УДК 004.62 https://doi.org/10.15407/pp2019.02.069

Б.О. Білецький

ГОРИЗОНТАЛЬНЕ ТА ВЕРТИКАЛЬНЕ МАСШТАБУВАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ

В роботі розглядаються основні етапи розв'язку задач машинного навчання (з учителем) розпізнаванню образів, а саме: управління навчальними вибірками, навчання, розпізнавання. Обговорюється вплив феномену великих даних (BigData) на кожен з етапів, а також методи ефективної організації обчислень на кожному з етапів при розв'язанні зазначених задач.

Ключові слова: методи машинного навчання, розпізнавання образів, горизонтальне масштабування, вертикальне масштабування, реляційні бази даних, ACID, NoSQL, CAP-теорема.

Вступ

Методи машинного навчання розпізнаванню образів останнім часом набули значної популярності. З одного боку цьому сприяє наявність предметних областей, що не підлягають ефективному описанню класичними алгоритмами та методами програмування. Це такі задачі як розпізнавання тону текстових повідомлень, виявлення аномалій у потоках даних, розпізнавання автомобільних номерних знаків, надання рекомендацій або відновлення складних функціональних залежностей. З іншого боку популярності методів машинного навчання сприяв феномен великих даних (Big Data), що тягнув за собою накопичення об'ємних навчаючих вибірок та використання їх у машинному навчанні. Адже складні задачі розпізнавання вимагають застосування більш складних моделей для розпізнавання (тобто таких моделей, що мають більше незалежних параметрів), а навчання таких моделей вимагає обробки більшої кількості прикладів у процесі навчання.

Відомо, що світові обсяги накопичених даних зростають експоненціально та, згідно оцінок, подвоюється кожні 20 місяців [1]. Натомість обчислювальна потужність комп'ютерів також зростає експоненціально згідно закону Мура [2], що був сформульований засновником компанії Intel у 1965 році, та зафіксував подвоєння швидкодії процесорів кожні 18 місяців. Такий баланс зростання обсягів даних та зростання обчислювальної потужності комп'ютерів дозволяв ефективно обробляти накопичені дані. Однак близько 2010 року стало зрозуміло, що закон Мура перестав виконуватися. Це сталося, зокрема, і через досягнення технологічних обмежень для поточної технології виробництва процесорів. У роботі [2] представлено дані щодо експоненціального зростання тактової частоти процесорів найбільших виробників, таких як: Intel, IBM, AMD, DEC, Sun з 1985 по 2010. З наведених даних видно, що близько 2010 року зростання досягло точки насичення та припинилося.

Для багатьох існуючих програмних систем, що були розраховані на обробку неперервно зростаючих масивів даних з новою актуальністю постало питання масштабування обчислень. До таких систем належать і системи машинного навчання розпізнаванню образів. Методи машинного навчання та розпізнавання складаються з кількох стадій, на які феномен великих даних (Big Data) впливає по різному. Серед важливих стадій машинного навчання розпізнаванню образів є: накопичення та зберігання навчаючих даних, навчання розпізнаванню та розпізнавання.

1. Масштабування

Після того, як перестав виконуватися закон Мура, постала необхідність у масштабуванні систем ефективного накопичення, зберігання та обробки даних. Масштабування надалі будемо розуміти як можливість системи виконувати більше обчислень паралельно без втрати швидкодії. На прикладі баз даних масштабування

можна проінтерпретувати як здатність обробляти кілька запитів одночасно, тоді як швидкодію системи пов'язуватимемо з середнім часом обробки запиту.

Для поточної технології виробництва процесорів масового споживання запропоновано два підходи масштабування: горизонтальне та вертикальне.

Вертикальне масштабування досягається за рахунок підвищення потужності окремого обчислювального пристрою. Після припинення дії закону Мура вертикальне масштабування зводиться до збільшення кількості обчислювальних ядер у комп'ютері. Такий підхід має свої обмеження, адже кількість процесорів не може зростати необмежено хоча б в силу обмеженості фізичних розмірів комп'ютерів. Крім того вартість багатопроцесорних комп'ютерів непропорційно зростає з кількістю обчислювальних ядер, що може підтримувати система. Крім того існують і принципові обмеження для розпаралелювання обчислень, що пов'язані з послідовною структурою традиційних методів програмування.

Горизонтальне масштабування полягає у використанні великої кількості достатньо простих комп'ютерів. До цієї моделі масштабування звернулась компанія Google, що одна з перших зіткнулася з феноменом великих даних. Компанія Google вирішила, відмовитися від вертикально масштабованих серверів та використовувати комп'ютери масового вжитку у власних дата-центрах через невисоку вартість останніх. Горизонтальний підхід до масштабування значно менш обмежений у сенсі простоти додавання нових вузлів та їхньої вартості, якщо порівнювати з вертикальним підходом.

Програми побудовані на основі класичних методів програмування не починають працювати швидше за наявністю додаткових процесорів або обчислювальних вузлів. Аби скористатися перевагами масштабування необхідно належним чином розпаралелити програми. Розпаралелювання призводить до втрати детермінованості програм, які стають значно складнішими для проектування та аналізу, оскільки традиційні методи (такі як операційна семантика) розраховані на послідовне виконання операцій. Розпаралелювання алгоритмів є непростою задачею, тому все більшою популярністю користуються різні парадигми (набори рекомендацій) організації паралельних обчислень, які дозволяють описувати паралельні обчислення та надають відповідний програмний інструментарій, реалізований на різних мовах програмування.

