізнавання осіб, перенесення стилю, генерації й домальовування зображень, створення ефектів типу слоу-мо і поліпшення якості фотографій. Сьогодні CNN застосовують всюди, де є картинки або відео. Навіть в айфоні кілька таких мереж дивляться на фотографії, щоб розпізнати об’єкти на них.

Картинка нижче – результат роботи бібліотеки Detectron, яку Facebook нещодавно оновив (рис 6.1).

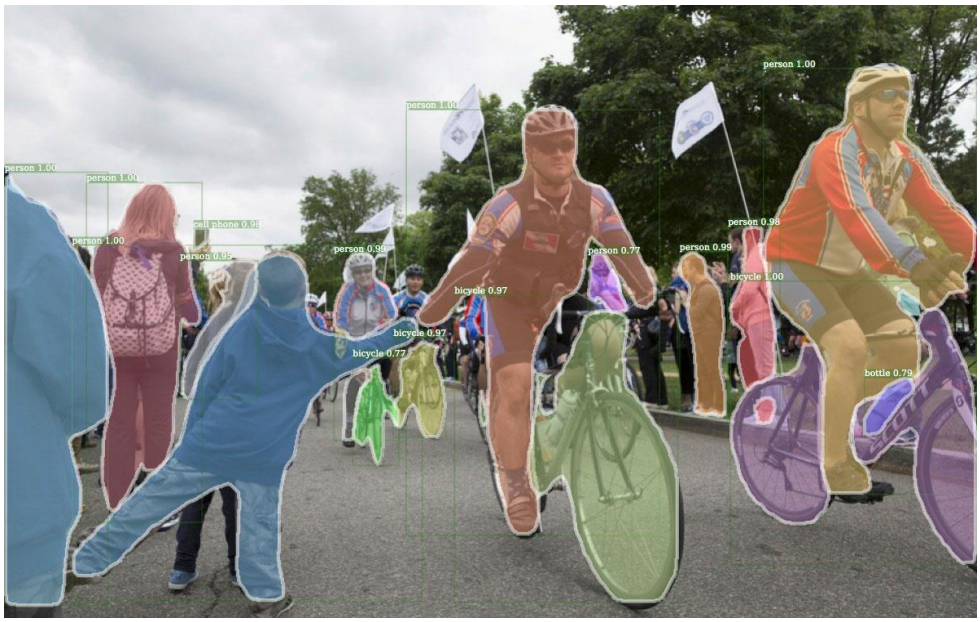


Рисунок 6.1 – Робота бібліотеки Detectron

Проблеми з зображеннями завжди були в тому, що незрозуміло, як виділяти на них ознаки. Текст можна розбити за реченнями, взяти властивості слів зі словників. Картинки ж доводилося мітити вручну, пояснюючи машині, де у котика на фотографії вушка, а де хвіст. Такий підхід навіть назвали «handcrafting ознак».

Проблем у ручного рафтингу багато. По-перше, якщо котик на фотографії притиснув вушка або відвернувся – нейромережа нічого не побачить (рис 6.2).

По-друге, складно назвати хоча б десять характерних ознак, що відрізняють котиків від інших тварин. Однак коли вночі повз людини пробігає чорна пляма, навіть краєм ока вона може сказати котик це, чи щур. Тому що людина не дивиться тільки на форму вух і кількість лап – вона оцінює об'єкт за купою різних ознак, про які навіть сама не замислюється. А отже, не розуміє і не може пояснити машині.

Рисунок 6.2 – «handcrafting ознак»

Як наслідок, складається враження, що машині треба самій вчитися шукати ці ознаки, складаючись з якихось базових ліній. Будемо робити так: для початку розділимо зображення на блоки 8x8 пікселів і виберемо яка лінія домінує в кожному – горизонтальна [-], вертикальна [|] або одна з діагональних [/]. Можуть і дві, і три, так теж буває.

На виході отримаємо кілька масивів паличок, які, по суті, є простими ознаками наявності обрисів об'єктів на зображенні. По суті, це теж картинки, просто з паличок. Отже, можемо знову вибрати блок 8x8 і подивитися вже, як ці палички поєднуються одна з одною. А потім ще і ще, і ще...

Така операція називається згорткою, звідки і пішла назва методу. Згортку можна уявити як шар нейромережі, адже нейрон – абсолютно будь-яка функція. (рис 6.3)

Рисунок 6.3 – Згорткова нейромережа

Коли ми «проганяємо» через нашу нейромережу купу фотографій котів, вона автоматично розставляє великі ваги тим сполученням з паличок, які побачила найчастіше. Причому неважливо, це пряма лінія, спини або складний

геометричний об’єкт типу мордочки – щось обов’язково буде яскраво активуватися.

На виході ж поставимо простий перцептрон, який буде дивитися які поєднання активувалися і говорити кому вони більше характерні – кішці або собаці (рис 6.4).

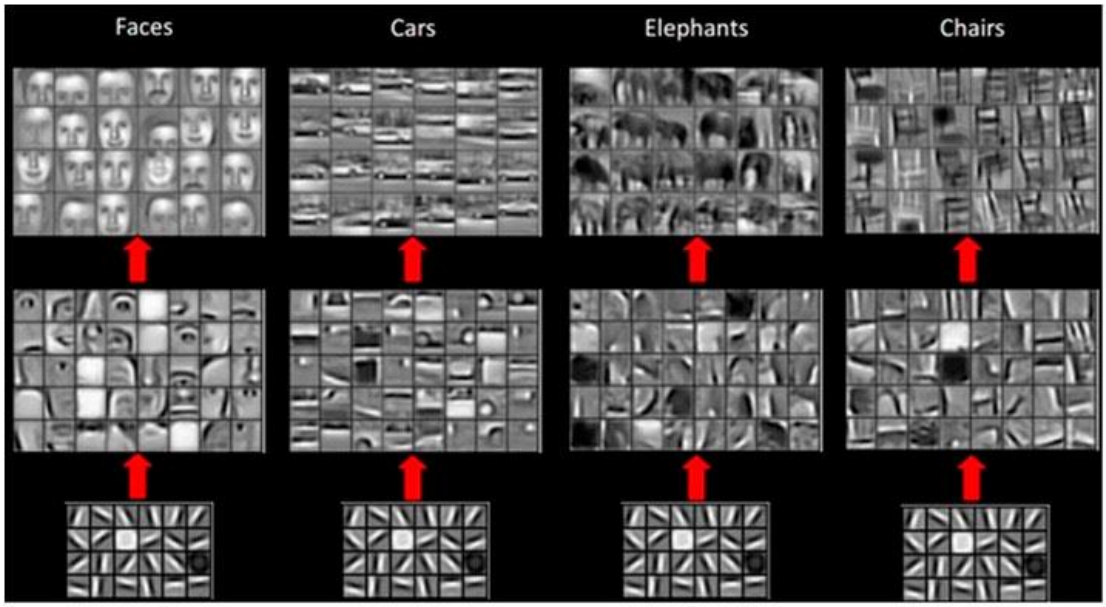


Рисунок 6.4 – Перцептрон

Зручність полягає в тому, що вийшла нейромережа, яка сама знаходить характерні ознаки об’єктів. Більше не треба відбирати їх вручну. Можна скільки завгодно «годувати» її зображеннями будь-яких об’єктів, просто наґуґліть мільйон картинок з ними – мережа сама складе мапи ознак з паличок і навчиться визначати що завгодно.

6.2 Деякі типи мереж прямого поширення

Першою моделлю нейромереж вважають **перцептрон Розенблата**. Теорія перцептронів є основою для багатьох типів штучних нейромереж прямого поширення і вони є класикою для вивчення.

Одношаровий перцептрон здатний розпізнавати найпростіші образи. Окремий нейрон обчислює зважену суму сигналів вхідних елементів, віднімає значення зсуву і пропускає результат через жорстку порогову функцію, вихід якої дорівнює +1 чи -1. Залежно від значення вихідного сигналу приймається рішення:

+1 - вхідний сигнал належить до класу А,

-1 - вхідний сигнал належить до класу В.

На рис 6.5 показано схему одношарового перцептрона, графік передатної функції і схему вирішальних галузей, створених у багатовимірному просторі вхідних сигналів. Вирішальні галузі визначають, які вхідні образи будуть

віднесені до класу A, які – до класу B. Перцептрон, що складається з одного нейрона, формує дві вирішальні галузі, які розділено гіперплощиною.