Наприклад, такий підхід як SOA (Service Oriented Architecture) дозволяє розпаралелювати досить широкий клас обчислень на основі сервісів без станів (детермінованих функцій). Цей підхід реалізовано чи не на всіх популярних мовах програмування. Іншим прикладом може бути бібліотеки Akka, що ґрунтується на алгебрі взаємодіючих процесів та похідних формалізмах, таких як π-числення, та дозволяє описувати значно ширший клас паралельних обчислень. Реалізація цієї моделі програмування доступна на мовах програмування Scala, C# та Java.

Обмеження масштабування. Ідеальним при масштабуванні є випадок, коли продуктивність обчислювальної системи лінійно залежить від кількості обчислювальних вузлів у системі. Така ситуація називається лінійною масштабованістю, та є практично недосяжною.

Закон Амдала описує приріст швидкодії системи залежно від кількості наявних обчислювальних ядер. Приріст швидкодії $S(n)$ обчислювальної системи за рахунок використання n обчислювальних ядер визначається відношенням

$$S(n) = \frac{T_s + T_p}{T_s + \frac{T_p}{n}},$$

де T_s – час обробки послідовної складової обчислення одним обчислювачем, T_p – час обробки паралельної складової обчислення одним обчислювачем.

2. Масштабування сховищ даних

Масштабування сховищ даних є важливою проблемою при побудові систем машинного навчання, адже ефективне зберігання навчаючих даних є ключовим чинником, що впливає на ефективність роботи таких систем. Окремої уваги вимагає необхідність постійно поповнювати навчаючі дані щоб покращувати якість моделей за рахунок навчання на більш актуальних даних.

Масштабування реляційних баз даних. Традиційно для ефективного зберігання великих обсягів навчаючих даних використовуються реляційні бази даних та системи управлінні ними (СУБД). Це системи що надають користувачеві механізм зберігання та отримання даних у вигляді реляцій або таблиць. Фізично реляційні СУБД являє собою потужний вертикально масштабований сервер, що виконує запити клієнтів, рис. 1. Запити до реляційної СУБД, як правило, надходять у вигляді виразів мовою SQL, а результати повертаються у вигляді реляцій. СУБД може обробляти запити багатьох клієнтів одночасно. Серед відомих реляційних СУБД є Oracle, Microsoft SQL Server, MySQL, PostgreSQL та інші.

Для реляційних баз даних характерне зберігання даних у нормалізованому вигляді, з метою мінімізації надлишковості даних. Існує ціла градація нормальних форм, кожна з яких визначає сукупність вимог до реляцій, які дозволяють уникнути певного виду надлишковості даних. Нормалізація дозволяє ефективно зберігати реляційні дані, та водночас ускладнює їхню обробку. Адже для агрегації даних, розбитих між багатьма таблицями, необхідно виконувати додаткові операції.

Окрім нормалізації даних для реляційних баз даних характерне гарантоване виконання так званих вимог ACID:

- Атомарність (Atomicity) означає, що транзакція (послідовність операцій у запиті користувача) ніколи не буде виконана частково. Або всі операції виконуються успішно, або жодна з них не виконається у разі помилки хоча б однієї із операцій у транзакції.
- Узгодженість (Consistency) означає, що в результаті успішного виконання транзакції узгодженість даних не порушується. Однак під час виконання транзакції узгодженість даних може тимчасово порушуватися.