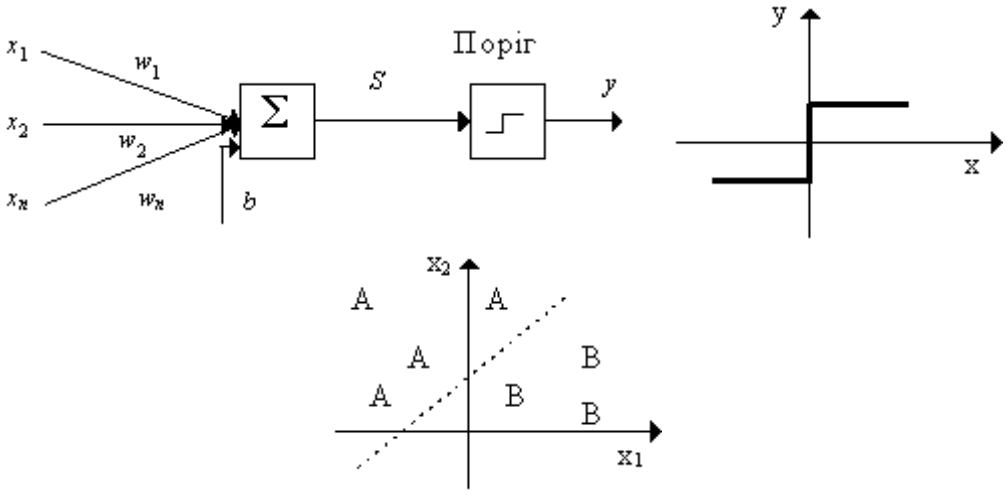


Рисунок 6.5 – Схема нейрона, графік передатної функції й поверхня, що поділяється

На рис 6.6 показано випадок з розмірністю вихідного сигналу – 2. Поверхня, що поділяється, є прямою лінією на площині. Рівняння задає пряму, що поділяється, залежить від значень синаптичних ваг і зсуву.

Розмірності входу і виходу обмежені при програмній реалізації тільки можливостями обчислювальної системи, на якій моделюється нейронна мережа, при апаратній реалізації – технологічними можливостями.

Галузі застосування: розпізнавання образів, класифікація.

Недоліки. Примітивні поверхні, що поділяються (гіперплощини) дають можливість вирішувати лише найпростіші задачі розпізнавання.

Переваги. Програмні та апаратні реалізації моделі прості. Простий і швидкий алгоритм навчання.

Модифікації. Багатошарові перцептрони дають можливість будувати складні поверхні, що поділяються і є більш поширеними.

На рис 6.6. показана схема перцептрона з декількома входами та виходами.

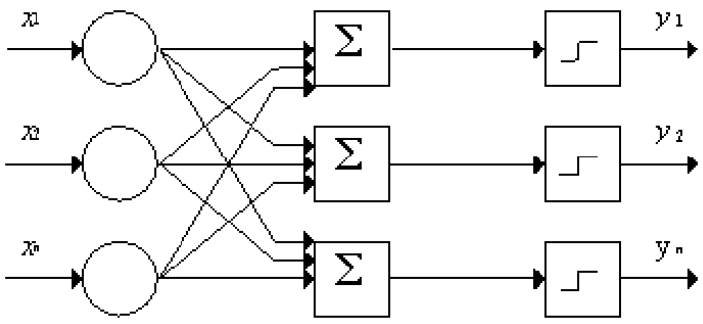


Рисунок 6.6 – Перцептрон з декількома входами та виходами

Мережа Кохонена. Мережа розроблена Теуво Кохоненом на початку 1980-х рр. і принципово відрізняється від розглянутих вище мереж, оскільки використовує неконтрольоване навчання і навчальна множина складається лише зі значень вхідних змінних.

Мережа розпізнає кластери в навчальних даних і розподіляє дані до відповідних кластерів. Якщо надалі мережа зустрічається з набором даних, несхожим ні з одним із відомих зразків, вона відносить його до нового кластеру. Якщо в даних містяться мітки класів, то мережа спроможна розв'язувати задачі класифікації. Мережі Кохонена можна використовувати й в задачах, де класи є відомими – перевага буде у спроможності мережі виявляти подібність між різноманітними класами.

Мережа Кохонена має лише два прошарки: вхідний і вихідний її ще називають самоорганізовуваною мапою. Елементи карти розташовуються в деякому просторі, як правило, двовимірному. Мережа Кохонена навчається методом послідовних наближень. У процесі навчання на входи подаються дані, але мережа при цьому підлаштовується не під еталонне значення виходу, а під закономірності у вхідних даних. Починається навчання з вибраного випадковим чином вихідного розташування центрів [14].

В процесі послідовної подачі на вхід мережі навчальних прикладів визначається найбільш схожий нейрон (той, у якого скалярний добуток ваг і поданого на вхід вектора є мінімальним). Цей нейрон оголошується переможцем і є центром при підлаштуванні ваг в сусідніх нейронів. Таке правило навчання передбачає «змагальне» (від «змагання») навчання з врахуванням відстані нейронів від «нейрона-переможця».

Навчання при цьому полягає не в мінімізації помилки, а в підлаштуванні ваг (внутрішніх параметрів нейронної мережі) для найбільшого збігу з вхідними даними (рис 6.7).

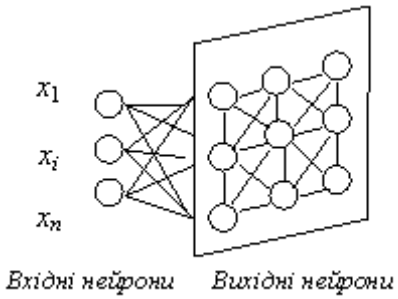


Рисунок 6.7 – Мережа Кохонена

Мережа Кохонена навчається методом послідовних наближень. Починаючи з випадковим чином обраного вихідного розташування центрів, алгоритм поступово поліпшується для кластеризації навчальних даних. Основний ітераційний алгоритм Кохонена послідовно проходить ряд епох, на кожній з яких обробляється один приклад з навчальної вибірки. Вхідні сигнали послідовно пред'являються мережі, при цьому бажані вихідні сигнали не

визначаються. Після пред'явлення достатнього числа вхідних векторів синаптичні ваги мережі стають здатні визначити кластери. Ваги організуються так, що топологічно близькі вузли реагують на схожі вхідні сигнали.

В результаті роботи алгоритму центр кластера встановлюється в певній позиції, яка задовольняє кластеризовані приклади, для яких даний нейрон є «переможцем». В результаті навчання мережі необхідно визначити міру сусідства нейронів, тобто коло нейрона-переможця, який представляє кілька нейронів, що оточують нейрон-переможець.

Для реалізації алгоритму необхідно визначити міру сусідства нейронів (знаходження нейрона-переможця). На рис 6.8 показані зони топологічного сусідства нейронів на мапі ознак у різні моменти часу. NEj(t) – множина нейронів, що вважаються сусідами нейрона j у момент часу t. Зони сусідства зменшуються з часом.

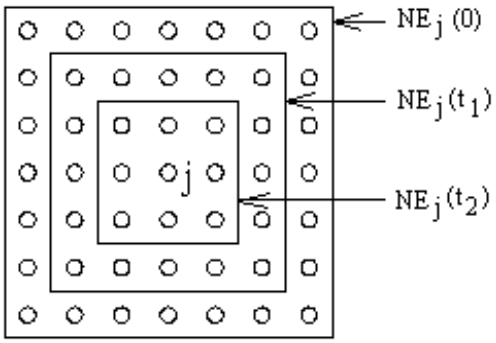


Рисунок 6.8 – Зони топологічного сусідства на карті ознак у різні моменти часу

Спочатку до кола належить велике число нейронів, далі його розмір поступово зменшується. Мережа формує топологічну структуру, в якій схожі приклади утворюють групи прикладів, які близько знаходяться на топологічній мапі.

Мережа Хопфілда. Джон Хопфілд вперше представив свою асоціативну мережу у 1982 р. у Національній Академії Наук. На честь Хопфілда та нового підходу до моделювання, ця мережна парадигма згадується як мережа Хопфілда. Мережа базується на аналогії фізики динамічних систем. Початкові застосування мережі включали асоціативну, або адресовану за змістом пам'ять та розв'язування задач оптимізації [15].

Мережа Хопфілда використовує три прошарки: вхідний, прошарок Хопфілда та вихідний прошарок. Кожен прошарок має однакову кількість нейронів. Виходи нейронів вхідного прошарку надходять до входів відповідних нейронів прошарку Хопфілда. Тут, зв'язки мають фіксовані вагові коефіцієнти. Виходи прошарку Хопфілда під'єднуються до входів всіх нейронів прошарку Хопфілда, за винятком самого себе, а також до відповідних елементів у вихідному прошарку. Під час навчання, мережа скеровує дані з вхідного прошарку до прошарку Хопфілда. Прошарок Хопфілда коливається, поки не

буде завершена певна кількість циклів, і змінений стан сигналів нейронів прошарку передається на вихідний прошарок. Цей стан відповідає образу, який буде запам’ятовано в мережі.