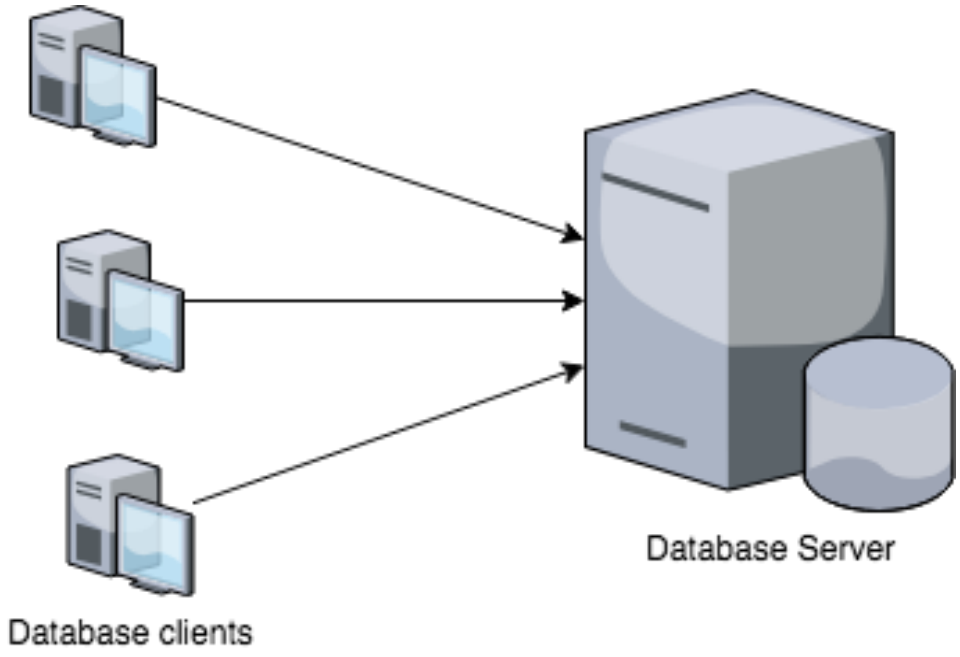

Рис. 1. Масштабування реляційних баз даних

- Ізоляція (Isolation) означає, що паралельні транзакції(обробка запитів різних користувачів) повинні бути незалежними. Ізоляція є досить складною для забезпечення вимогою, оскільки вимагає організувати ефективний доступ та модифікацію спільних ресурсів. Для забезпечення ізоляції в реляційних СУБД використовуються 2 стратегії блокування доступу до спільних ресурсів:
  - песимістичне блокування – припускає наявність колізій при паралельній роботі з ресурсом, тому доступ до нього надається тільки одній транзакції, а всі конкуруючі за доступ до ресурсу транзакції вишикуються в чергу;
  - оптимістичне блокування ґрунтується на припущенні, що колізії трапляються рідко, тому перед завершенням транзакції робиться перевірка чи не змінилася версія спільного ресурсу під час виконання транзакції. Якщо версія ресурсу не змінилася, то нова версія ресурсу зберігається в базі даних, інакше транзакція відповідальна за усунення колізій.
- Стійкість (Durability) – означає що результати успішних транзакцій будуть збережені навіть після відновлення роботи СУБД в результаті збою (такого як знеструмлення системи).

Нормалізація даних разом з вимогами ACID ускладнюють масштабування СУРБД, в результаті чого такі системи, як правило, традиційно масштабуються вертикально.

Горизонтальне масштабування нереляційних (NoSQL) баз даних. Існує ціла низка горизонтально масштабованих систем управління базами даних, що надають користувачеві механізм зберігання та отримання даних, які моделюються іншими засобами ніж табулярні реляції, які притаманні для СУБД. Такі СУБД називаються NoSQL СУБД, або NoACID СУБД, оскільки функціонують за рахунок послаблення вимог ACID. NoSQL СУБД з'явилися у 1960 роках однак набули популярності на фоні феномену великих даних. Фізично NoSQL СУБД представляють собою систему пов'язаних обчислювальних вузлів, на яких зберігаються дані, та поміж якими розподіляється обробка запитів користувачів, (рис. 2).

На сьогодні виділяють наступні типи NoSQL СУБД:

- Стовпчикові NoSQL СУБД, де дані зберігаються у вигляді стовпчиків, що характеризуються унікальним ім'ям, значенням та часом останньої модифікації). Прикладом стовпчикових NoSQL СУБД є Cassandra.
- Документо-орієнтовані NoSQL СУБД – орієнтовані на роботу з неструктурованими даними, або документами. Як документи, наприклад, можуть виступати звичайні текстові файли, або документи у

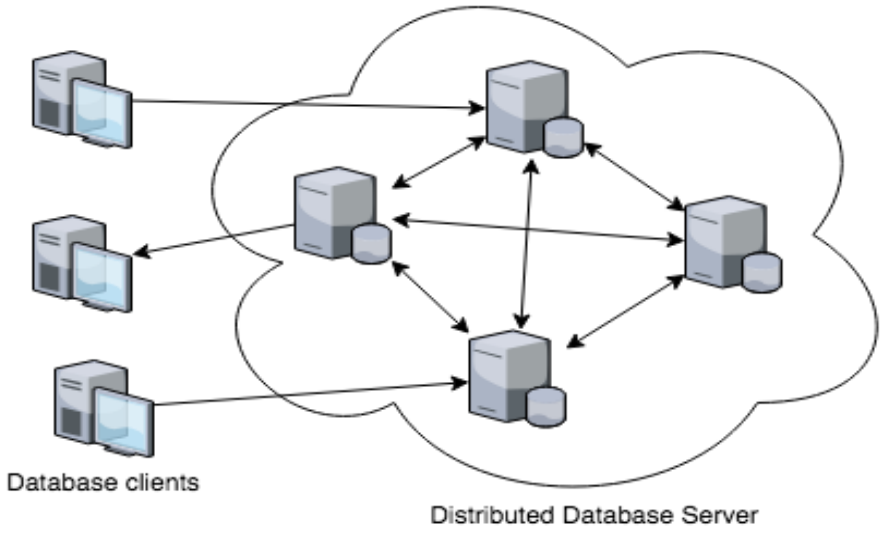

Рис. 2. Масштабування нереляційних (NoSQL) баз даних

форматі XML чи JSON. До стовпчикових відноситься така NoSQL СУБД як MongoDB.

- Ключ-значення NoSQL СУБД – дані представляються у вигляді хеш-таблиці пар ключ-значення. Ключ однозначно визначає значення, яке в свою чергу може мати внутрішню структуру та складатися з кількох елементів різних типів. До NoSQL СУБД типу ключ-значення відноситься Dynamo.

- Графові NoSQL СУБД – дані представляються у вигляді графів, тобто набору вершин та дуг, з якими пов'язані властивості різних типів. Популярною графовою NoSQL СУБД є Neo4j.

Характерними властивостями NoSQL СУБД є надлишковість при зберіганні даних, та зберігання їх у денормалізованому вигляді, а також відсутність підтримки розподілених транзакцій. Серед переваг – з'являється можливість горизонтального масштабування таких СУБД.