Навчання мережі Хопфілда вимагає, щоб навчальний образ був представлений на вхідному та вихідному прошарках одночасно. Рекурсивний характер прошарку Хопфілда забезпечує засоби корекції всіх ваг з’єднань. Для правильного навчання мережі відповідні пари «вхід-вихід» мають відрізнятися між собою.

Якщо мережа Хопфілда використовується як пам’ять, що адресується за змістом вона має два головних обмеження.

Число образів, що можна зберегти та точно відтворити є строго обмеженим. Якщо зберігається занадто багато образів, мережа може збігтись до нового образу, що не існує у відмінному від всіх запрограмованих образів, або не збігтись взагалі. Межа місткості пам’яті для мережі приблизно 15% від числа нейронів у прошарку Хопфілда.

Якщо навчальні приклади є занадто подібними, прошарок Хопфілда може стати нестабільним. Зразок образу вважається нестабільним, якщо він застосовується за нульовий час і мережа збігається до деякого іншого образу з навчальної множини. Ця проблема може бути вирішена вибором навчальних прикладів, більш ортогональних між собою.

Структурна схема мережі Хопфілда приведена на рис 6.9.

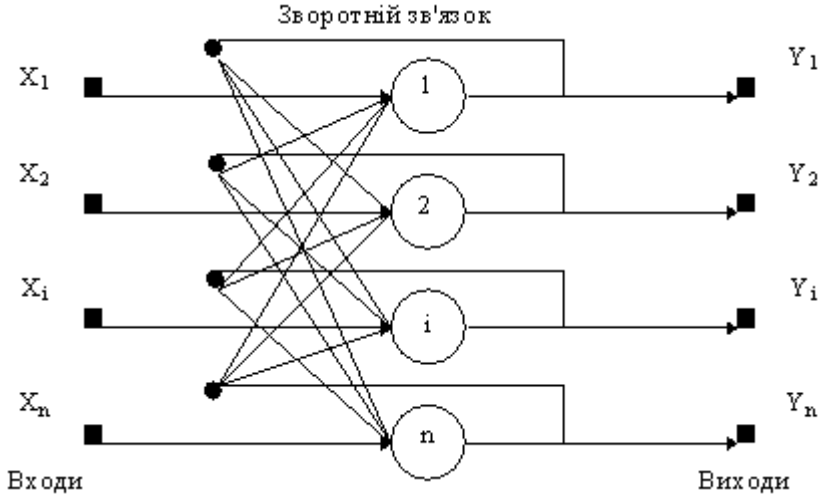


Рисунок 6.9 – Структурна схема мережі Хопфілда

Для вирішення завдання асоціативної пам’яті є деякий набір двійкових сигналів (зображень, звукових оцифрованих, інших даних, що описують об’єкти або характеристики процесів), який вважається зразковим. Мережа повинна вміти з зашумленого сигналу, поданого на її вхід, виділити («пригадати» за частковою інформацією) відповідний зразок або «дати висновок» про те, що вхідні дані не відповідають жодному зі зразків.

Іноді мережа не може провести розпізнавання і видає на виході образ, що не існує. Це пов’язано з проблемою обмеженості можливостей мережі. Для мережі Хопфилда число збережених образів m не повинно перевищувати 0.15*n (n – кількість нейронів вихідного прошарку). Крім того, якщо два образи А і Б сильно схожі, вони, можливо, будуть викликати в мережі перехресні асоціації, тобто пред’явлення на входи мережі вектора А призведе до появи на її виходах вектори Б і навпаки.

Мережа Хемінга. Мережа Хемінга (Hamming) є розширенням мережі Хопфілда. Ця мережа була розроблена Річардом Ліппманом (Richard Lippman) у середині 80-х рр. Мережа Хемінга реалізує класифікатор, що базується на найменшій похибці для векторів двійкових входів, де похибка визначається відстанню Хемінга. Відстань Хемінга визначається як число бітів, які відрізняються між двома відповідними вхідними векторами фіксованої довжини. Один вхідний вектор є незашумленим прикладом образу, інший є спотвореним образом. Вектор виходів навчальної множини є вектором класів, до яких належать образи. У режимі навчання вхідні вектори розподіляються до категорій для яких відстань між зразковими вхідними векторами та зміненим вхідним вектором є мінімальною.

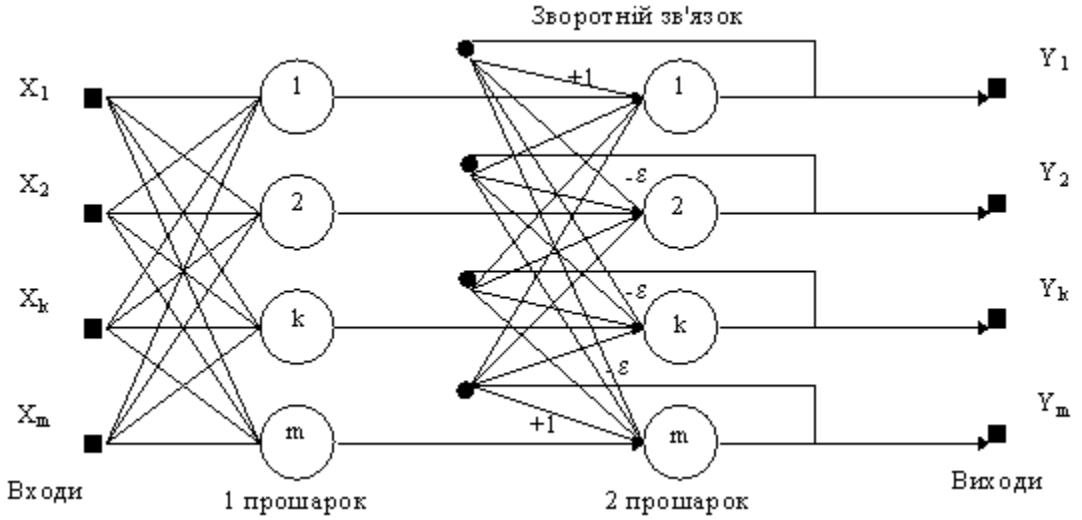


Рисунок 6.10 – Робота мережі Хемінга

Мережа Хемінга має три прошарки: вхідний прошарок з кількістю вузлів, скільки є окремих двійкових ознак; прошарок категорій (прошарок Хопфілда), з кількістю вузлів, скільки є категорій або класів; вихідний прошарок, який відповідає числу вузлів у прошарку категорій (рис 6.10).

Мережа є простою архітектурою прямого поширення з вхідним рівнем, повністю під’єднаним до прошарку категорій. Кожен нейрон у прошарку категорій є зворотно під’єднаним до кожного нейрона у тому ж самому прошарку і прямо під’єднаним до вихідного нейрону. Вихід з прошарку категорій до вихідного прошарку формується через конкуренцію [16].

Навчання мережі Хемінга є подібним до методології Хопфілда. На вхідний прошарок надходить бажаний навчальний образ, а на виході вихідного прошарку надходить значення бажаного класу, до якого належить вектор. Вихід містить лише значення класу до якого належить вхідний вектор. Рекурсивний характер прошарку Хопфілда забезпечує засоби корекції всіх ваг з'єднань.

🔔 *Дослідницьке завдання:*

1) Порівняти продуктивність різних архітектур CNN, таких як VGG, ResNet, Inception та EfficientNet, у задачах класифікації зображень.
2) Дослідити ефективність використання згорткових нейронних мереж у задачах сегментації зображень, зокрема архітектури U-Net, SegNet та Mask R-CNN. Провести порівняльні експерименти з різними наборами даних (медичні зображення, супутникові знімки).

❓ *Контрольні питання:*

1) Надати визначення: згорткові нейронні мережі. Для чого згорткові нейронні мережі використовуються у машинному навчанні?
2) Які основні компоненти згорткової нейромережі і як вони взаємодіють між собою?
3) Які переваги згорткових нейромереж порівняно з повнозв'язаними нейромережами для обробки зображень?
4) Назвати основні типи шарів у згорткових нейромережах і їх функції.
5) Як працює згортковий шар і яке його призначення в архітектурі CNN?