Обмеження горизонтального масштабування. Окрім закону Амдала існує ряд інших принципових обмежень для горизонтального масштабування.

Універсальний закон масштабування стверджує, що при збільшенні кількості обчислювальних вузлів продуктивність системи не тільки перестане зростати (згідно закону Амдала), але й взагалі знизиться. Це відбуватиметься за рахунок низки чинників, одним з яких є необхідність мережевої комунікації між вузлами в обчислювальній системі при досягнені консенсусу. Передача даних по мережі належить до операцій вводу/виводу, що є значно повільнішими за інші операції, такі як операції з оперативною пам'яттю.

CAP-теорема. Розглянемо важливі властивості розподіленої системи.

- **Узгодженість** (Consistency): всі користувачі отримують однаковий найактуальніший стан системи (або помилку), незалежно від вузла системи, що обробляє запит.

- **Доступність** (Availability): на кожен запит користувачів отримується успішна відповідь, без гарантії актуальності отриманих результатів.

- **Стійкість до втрати зв'язку** (Partition tolerance): система не втрачає працездатності та функціонує у нормальному режимі навіть за умов втрати (або затримки передачі) довільної кількості повідомлень при передачі даних між вузлами системи.

CAP-теорема стверджує, що з трьох вищенаведених властивостей в розподіленій системі одночасно досяжні не більше двох [4].

Оскільки передача даних у сучасній комп'ютерній мережі є ненадійною за своєю природою, тому у випадку розподілених СУБД неможливо знехтувати вимогою стійкості щодо втрати зв'язку (Partition tolerance), адже такі втрати завжди матимуть місце. При побудові розподілених СУБД вимога стійкості до втрати зв'язку реалізується за замовчуванням, а вибір робиться між системами, що є узгодженими (PC) або доступними (PA), рис. 3.

Популярні (PA) NoSQL СУБД гарантують послаблений варіант узгодженості системи, так звана узгодженість згодом (Eventual Consistency), що гарантує збіжність системи до узгодженого стану.

Досягнення узгодженого стану, або консенсусу, в розподілених системах стійких до обривів зв'язку є складною задачею, оскільки в таких системах не може бути фіксованої топології і, відповідно, центрального сервера, який виконував би функцію єдиного джерела актуальних стану системи. У системах, де постійно відбуваються обриви зв'язку, роль центрального сервера призначається динамічно, за допомогою алгоритмів досягнення консенсусу. Найпопулярнішими алгоритмами досягнення консенсусу, збіжність яких доведено, є Paxos та RAFT.

Окрім алгоритмів досягнення консенсусу значну роль при побудові (PA) NoSQL СУБД відіграють спеціальні структури даних CRDT (Conflict-free replicated data types), що дозволяються завжди розв'язувати неузгодженості при паралельній модифікації спільних ресурсів.

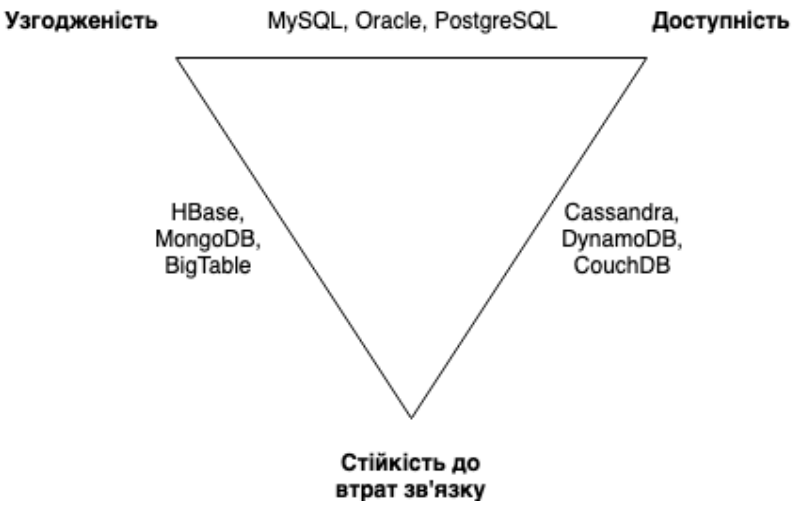

Рис. 3. CAP-теорема

3. Горизонтальне масштабування обчислень

Існує низка підходів до горизонтального масштабування обчислень. Тобто розпаралелювання обчислень за рахунок їх виконання на множині пов'язаних між собою обчислювальних пристроїв. При цьому масштабування відбувається за рахунок збільшення кількості обчислювальних вузлів системи. Горизонтальне масштабування підтримують такі методи як SOA або модель акторів, що вже згадувалися вище.

Одним з популярних підходів до масової обробки великих даних є модель програмування MapReduce що також використовує принцип горизонтального масштабування. Цей підхід надає досить простий інструментарій для описання широко класу обчислень на великих даних [5]. Простота досягається за рахунок представлення обчислень у вигляді базових операцій двох типів (Map та Reduce), що визначаються своїми алгебраїчними властивостями. Достатньо представити обчислення у вигляді композиції операцій двох зазначених типів, і реалізація моделі MapReduce бере на себе розпаралелювання відповідного обчислення, рис. 4.