ЗМІСТОВИЙ МОДУЛЬ 5. УЗАГАЛЬНЕННЯ РІШЕНЬ В ЗАДАЧАХ МАШИННОГО НАВЧАННЯ. СКАЛЯРИ

Тема 7. Узагальнення рішень в задачах машинного навчання. Скаляри. Подання даних для нейронних мереж

7.1 Скаляри. Визначення задачі, створення набору даних

Як було сказано вище дані, те що зберігаються в багатовимірних масивах Numpy, називають також тензорами. Всі сучасні системи машинного навчання використовують тензори, як основну структуру даних. Тензори є фундаментальною структурою даних – настільки фундаментальної, що це відбилося на назві бібліотеки Google TensorFlow. Отже, що ж таке тензор?

Фактично тензор – це контейнер для даних, практично завжди числових. Іншими словами, це контейнер для чисел. Наприклад, матриці є двовимірними тензорами: тензори – це узагальнення матриць з будь-якою кількістю вимірювань (слід звернути увагу, що в термінології тензорів виміру часто називають осями).

Скаляри (тензори нульового рангу). Тензор, що містить єдине число, називається скаляром (скалярним, або тензором нульового рангу). У Numpy число типу float32 або float64 – це скалярний тензор (або скалярний масив). Визначити кількість осей тензора Numpy можна за допомогою атрибута ndim; скалярний тензор має 0 осей (ndim == 0). Кількість осей тензора також називають його рангом. Приклад скаляра в Numpy:

```
>>> import numpy as np
>>> x = np.array (12)
>>> x
array (12)
>>> x.ndim
0
```

Вектори (тензори першого рангу). Одновимірний масив чисел називають вектором, або тензором першого рангу. Тензор першого рангу має єдину вісь. Приклад вектору в Numpy:

```
>>> x = np.array ([12, 3, 6, 14])
>>> x
array ([12, 3, 6, 14])
>>> x.ndim

1
```

Цей вектор містить п’ять елементів і тому називається п’ятимірним вектором. Не плутати п’ятимірні вектори з п’ятимірними тензорами! П’ятимірний вектор має тільки одну вісь і п’ять значень на цій осі, тоді як п’ятимірний тензор має п’ять осей (і може мати будь-яку кількість значень на

кожній з них). Мірність може позначати або кількість елементів на даній осі (як у випадку з п’ятимірним вектором), або кількість осей в тензорі (як в п’ятимірному тензорі), що іноді може викликати плутанину. В останньому випадку технічно більш коректно говорити про тензор п’ятого рангу (ранг тензора збігається з кількістю осей), але для тензорів використовується неоднозначне позначення: п’ятимірний тензор.

Масив векторів – це матриця, або двовимірний тензор. Матриця має дві осі (часто їх називають рядками й стовпцями). Матрицю можна уявити як прямокутну таблицю з числами. Приклад матриці в Numpy:

```
>>> x = np.array ([[5, 78, 2, 34, 0],
[6, 79, 3, 35, 1],
[7, 80, 4, 36, 2]])
>>> x.ndim
2
```

Елементи на першій осі називають рядками, а на другій – стовпцями. У попередньому прикладі [5, 78, 2, 34, 0] – це перший рядок матриці x, а [5, 6, 7] – її перший стовпець.