При цьому дані, що обробляються, повинні зберігатися на вузлах розподіленої обчислювальної системи. При обчисленнях дані не рухаються між вузлами обчислювальної системи, а, навпаки, обчислення надсилаються до вузлів, на яких зберігаються дані для проведення обрахунків. Це

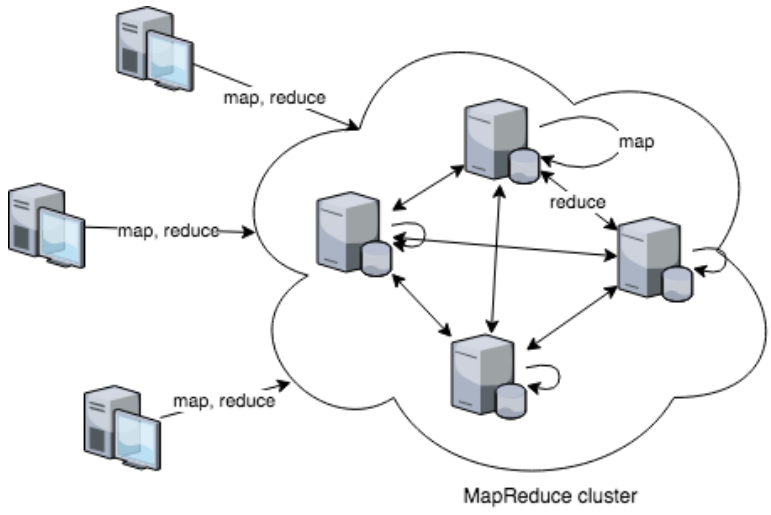

Рис. 4. MapReduce кластер з чотирьох вузлів

докорінно відрізняється від традиційного підходу Data Warehouse, за якого дані надсилаються для обробки до потужного вертикально масштабованого центрального сервера, де і проводяться обчислення.

Розглянемо основні засади, на яких ґрунтується модель програмування MapReduce. Нехай $a = (a_1, \dots, a_l)$, $a \in A^l$ – масив даних деякого типу A, розподілених між вузлами MapReduce кластера. І нехай $q: A^l \to B$ – складна процедура обробки розподіленої колекції a, що повертає результат деякого типу B. Обчислення $q(a_1, \dots, a_l)$ можна розпаралелити, якщо існує представлення

$$q(a_1, \dots, a_l) = m(a_1) \oplus \dots \oplus m(a_l),$$

де $m: A \to B$ операція обробки окремих елементів розподіленої колекції, що є детермінованою функцією, а $\oplus : B \times B \to B$ це бінарна операція, що застосовується для агрегації часткових результатів обробки масиву даних.

Операція агрегації $\oplus$ задовольняє властивостям комутативності, асоціативності та існування нейтрального елемента у множині B.

Обчислення $q(a_1, \dots, a_l)$ представлене у вигляді Map і Reduce можна зобразити у вигляді дерева, листки якого відповідають частковим результатам $m(a_1) \in B$, а вершини – операціям агрегації часткових результатів. Алгебраїчні властивості операцій Map і Reduce дозволяють перебудувати дерево обчислень зі збереженням остаточного результату обчислення таким чином, щоб мінімізувати обмін інформацією між вузлами обчислювальної системи, рис. 5.

Серед переваг підходу MapReduce є те, що у багатьох випадках він дозволяє наблизитися до лінійної масштабованості. Це досягається за рахунок мінімізації частки послідовних обчислень, що зменшує ефект закону Амдала, а також за рахунок мінімізації пересилки даних по мережі, що, в свою чергу, зменшує негативні наслідки Універсального закону масштабування.

Реалізації MapReduce. Модель програмування MapReduce є технічною специфікацією розробленою компанією Google, що обґрунтовує та надає рекомендації щодо ефективної організації масової обробки даних. Існує велика кількість реалізацій моделі MapReduce у вигляді бібліотек, або IaaS сервісів. Провайдери хмарних обчислень, такі як Google Cloud, Amazon Web Services, та Microsoft Azure дозволяють купувати кластери на основі MapReduce для масової обробки даних. Також існують провайдери спеціалізованих послуг, що фокусуються на MapReduce обчисленнях, наприклад, MapR, Databrix та Cloudera.

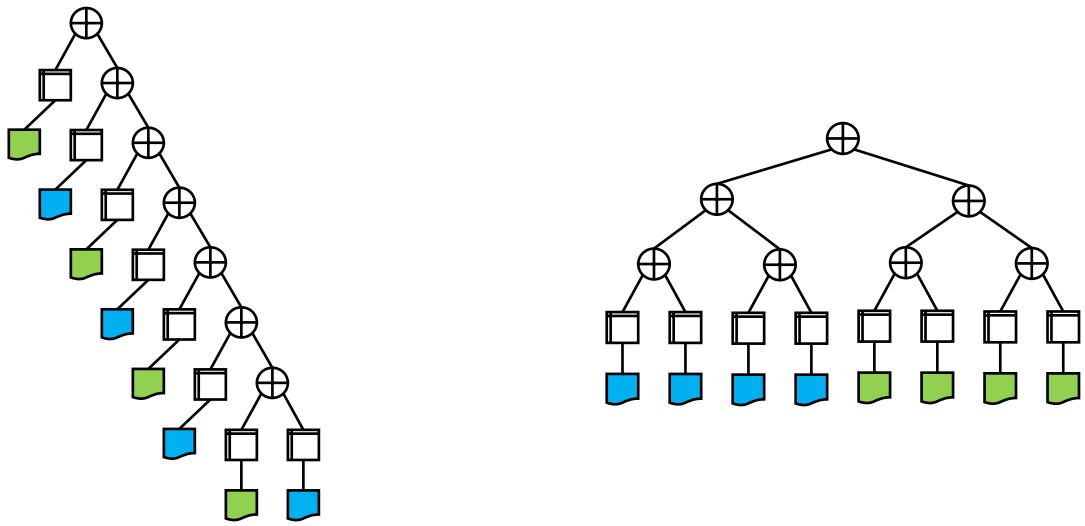

Рис. 5. Дерево обчислень MapReduce

Популярною реалізацією моделі MapReduce є бібліотека з відкритим кодом Apache Hadoop. Вона складається з двох модулів:

- з розподіленої файлової системи HDFS, що відповідає за ефективне зберігання розподілених даних;
- координатора розподілених обчислень YARN, що відповідає за координацію обчислювального навантаження між вузлами системи.

За допомогою цих двох модулів реалізується модель програмування MapReduce. Apache Hadoop є бібліотекою з відкритим кодом, тому її можна завантажити та встановити у приватній мережі комп'ютерів масового вжитку, або купити Hadoop кластер у вигляді IaaS.

Бібліотека Apache Hadoop набула значної популярності, не дивлячись на те, що базові операції Map і Reduce носять досить низькорівневий характер. Реальні обчислення складаються з композиції великої кількості таких операцій. Відповіддю на цей виклик стали більш високорівневі обчислювальні системи та бібліотеки побудовані на основі Apache Hadoop. Такі системи надають користувачеві більш високорівневі і, відповідно, більш зручні методи описання певних видів розподілених обчислень. Наприклад, існує ціла низка систем, що надають можливість виконувати SQL запити на Apache Hadoop кластерах. До таких систем належать Hive, Pig, Drill, та багато інших. Такі системи автоматично перекладають запити побудовані на мові SQL у операції Map і Reduce, та виконують їх, користуючись перевагами розподілених обчислень.

Інша відома високорівнева бібліотека, що використовує Apache Hadoop як обчислювальна модель є Apache Spark. Ця бібліотека надає зручний високорівневий інтерфейс для роботи з даними, та дозволяє зберігати проміжні дані та проводити обчислення над ними в оперативній пам'яті вузлів обчислювальної системи. Це дозволяє значно прискорити процес обчислень за рахунок уникнення чисельних звернень до накопичувачів даних, і відповідного зменшення повільних операцій вводу/виводу. Apache Spark складається з декількох модулів.

- Базового модуля Apache Spark, що реалізує базову структуру представлення даних, RDD – стійкий розподілений набір даних (Resilient Distributed Dataset), та надає високорівневі операції роботи з такими даними. Робота з розподіленими масивами даних при використанні RDD не відрізняється від роботи зі звичайними колекціями, з використанням функцій вищого порядку (тобто таких функцій, що приймають на вхід інші функції, які застосовуються до елементів колекції). Ці функції автоматично перекладаються на операції Map та Reduce та виконуються за допомогою Apache Hadoop.
- Spark Streaming – модуль, що дозволяє здійснювати розподілену обробку масивних потоків даних в реальному часі. Дані обробляються у по мірі їх надходження до обчислювальної системи. Це досягається за рахунок розбиття потоків даних у послідовність відносно невеликих RDD. Тому при роботі з потоками даних використовуються майже ті самі інтерфейси, що й при роботі з розподіленими даними.
- Spark SQL – модуль, що надає реалізацію мови запитів SQL використовуючи модель обчислень на основі RDD та Apache Hadoop. SQL – запити перетворюються на операції з RDD, які в свою чергу перетворюються на операції Map та Reduce та виконуються на Apache Hadoop.
- Spark ML – модуль, що дозволяє будувати горизонтально масштабовані розподілені методи машинного навчання, та надає цілу низку готових реалізацій різноманітних методів, серед яких: методи обробки навчаючих вибірок, методи виділення характеристик об'єктів, методи зменшення розмірності, методи навчання без учителя, методів класифікації та регресії, методи оцінки якості навчання, та багато інших.

4. Розподілені методи машинного навчання

Для з'ясування впливу феномену великих даних на методи машинного навчання розпізнаванню образів розглянемо

постановку такої задачі машинного навчання, як класифікація. Задача класифікації полягає у тому, щоб за допомогою навчаючої вибірки $\tau = (x_i, y_i)_{i=1}^{l}$ з пар $(x_i, y_i) \in X \times Y$ (характеристики об'єкта – клас), де X – множина можливих характеристик об'єкта, та Y– скінченна множина класів. Вимагається побудувати ефективну в деякому сенсі функцію класифікації $f: X \to Y$. Процедура $q(\tau)$ побудови функції класифікації $f(\cdot)$ за навчаючою вибіркою τ називається процедурою навчання.

Сама задача класифікації окремих об'єктів за наявністю побудованої в процесі навчання функції класифікації $f(\cdot)$, є, як правило, простою відносно нескладно в сенсі обчислень. Задача масової класифікації об'єктів досить просто розпаралелюється, оскільки розпізнавання окремих об'єктів відбувається незалежно, а сама функція розпізнавання є детермінованою. Горизонтальне масштабування таких обчислень є простою задачею. Обчислення такого виду можна розпаралелити за допомогою SOA, або за допомогою лише функції Map моделі програмування MapReduce.