Якщо упакувати такі матриці в новий масив, вийде тривимірний тензор, який можна уявити як числовий куб. Нижче наводиться приклад тривимірного тензора в Numpy:

```
>>> x = np.array ([[[5, 78, 2, 34, 0],
                    [6, 79, 3, 35, 1],
                    [7, 80, 4, 36, 2]],
                   [[5, 78, 2, 34, 0],
                    [6, 79, 3, 35, 1],
                    [7, 80, 4, 36, 2]],
                   [[5, 78, 2, 34, 0],
                    [6, 79, 3, 35, 1],
                    [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

Упакувавши тривимірні тензори в масив, отримаємо чотиривимірний тензор і т.д. У глибокому навчанні найчастіше використовуються тензори від нульового рангу до чотиривимірних, але іноді, наприклад при обробці відеоданих, справа може дійти й до п’ятимірних тензорів.

7.2 Подання даних для нейронних мереж

Подання (або представлення) для нейронних мереж, також відоме як інженерія ознак (feature engineering), є важливим етапом в процесі побудови моделей машинного навчання. Від того, як дані представлені для навчання, значною мірою залежить ефективність і точність роботи нейронної мережі.

Прикладом подання є зображення, які повинні бути нормалізовані, а їх розміри приведені до єдиного формату перед передачею в нейронну мережу (Для ознайомлення з поданням зображень можна використати MNIST).

Подання даних – це критично важливий етап, що впливає на продуктивність моделі та якість передбачень [17].

Основними аспекти подання даних можна вважати:

– **числові подання** (масштабовані та нормалізовані значення, числові дані можуть бути нормалізовані (приведені до діапазону [0, 1]) або стандартизовані (мають середнє значення 0 і стандартне відхилення 1));

– **логарифмічні перетворення** (використовуються для обробки даних з великим діапазоном значень, таких як фінансові показники);

– **категоріальні подання** (One-hot кодування, кожне значення категоріальної змінної представлено бінарним вектором);

– **кодування цілочисельними значеннями** (кожне категоріальне значення представлено унікальним цілим числом);

– **текстові подання** (токенізація, де розбиття тексту на окремі слова або токени; Bag-of-Words (BoW), де представлення тексту у вигляді частотного словника; TF-IDF (Term Frequency-Inverse Document Frequency), тут важливість слова в документі залежить від частоти його використання в інших документах; Word embeddings: представлення слів у вигляді векторів фіксованої розмірності (наприклад, word2vec, GloVe, FastText); Contextual embeddings – представлення слів з урахуванням контексту (наприклад, BERT, GPT));

– **зображення** (сирі піксельні значення, де представлення зображення у вигляді масиву піксельних значень; нормалізовані значення, де піксельні значення можуть бути нормалізовані (діапазон [0, 1]); аугментовані зображення, тут використання методів аугментації для збільшення різноманітності даних (обертання, масштабування, зміна яскравості тощо));

– **часові ряди** (неправильні ознаки, де використання попередніх значень як ознак; рухомі середні, де використання середніх значень за певний період часу; Фур'є- або вейвлет-перетворення, тут представлення даних у частотному просторі);

– **графові подання** (аджасентні матриці, де представлення графів у вигляді матриць суміжності; Node embeddings, тут векторні представлення вузлів графу (наприклад, Node2Vec, GraphSAGE)).

Ці подання можуть бути комбіновані або адаптовані залежно від конкретної задачі та типу даних. Важливо правильно обрати метод подання, щоб нейронна мережа могла ефективно навчатися та робити точні передбачення. Характеристика декількох з них:

Векторні дані. Найбільш часто відома форма даних. У таких наборах кожен зразок може бути представлений вектором, а пакет, відповідно, двовимірним тензором (тобто масивом векторів), де перша вісь – це вісь зразків, а друга – вісь ознак.

Розглянемо два приклади.

– Актуарний набір даних з інформацією про людей, де для кожної людини вказуються вік, поштовий індекс і дохід. Кожна людина характеризується вектором з трьома значеннями, відповідно, весь набір даних,

що описує 100 000 людей, можна зберегти у двовимірному тензорі з формою (100000, 3).

– Колекція текстових документів, де кожен документ представлений кількістю повторень кожного слова (зі словника з 20 000 найбільш застосованих слів). Кожен документ можна представити як вектор з 20 000 значень (по одному лічильнику на кожне слово зі словника), відповідно, весь набір даних, що описує 500 документів, можна зберегти у двовимірному тензорі з формою (500, 20000).

Тимчасові ряди або послідовності. Всякий раз, коли час (або поняття послідовної впорядкованості) грає важливу роль в ваших даних, такі дані переважно зберігати в тривимірному тензор з явною віссю часу. Кожен зразок може бути представлений як послідовність векторів (двовимірних тензорів), а сам пакет даних – як тривимірний тензор (рис 7.1).

Рисунок 7.1 – Тривимірний тензор з тимчасовим поруч

Відповідно до угод, вісь часу завжди є другою віссю (віссю з індексом 1). Розглянемо кілька прикладів.

1. Набір даних з цінами акцій. Кожну хвилину зберігається поточна ціна акцій, а також найбільша і найменша ціни за минулу хвилину. Тобто кожна хвилина представлена тривимірним вектором, весь торговий день – двовимірним тензором з формою (390, 3) (де 390 – тривалість торгового дня у хвилинах), а дані за 250 днів – тривимірним тензором з формою (250, 390, 3). В такому випадку кожен зразок представляє дані за один торговий день.
2. Набір даних з твітами, де кожен твіт кодується послідовністю із 280 символів з алфавіту зі 128 унікальними символами. В такому випадку кожен символ можна закодувати як двійковий вектор зі 128 елементами (містить нулі у всіх елементах, крім елемента з індексом, відповідним номеру символу в алфавіті, в який записується 1). При такій організації кожен твіт можна уявити як двовимірний тензор з формою (280, 128), а набір з мільйоном твітів – як тензор з формою (1000000, 280, 128).

Зображення. Зазвичай зображення мають три виміри: висоту, ширину і колір. Навіть при тому, що чорно-білі зображення (як в наборі даних MNIST) мають тільки один канал кольору і могли б зберігатися у двовимірних тензорах, за умовами тензори з зображеннями завжди мають три виміри, де для чорно-білих зображень відводиться тільки один канал кольору. Відповідно, пакет зі

128 чорно-білими зображеннями, що мають розмір 256×256, можна зберегти в тензор з формою (128, 256, 256, 1), а пакет зі 128 кольоровими зображеннями – в тензор з формою (128, 256, 256, 3) (рис 7.2).

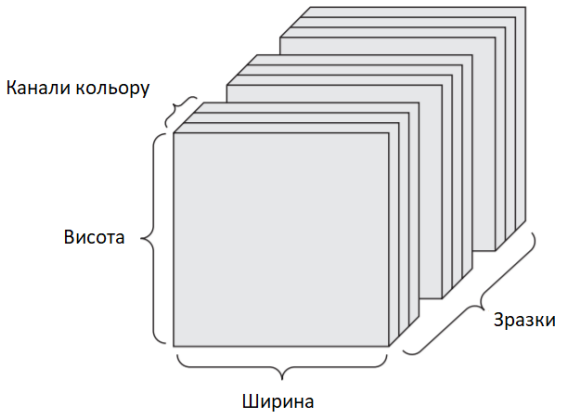


Рисунок 7.2 – Чотиривимірний тензор з зображеннями

Щодо форм тензорів з зображеннями існує дві угоди: угода канал слідує останнім (використовується в TensorFlow) і угода канал слідує першим (використовується в Theano). Фреймворк машинного навчання TensorFlow, розроблений компанією Google, відводить для кольору останню вісь: (зразки, висота, ширина, колір). А бібліотека Theano відводить для кольору вісь, наступну відразу за віссю пакетів: (зразки, колір, висота, ширина). Якщо дотримуватися умови, прийнятою в Theano, попередні приклади тензорів матимуть форму (128, 1, 256, 256) і (128, 3, 256, 256). Фреймворк Keras підтримує обидва формати.

Відео. Відео дані – один з небагатьох типів даних, для зберігання яких потрібні п’ятимірні тензори. Відео можна уявити як послідовність кадрів, де кожен кадр – кольорове зображення. Кожен кадр можна зберегти в тривимірному тензорі (висота, ширина, колір), відповідно, їх послідовність можна зберегти у чотиривимірному тензорі (кадри, висота, ширина, колір), а пакет різних відеороликів – у п’ятимірному тензорі з формою (зразки, кадри, висота, ширина, колір).

Наприклад, 60-секундний відеокліп з роздільною здатністю 144×256 і частотою 4 кадри в секунду буде складатися з 240 кадрів. Для збереження пакета з чотирьох таких кліпів потрібно тензор з формою (4, 240, 144, 256, 3). Тобто 106 168 320 значень! Якщо припустити, що dtype тензора визначено як float32, тоді для зберігання кожного значення знадобиться 32 біти, тобто для зберігання всього тензора – 405 Мбайт.

Дослідницьке завдання:

1) Розглянути методи створення та розширення наборів даних для задач прогнозування скалярних величин, таких як температура, фінансові індекси або час до відмови обладнання.

2) Розробити методику вибору міри успіху (наприклад, середня абсолютна похибка, середнє квадратичне відхилення) та протоколу оцінки (крос-валідація, відкладений тестовий набір) для задачі прогнозування скалярів.

? *Контрольні питання:*

1) Поняття скаляру в математиці та машинному навчанні?

2) Які основні властивості скалярів і для чого вони використовуються в обчисленнях моделей машинного навчання?

3) Як визначається задача в машинному навчанні і чому чітка формулювання задачі є важливою?

4) Які критерії вибору набору даних для задачі машинного навчання?

5) Які основні етапи включає процес розробки моделі машинного навчання?

ЗМІСТОВИЙ МОДУЛЬ 6. ЛОГІЧНЕ ВИВЕДЕННЯ, ПРЯМИЙ І ЗВОРОТНИЙ МЕТОДИ НЕЧІТКОГО ВИСНОВКУ ТА ПОШИРЕННЯ

Тема 8. Логічне виведення, прямий і зворотний методи нечіткого висновку

8.1 Прямий і зворотний методи нечіткого висновку та поширення

Прямий метод нечіткого висновку (forward chaining) є методом логічного виведення, який починається з початкових фактів і поступово застосовує правила для виведення нових фактів, доки не буде досягнута мета. Даний метод використовує всі доступні дані для формування ланцюжка логічних висновків. При цьому система починає з набору відомих фактів і перевіряє, чи є ці факти частиною передумов правил. Якщо передумови правила задовольняються, то виконується дія цього правила, що призводить до появи нового факту. Цей новий факт додається до бази знань і використовується для подальшого застосування правил.

Процес повторюється доти, доки не буде досягнута задана мета або не залишиться правил для застосування. Прямий метод часто використовується в системах, де важливо поступово нарощувати знання на основі поточних фактів. Це особливо корисно в реальному часі, де нові дані можуть постійно надходити, і система повинна оперативно реагувати на них.

Цей метод широко застосовується в експертних системах, системах підтримки прийняття рішень та інших галузях, де необхідно автоматизувати процес виведення нових знань з наявних даних. Основна перевага прямого методу полягає в його здатності працювати в режимі реального часу й обробляти великі обсяги даних, поступово уточнюючи результати на основі нових фактів.

Використання **зворотного методу** нечіткого висновку (backward chaining) починається з визначеної мети й працює у зворотному напрямку, щоб з’ясувати, які умови повинні бути виконані для досягнення цієї мети.

У цьому методі процес розпочинається з перевірки кінцевого результату або мети, яку ми хочемо досягти. Система визначає, які правила можуть привести до цього результату, і перевіряє передумови цих правил. Якщо передумови не виконуються, система розбиває їх на під цілі й починає працювати над ними.

Наприклад, якщо метою є певний стан системи, то система шукає правила, які можуть призвести до цього стану, і перевіряє, які умови повинні бути виконані для застосування цих правил. Кожна з умов стає новою метою, і процес повторюється доти, доки не будуть знайдені відомі факти, які задовольняють усі необхідні умови.

Даний підхід часто використовується в експертних системах для діагностики або розв’язання складних задач, де кінцева мета відома, але шлях

до неї може бути невизначеним. Зворотний метод дозволяє системі сконцентруватися на кінцевій меті й уникнути непотрібних обчислень, працюючи тільки з тими правилами й фактами, які безпосередньо стосуються досягнення мети.

Основна перевага зворотного методу полягає в його ефективності, оскільки він зосереджується лише на тих аспектах, які необхідні для досягнення конкретної мети. Це дозволяє зменшити обсяг обчислень і зосередитися на найважливіших частинах проблеми, що робить його корисним в ситуаціях, де важливо швидко знайти рішення для складних задач з багатьма можливими варіантами розвитку подій.

Основні етапи нечіткого логічного висновку:

– **фазифікація (Fuzzification).** Це перший етап, на якому чіткі (числові) вхідні дані перетворюються на нечіткі множини. Це досягається за допомогою функцій належності, які визначають ступінь належності кожного вхідного значення до нечіткої множини.

– **застосування правил (Rule Application).** На даному етапі до нечітких вхідних даних застосовуються правила. Кожне правило має форму «Якщо X, то Y», де X і Y є нечіткими множинами. Вхідні дані порівнюються з передумовами правил, і обчислюється ступінь виконання кожного правила.

– **агрегація результатів (Aggregation).** Тут результати від усіх правил об’єднуються, щоб отримати єдиний нечіткий вихід. Це об’єднання може здійснюватися за допомогою різних методів, таких як мінімум, максимум або середнє значення.

– **дефазифікація (Defuzzification).** Це останній етап, на якому нечіткий вихід перетворюється на чітке (числове) значення. Існують різні методи дефазифікації, такі як центр ваги (centroid), середнього максимуму (mean of maxima) та інші.

При цьому основними алгоритмами виступають:

– **Метод Мамдані (Mamdani**). Це один з найбільш поширених методів нечіткого висновку. Використовується для систем керування і прийняття рішень. Він включає фазифікацію вхідних даних, застосування нечітких правил, агрегацію і дефазифікацію результатів. Метод Мамдані використовує функції належності для визначення ступеня виконання правил і об’єднує результати за допомогою операцій мінімуму або максимуму.

– **Метод Суге́но (Sugeno**). Метод схожий на Мамдані, але відрізняється тим, що виходи правил є функціями вхідних даних, а не нечіткими множинами. Це спрощує процес дефазифікації і робить його ефективнішим для реального часу. Метод Сугено часто використовується в системах, де потрібна висока швидкість обчислень.

– **Метод Цукермана (Tsukamoto)**. Даний метод менш поширений, але корисний у певних застосуваннях. У ньому кожне правило генерує нечітке вихідне значення, яке потім дефазифікується. Результати усіх правил об’єднуються за допомогою вагових коефіцієнтів.

Ці алгоритми нечіткого висновку дозволяють системам обробляти нечіткі дані й приймати рішення в умовах невизначеності, що робить їх дуже корисними для різноманітних застосувань, включаючи системи керування, експертні системи та інші галузі.

8.2 Механізм нейронних мереж: оптимізація на основі градієнта

Градієнт – це похідна операції з тензором, узагальнення поняття похідної на функції з багатовимірними вхідними даними, тобто на функції, які беруть на вході тензори.

Теоретично мінімум диференційної функції можна знайти аналітично. Як відомо, мінімум функції – це точка, де похідна дорівнює 0. Тобто залишається тільки знайти всі точки, де похідна доівнює в 0, і з'ясувати, в який з цих точок функція має найменше значення.

Стосовно до нейронних мереж це означає аналітичний пошук комбінації значень ваг, при яких функція втрат матиме найменше значення. Цього можна домогтися, розв'язавши рівняння gradient (f) (W)= 0 для W. Це поліноміальне рівняння з N змінними, де N – кількість ваг в мережі. Розв'язати рівняння для випадку N = 2 або N = 3 не складає труднощів, але перетворюється в практично нездійсненне завдання для нейронних мереж, в яких кількість параметрів рідко буває менше кількох тисяч і часто досягає декількох десятків мільйонів.

Тому на практиці використовується алгоритм з чотирьох кроків, представлений на початку цього розділу: можна потроху змінювати параметри, спираючись на поточні значення втрат у випадковому пакеті даних. Оскільки функція диференційована, можна обчислити її градієнт, який дозволяє ефективно реалізувати крок 4. Якщо ваги змінити в напрямку, протилежному градієнту, втрати з кожним циклом будуть потроху зменшуватися:

– Дістається пакет навчальних примірників x і відповідних цілей y.
– Мережа обробляє пакет x і отримує пакет передбачень.
– Обчислюються втрати мережі на пакеті, що дають оцінку розбіжності між крок 1 і 2.
– Обчислюється градієнт втрат для параметрів мережі (зворотний прохід).
– Параметри коригуються на невелику величину в напрямку, протилежному градієнту, наприклад W -= step*gradient, і тим самим знижуються втрати.

Термін «стохастичний» відображає той факт, що кожен пакет даних вибирається випадково (в науці слово «стохастичний» вважається синонімом слова «випадковий»). рис 8.1 ілюструє те, що відбувається на прикладі одновимірних даних, коли мережа має тільки один параметр.

Рисунок 8.1 – Градієнтний спуск вниз по одномірній кривій втрат
(один повчальний параметр)

На рис 8.2 зображений градієнтний спуск в одновимірному просторі параметрів, на практиці будемо використовувати градієнтний спуск в просторах з набагато більшою кількістю вимірів: кожен ваговий коефіцієнт в нейронної мережі – це незалежне вимірювання в просторі, і їх може бути десятки тисяч або навіть мільйони [18].

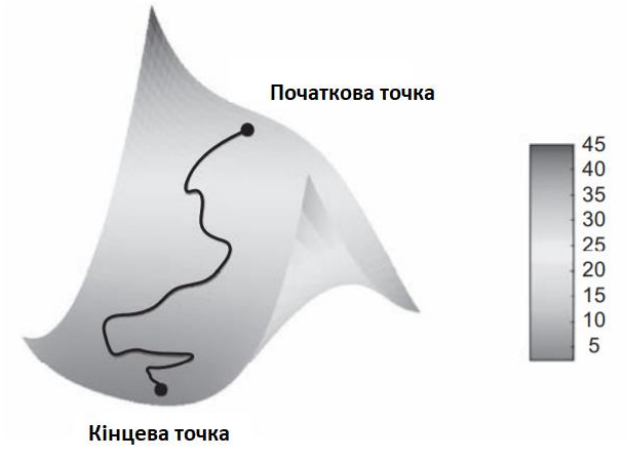


Рисунок 8.2 – Градієнтний спуск вниз по двовимірній поверхні витрат
(два повчальних параметри)

Існує також безліч варіантів стохастичного градієнтного спуску, які відрізняються тим, що при обчисленні наступних збільшень ваг беруть до уваги не тільки поточні значення градієнтів, а й попередні збільшення. Прикладами можуть служити такі алгоритми, як SGD з імпульсом, Adagrad, RMSProp і деякі інші. Ці варіанти відомі як методи оптимізації, або оптимізатори. Зокрема, уваги заслуговує ідея імпульсу, яка використовується в багатьох цих варіантах. Імпульс вводиться для розв’язання двох проблем SGD: невисокій швидкості збіжності й попадання в локальний мінімум.

🔔 *Дослідницьке завдання:*

1) Порівняти ефективність прямого і зворотного методів нечіткого логічного висновку у задачах прийняття рішень. Провести експерименти в різних доменах, наприклад, управління промисловими процесами або системи підтримки прийняття рішень, оцінити точність, швидкість та адаптивність методів.

2) Розробити та дослідити методи оптимізації нечітких логічних систем на основі градієнтного спуску.

? *Контрольні питання:*

1) Надати визначення прямого методу нечіткого висновку. Які етапи включає його виконання?

2) Які основні операції проводяться на кожному етапі прямого методу нечіткого висновку?

3) Надати визначення зворотних методів нечіткого висновку. Як вони відрізняються від прямих методів?

4) Як відбувається процес зворотного нечіткого висновку в системах інтелектуальних та експертних систем?

5) Які основні алгоритми оптимізації на основі градієнту використовуються в машинному навчанні?

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Калюжняк А. В. Застосування штучного інтелекту в прогнозуванні економічної моделі. *Artificial Intelligence*. №28(2), 2023. С. 88-93.
2. Надригайло Т. Ж., Молчанова К. А. Аналіз нейронних алгоритмів. *Математичне моделювання* : електрон. наук. фахове вид. 2015.С. 46-52. URL: http://www.dstu.dp.ua/Portal/Data/74/68/13-st13.pdf (дата звернення 15.05.2024).
3. Avramenko V. V., Kalashnykova N. I., Kalashnikov V. V., Watada J. Derivative of disproportion functions for pattern recognition. *In Unconventional Methods for Geoscience, Shale Gas and Petroleum in the 21st Century*. 2023. 125 p. DOI: https://doi.org/10.3233/AERD230022
4. Avramenko V., Demianenko V. Cryptosystem based on a key function of a real variable. *Florida Journal of Development.* 2021. Vol. 2, No. 2. P. 2576–2591. https://doi.org/10.46932/sfjdv2n2-113
5. Bickley S. J., Chan H. F., Torgler B. Artificial intelligence in the field of Economic. *Scientometrics.* 2022. Vol. 127. P. 2055–2084.
6. Gao S., Dong P.,. Pan Z., Li G. Y. Deep learning based channelestimation for massive mimo with mixed-resolution adcs, *IEEE Communications Letters*. 2019.Vol. 23, No. 11. P. 1989–1993.
7. Kalashnikov V. V., Avramenko V. V., Slipushko N. Y., Kalashnykova N. I., Konoplyanchenko A. E. Identification of Quasi-Stationary Dynamic Objects with the Use of Derivative Disproportion Functions. *Procedia Computer Science*. 2017. Vol. 107. P. 50-108. DOI: https://doi.org/10.1016/j.procs.2017.05.266
8. Kalashnykova N., Avramenko V., Kalashnikov V. Sums of Key Functions Generating Cryptosystems. *Computational Science.* 2019. P. 293–302. DOI: https://doi.org/10.1007/978-3-030-22750-0_23
9. Kato N., Fadlullah Z. Md., Tang F., Mao B., Tani S., Okamura A., Liu J. Optimizing space-air-ground integrated networks by artificial intelligence. *IEEE Wireless Commun.* 2019. 8 p.
10. Khaled B., Letaief Wei Chen, Yuanming Shi, Jun Zhang, Ying-Jun Angela Zhang. The Roadmap to 6G. *AI Empowered Wireless Networks*. 2019. 7 p.
11. Lyan A. How AI is regulated in Ukraine and the world. *AIN.UA.* URL:https://ain.ua/2022/02/01/yak-ai-rehulyuyut-v-ukrayini-ta-sviti/ (дата звернення 22.06.2024).
12. MS Usha Yadav Nift, Jodhpur. The Role of Artificial Intelligence in Telecommunications Industry: Aastha Arora & Neelesh Verma, 2019. 49-56 pp.
13. Ostrianska Y. V., Yesina M. V., Gorbenko I. D. Analysis of views of the European Union on quantum-post-quantum limitations. *Radiotekhnika.* 2022. 210 p. DOI: https://doi.org/10.30837/rt.2022.3.210.06
14. Soltani M., Pourahmadi V., Mirzaei A., Sheikhzadeh H. Deeplearning based channel estimation. *IEEE Communications Letters*. 2019. Vol. 23, No. 4. P. 652–655.

15. Stashkevych O. The impact of technology and artificial intelligence on the labor market in Ukraine. *InterConf.* 2021. № 81. 25-30 с.

16. The Importance of Artificial Intelligence and Data for the Telecommunications Industry and the FCC/ Report of the FCC's Technological Advisory Council Working Group on Artificial Intelligence and Computing/ (FCC TAC AIWG), 2021. 43-47 pp.

17. Tommy van der Vorst. Nick Jelicic. Jan van Rees. Managing AI use in telecom infrastructures. *Advice to the supervisory body on establishing riskbased AI supervision.* Dutch Radiocommunications Agency, 2020. 21-53 pp.

18. Vyshnyakova O. AI and education: how artificial intelligence will affect school education. *LB.ua.* URL: https://lb.ua/blog/olena_vyshniakova/547626_ai_osvita_yak_shtuchniy_intelekt.html (дата звернення 19.05.2024).

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Величко О. М., Гордієнко Т. Б. Інтелектуальні інформаційні системи: структура і застосування. Херсон : ОЛДІ+, 2022. 728 с.
2. Єремєєв І. С., Гуйда О. Г. Інтелектуальні системи підготовки рішень. Одеса : Видавничий дім «Гельветика», 2021. 376 с.
3. Литвин В. В., Пасічник В. В., Яцишин Ю. В. Інтелектуальні системи. Львів : Видавництво «Новий Світ – 2000», 2020. 405 с.
4. Adaptive learning in the system of higher education. *Scientific progress.* 2021. No. 3 (1). P. 88-97.
5. Bickley S. J., Chan H. F., Torgler B. Artificial intelligence in the field of Economic. *Scientometrics*. 2022. Vol. 127. P. 2055–2084.
6. Cockburn I. M., Henderson R., Stern S. The Impact of Artificial Intelligence on Innovation. *The Economic of Artificial Intelligence: An Agenda*. Chicago : University of Chicago Press, 2019, P. 115–146.
7. Ghoddusi H., Creamer G. G., Rafizadeh N. Machine learning in energy Economic and finance: A review. *Energy Economic*. 2019. Vol. 81. P. 709–727.
8. Mahjourian R. Neuroevolutionary Planning for Robotic Control : Doctoral Dissertation Proposal. The University of Texas at Austin Austin, 2018. 44p.
9. Rasulova Data augmentation. TensorFlow Developers. (2023). TensorFlow (v2.14.0-rc0). Zenodo. Available: https://doi.org/10.5281/zenodo.8256979.
10. Russell S., Norvig P. Artificial intelligence: a modern approach. *4th Edn. Hoboken*, NJ: Pearson, 2021. 1115 p.
11. Xiong Z.; Zhang X.; Hu Q., Han H. FormerFusion: Cross-Domain Frequency Information Learning for Infrared and Visible Image Fusion Based on the Inception Transformer. *Remote Sens.* 2023. Vol. 15, 1352 p.

Додаток А

Навчання нейронної мережі на основі алгоритму градієнтного спуску