Значно складнішою є процедура побудови функції класифікації $f(\cdot) = q(\tau)$ в процесі навчання. Навчання вимагає обробки масивних об'ємів навчаючих даних τ. Для ефективного виконання обробки навчаючих даних можна застосовувати модель програмування MapReduce. Для цього необхідно представити процедуру навчання $q(\tau)$ у вигляді операцій Map і Reduce, або описати її за допомогою більш високорівневих бібліотек. Наприклад бібліотека Apache Spark надає розподілені реалізації цілої низки методів навчання класифікації, таких як Байєсівські методи, випадковий ліс, дерево рішень, метод опорних векторів та інші.

Розподілена Байєсівська процедура розпізнавання. Розглянемо стадії навчання та класифікації на прикладі байєсівської класифікації. Байєсівський класифікатор будується на основі відомої формули Байєса у вигляді функції $f: X \to Y$

$$f_{\pi,\theta}(x) = argmax_{y \in Y} lg(\pi_x) lg(\theta_{x,y}),$$

що параметризована вектором оцінок апріорних ймовірностей класів π, а також матрицею умовних ймовірностей θ. Етап навчання $q(\tau)$ полягає у побудові вектора π та матриці θ за допомогою навчаючих даних τ. Процедуру навчання $q(\tau)$ можна розпаралелити, якщо представити її у вигляді операцій Map і Reduce.

Продемонструємо на прикладі обчислення вектора апріорних ймовірностей $\pi \in R^{|Y|}$ як виконується обробка розподіленого масиву даних під час навчання. Підрахунок елементів вектору в процесі навчання відбувається у вигляді

$$\pi_i = \frac{n_i(\tau)}{l},$$

де $n_i(\tau)$ кількість навчаючих прикладів, що потрапляють до i-го класу $i \in Y$, а l – загальна кількість навчаючих прикладів, $l = \sum_{i=1}^{n} \quad n_i(\tau)$.

Нехай Map це функція $m: X \times Y \to \to \{0,1\}^{|Y|}$, що ставить у відповідність навчаючому прикладу $(x, y) \in X \times Y$ булевий вектор розмірності $|Y|$ з одиницею на тій позиції, що відповідає порядковому номеру класу $y \in Y$, до якого потрапляє навчаючий приклад (x, y), решта елементів вектора є нульовими. Тоді операція Reduce є операцією $\oplus$ поелементної суми двох векторів розмірності $|Y|$, для таким чином визначеної операції агрегації виконуються необхідні умови асоціативності, комутативності та існування нейтрального елемента (нульовий вектор), в силу виконання всіх цих умов для операції звичайної суми цілих чисел. Після агрегації всіх часткових результатів, кожна позиція отриманого в результаті вектора відповідатиме кількості навчаючих прикладів певного класу.

Аналогічним чином підраховуються і елементи матриці θ оцінок умовних ймовірностей.

Штучні нейронні мережі та глибоке навчання. Останнім часом значної популярності набули методи глибокого навчання (Deep Learning), що використовують багатошарові штучні нейронні ме-

режі. Ці методи демонструють приголомшливі результати, особливо у розпізнаванні на неструктурованих даних. До таких задач відносяться автоматичний переклад текстів, виділення об'єктів на відео, виділення тексту зі звукових повідомлень та інші. Однією з важливих переваг таких методів є можливість автоматичного виділення характеристик об'єктів, за якими відбувається розпізнавання, з неструктурованих навчаючих даних.

Функція розпізнавання $f(x)$ у випадку штучних нейронних мереж будується у вигляді

$$f(x) = argmax_{i \in Y} a^l,$$

де σ – вектор-функція активації, a^i – збудженість нейронів i-го шару, b^i – вектор порогів активації нейронів i-го шару, а w^i матриця зв'язків нейронів i-го шару з нейронами попереднього шару.

Процедура навчання $q(\tau)$ зводиться до побудови матриць w^i та векторів b^i, що мінімізують похибку розпізнавання. Популярним методом навчання нейронних мереж є метод стохастичного градієнтного спуску, що використовує метод зворотного розповсюдження для підрахунку часткових помилок розпізнавання та для зміни відповідних коефіцієнтів матриць w^i та векторів b^i. Процедура навчання $q(\tau)$ штучних нейронних мереж заснована на зазначених методах має компактне представлення у матричній формі.

Метод градієнтного спуску є послідовною ітеративною процедурою, кожний крок якої залежить від результатів попередніх ітерацій. Навчання ускладняється глобальним характером оптимізації коефіцієнтів нейронної мережі, адже відсутність єдиності розв'язку унеможливлює агрегацію часткових результатів паралельних процесів оптимізації.

Наразі значні зусилля дослідників покладаються для розв'язання проблеми горизонтального масштабування процедур навчання штучних нейронних мереж. Одним з підходів є використання генетичних алгоритмів, за якого операція схрещування використовується для поєднання результатів, отриманих паралельними процесами оптимізації на основі стохастичного градієнтного спуску.