```
    def SGD( # Стохастичний граєнтний спуск
          self              #
        , training_data     # навчальна вибірка
        , epochs            # кількість епох навчання
        , mini_batch_size  # розмір підвибірки
        , eta               # швидкість навчання
        , test_data         # тестуюча вибірка
        ):
        test_data = list(test_data) # створюємо список об'єктів тестуючої
        вибірки
        n_test = len(test_data) # обчислюємо довжина тестуючої вибірки
        training_data = list(training_data) # створюємо список об'ктів
навчальної вибірки
        n = len(training_data) # обчислюємо розмір навчальної вибірки
        for j in range(epochs): # цикл по епохам
            random.shuffle(training_data) # перемішуємо елементи вибірки
            mini_batches = [training_data[k:k+mini_batch_size] for k in
range(0, n, mini_batch_size)] # створюємо підвибірки
            for mini_batch in mini_batches: # цикл по підвибіркам
                self.update_mini_batch(mini_batch, eta) # один крок
градієнтного спуску
            print ("Epoch {0}: {1} / {2}".format(j,
self.evaluate(test_data), n_test)) # дивимося прогрес в навчанні
```

Цей програмний код працює таким чином. На початку кожної епохи навчання елементи навчальної вибірки перемішуються (переставляються у випадковому порядку) за допомогою функції shuffle() із випадкової бібліотеки, після чого навчальна вибірка послідовно розбивається на підвибірки довжини mini_batch_size.

Для кожної підвибірки виконується один крок градієнтного спуску за допомогою методу update_mini_batch. Після того, як виконано останній крок градієнтного спуску, виконується метод update_mini_batch для останньої підвибірки, на екрані якого виводиться досягнутий прогрес в навчанні нейронної мережі, що вираховується на тестовому виборі за допомогою методу evaluate (див. нижче).

Аналізуючи програмний код методу update_mini_batch, можна побачити, що основна частина вираховується в результаті виклику методу backprop (див. нижче). Цей метод класу мережі реалізує алгоритм зворотного поширення помилок, який є швидким способом вирахування градієнта стоїчкової функції. Таким чином, метод update_mini_batch вираховує всі градієнти для кожного прецеденту ($\vec{x}$, y) у підвибірці, а також оновлює масу та зміщення нейронної мережі. Обчислимо метод update_mini_batch в розділ опису класу. Мережа:

```
    def update_mini_batch( # крок градієнтного спуску
         self              # вказівник на об'єкт класу
         , mini_batch      # підвибірка
         , eta             # швидкість навчання
         ):
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        for x, y in mini_batch:
            delta_nabla_b, delta_nabla_w = self.backprop(x, y)
            nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
            nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
        self.weights = [w-(eta/len(mini_batch))*nw
                        for w, nw in zip(self.weights, nabla_w)]
        self.biases = [b-(eta/len(mini_batch))*nb
                       for b, nb in zip(self.biases, nabla_b)]
```

```
    def backprop( # Алгоритм зворотнього
         розповсюдження
          self  # вказівник
         , x    # вектор вхідних сигналів
         , y    # очікуєви вектор вихідних сигналів
         ):
        nabla_b = [np.zeros(b.shape) for b in self.biases])
        nabla_w = [np.zeros(w.shape) for w in self.weights]

        # визначення змінних
        activation = x # вихідні сигнали шару
        activations = [x] #
        zs = [] #

        # пряме розповсюдження
        for b, w in zip(self.biases, self.weights):
            z = np.dot(w, activation)+b # рахуємо активаційні потенціали
            zs.append(z) # додаємо елемент
            activation = sigmoid(z) # рахуємо вихідні сигнали
            activations.append(activation) # додаємо елемент
```

```
        # зворотнє розповсюдження
        delta = self.cost_derivative(activations[-1], y) *
sigmoid_prime(zs[-1])
        nabla_b[-1] = delta # градиент dC/db для слоя L (BP3)
        nabla_w[-1] = np.dot(delta, activations[-2].transpose()) # градієнт
dC/dw для шару L (BP4)

        for l in range(2, self.num_layers):
            z = zs[-l]
            sp = sigmoid_prime(z) # рахуємо сигмоїдну функцію
            delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
            nabla_b[-l] = delta
            nabla_w[-l] = np.dot(delta, activations[-l-1].transpose
return (nabla_b, nabla_w)
```

```
    def evaluate(self, test_data): # Оцінка прогресу в навчанні
        test_results = [(np.argmax(self.feedforward(x)), y)
                        for (x, y) in test_data]
        return sum(int(x == y) for (x, y) in test_results)
```

Розпізнавання тексту написаного від руки за допомогою готових наборів даних MNIST

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Flatten, Dense, Dropout, Activation, BatchNormalization,
AveragePooling2D
from tensorflow.keras.optimizers import SGD, RMSprop, Adam
import tensorflow_datasets as tfds  # pip install tensorflow-
datasets
import tensorflow as tf
import logging
import numpy as np

tf.logging.set_verbosity(tf.logging.ERROR)
tf.get_logger().setLevel(logging.ERROR)

def mnist_make_model(image_w: int, image_h: int):
    # Модель нейронної мережі
    model = Sequential()
    model.add(Dense(1024, activation='relu',
input_shape=(image_w*image_h,)))
    model.add(Dropout(0.2))  #0.2 - встановити для 20% входів
до 0
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(10, activation='softmax'))
    model.compile(loss='categorical_crossentropy',
optimizer=RMSprop(),
                  metrics=['accuracy'])
    return model

def mnist_mlp_train(model):
    (x_train, y_train), (x_test, y_test) =
keras.datasets.mnist.load_data()
    # x_train: 60000x28x28 масив, x_test: 10000x28x28 масив
# масив x_train для навчання
```

```
# масив x_test для верифікації результатів
image_size = x_train.shape[1]
train_data = x_train.reshape(x_train.shape[0],
image_size*image_size)
test_data = x_test.reshape(x_test.shape[0],
image_size*image_size)
train_data = train_data.astype('float32')
test_data = test_data.astype('float32')
train_data /= 255.0
test_data /= 255.0
num_classes = 10
train_labels_cat = keras.utils.to_categorical(y_train,
num_classes)
test_labels_cat = keras.utils.to_categorical(y_test,
num_classes)
print("Training the network...")
# Початок навчання мережі
model.fit(train_data, train_labels_cat, epochs=8,
batch_size=64,
verbose=1, validation_data=(test_data, test_labels_cat))
# створюємо модель і навчаємо її
model = mnist_make_model(image_w=28, image_h=28)
mnist_mlp_train(model)
model.save('mlp_digits_28x28.h5')

# ф-ція розпізнавання картинки
def mlp_digits_predict(model, image_file):
image_size = 28
img = keras.preprocessing.image.load_img(image_file,
target_size=(image_size, image_size), color_mode='grayscale')
img_arr = np.expand_dims(img, axis=0)
img_arr = 1 - img_arr/255.0
img_arr = img_arr.reshape((1, image_size*image_size))
result = model.predict_classes([img_arr])
return result[0]

# передаємо картинки чисел, які потрібно розпізнати
model = tf.keras.models.load_model('mlp_digits_28x28.h5')
print(mlp_digits_predict(model, 'digit_1.png'))
print(mlp_digits_predict(model, 'digit_2.png'))
print(mlp_digits_predict(model, 'digit_3.png'))
print(mlp_digits_predict(model, 'digit_4.png'))
print(mlp_digits_predict(model, 'digit_2.png'))
```

Навчаємо нейромережу.

```
Training the network...
Train on 60000 samples, validate on 10000 samples
Epoch 1/8
60000/60000 [==============================] - 8s 131us/sample - loss: 0.2098 - acc: 0.9376 - val_loss: 0.1126 - val_acc: 0.9652
Epoch 2/8
60000/60000 [==============================] - 8s 132us/sample - loss: 0.0852 - acc: 0.9743 - val_loss: 0.0888 - val_acc: 0.9745
Epoch 5/8
60000/60000 [==============================] - 8s 130us/sample - loss: 0.0312 - acc: 0.9904 - val_loss: 0.0907 - val_acc: 0.9756
Epoch 6/8
60000/60000 [==============================] - 9s 145us/sample - loss: 0.0243 - acc: 0.9930 - val_loss: 0.0751 - val_acc: 0.9815
Epoch 7/8
60000/60000 [==============================] - 10s 159us/sample - loss: 0.0191 - acc: 0.9944 - val_loss: 0.0794 - val_acc: 0.9791
Epoch 8/8
60000/60000 [==============================] - 10s 159us/sample - loss: 0.0144 - acc: 0.9958 - val_loss: 0.0827 - val_acc: 0.9799
```

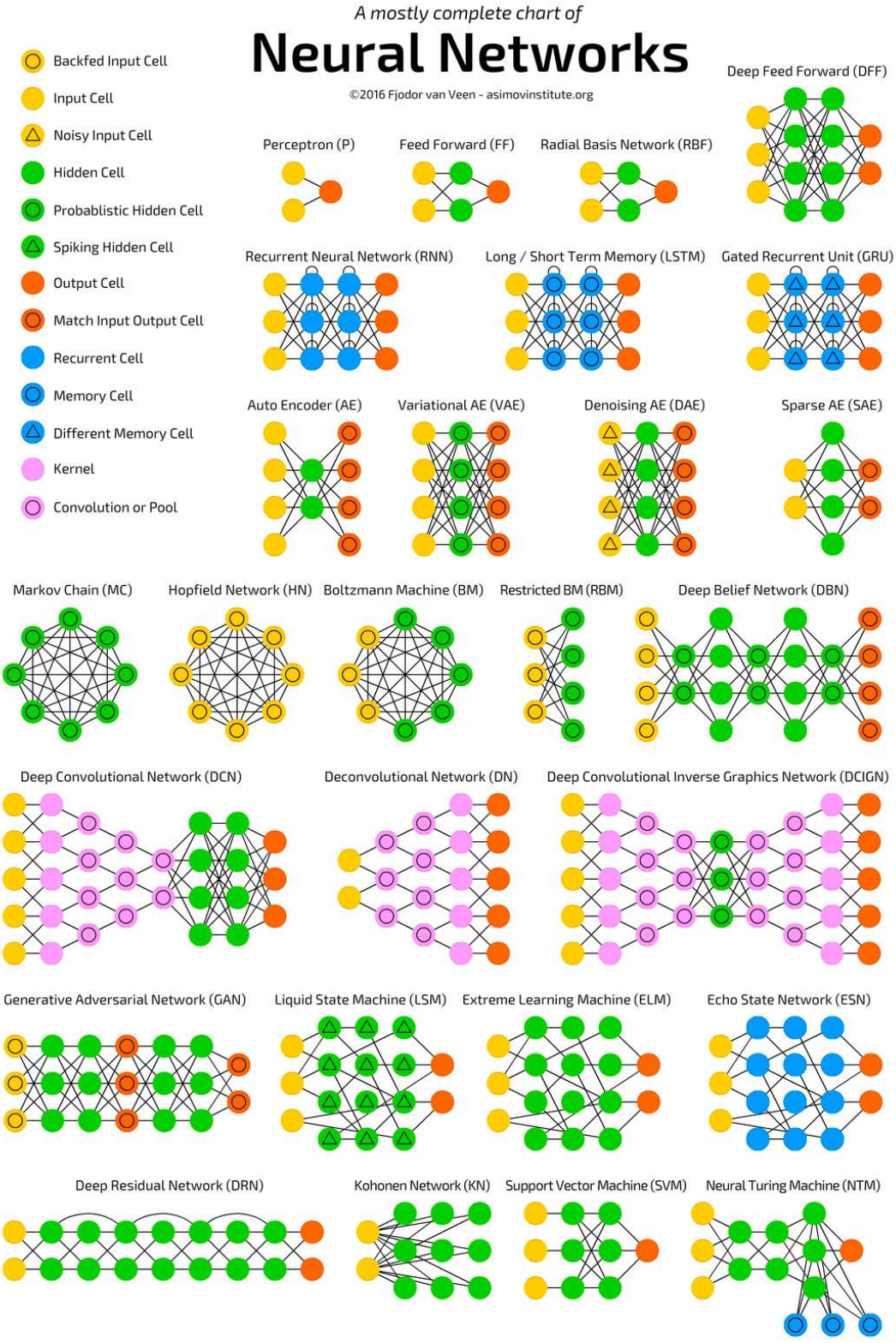
Далі даємо їй розпізнати цифри, які були створені у програмі Paint. Всього даємо 5 чисел.

Результат роботи нейромережі:

```
Epoch 8/8
60000/60000 [==============================] - 26s 433us/sample - loss: 0.0656 - acc: 0.9869 - val_loss: 0.1338 - val_acc: 0.9793
1
2
3
4
2
```

Додаток Б

Відомі алгоритми нейронних мереж



Навчальне видання
(українською мовою)

Калюжняк Анастасія Вікторівна

ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ СИСТЕМИ

Навчальний посібник
для здобувачів ступеня вищої освіти магістра
спеціальності «Комп’ютерні науки»
освітньо-професійної програми «Комп’ютерні науки»

Рецензент О. В. Кудін
Відповідальний за випуск Г. М. Шило
Коректор А. В. Калюжняк