Масштабування штучних нейронних мереж. Слабка горизонтальна масштабованість змушує шукати альтернативні методи ефективної організації процедур навчання розпізнаванню штучних нейронних мереж. Останнім часом значної популярності набуло вертикальне масштабування за допомогою спеціалізованих графічних процесорів.

Архітектура графічних процесорів (GPU) відрізняється від архітектури традиційних процесорів (CPU).

Для традиційних CPU процесорів характерні: відносно мала кількість обчислювальних ядер (сучасні персональні комп'ютери масового вжитку мають 4 ядра), обчислювальні ядра можуть незалежно виконувати широкий спектр складних логічних операцій, відносно висока тактова частота, наявність розвиненої логіки оптимізації операцій.

Для графічних GPU процесорів навпаки, характерні: значно більша кількість обчислювальних ядер (кількість яких може сягати кількох тисяч), значно менша тактова частота, здатність виконувати паралельно лише однотипні та відносно прості операції, рис. 6.

Графічні процесори стають у нагоді для розпаралелювання масових однотипних операцій над масивами однорідних даних. До таких задач належать операції над матрицями, на яких ґрунтуються методи навчання розпізнаванню нейронних мереж. Горизонтальне масштабування за допомогою графічних процесорів реалізоване в популярній бібліотеці TensorFlow, запропонованою компанією Google [6]. Модель паралельних обчислень TensorFlow представляє обчислення у вигляді дерева, листки якого відповідають тензорам, а вузли відповідають операціям над тензорами.

Бібліотека TensorFlow є низькорівневим інструментом, тому існує низка більш високорівневих бібліотек, що спрощують розпаралелювання обчислень певного виду, використовуючи TensorFlow як

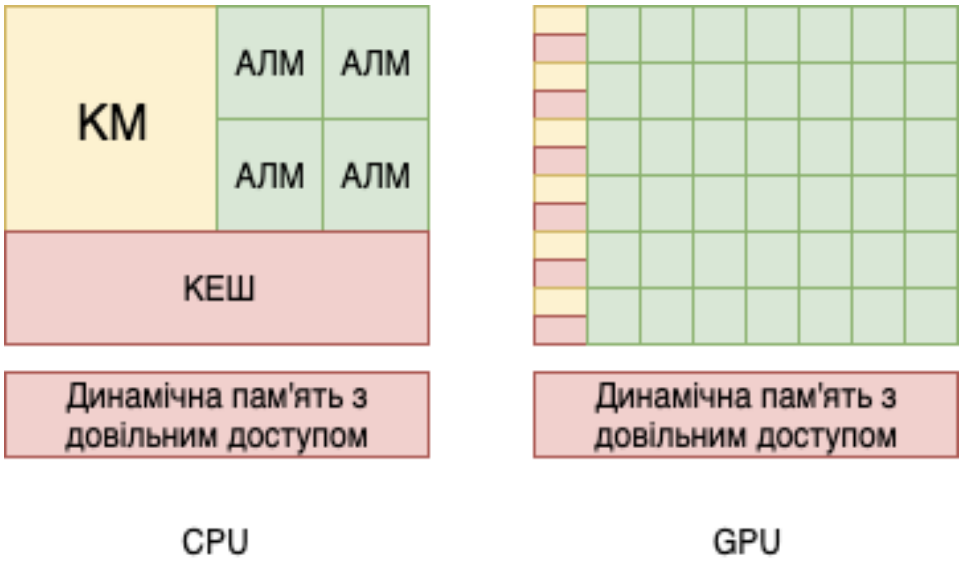

Рис. 6. Архітектура традиційного (CPU) та графічного (GPU) процесорів

обчислювальну модель. До таких бібліотек, належить Keras, що надає високорівневі методи роботи з штучними нейронними мережами.

Висновки

В роботі проаналізовано різні складові задачі машинного навчання розпізнаванню образів, а саме: 1) накопичення, збереження та обробка навчаючих даних; 2) машинне навчання; 3) розпізнавання. Було проаналізовано вплив феномену великих даних на кожну складову, а також методи масштабування та їхні обмеження. Показано, що більшість систем навчання розпізнаванню з учителем підлягають, як правило, горизонтальному масштабуванню, за виключенням методів навчання на основі нейронних мереж. Останні вимагають вертикального масштабування, оскільки використовують ітеративні алгоритми у процесі навчання.

Література

1. Hilbert M., López P. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science.* 2011. Vol. 332. P. 60−65.
2. Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, Mark Horowitz Communications of the ACM. Vol. 55, N 4. P. 55–63.
3. Moore G. Cramming more components onto integrated circuits. *Electronics.* 1965. Vol. 38. P. 114−117
4. Seth Gilbert and Nancy Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News. 2002. Vol. 33, Issue 2. P. 51–59.
5. MapReduce: Simplified Data Processing on Large Clusters by Jeffrey Dean and Sanjay Ghemawat https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
6. A computational model for TensorFlow: an introduction by Martín Abadi, Michael Isard, Derek G. Murray https://dl.acm.org/citation.cfm?doid=3088525.3088527

References

1. Hilbert M., López P. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science.* 2011. Vol. 332. P. 60−65.
2. Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, Mark Horowitz Communications of the ACM. Vol. 55, N 4. P. 55–63.

3. Moore G. Cramming more components onto integrated circuits. *Electronics.* 1965. Vol. 38. P. 114−117
4. Seth Gilbert and Nancy Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News. 2002. Vol. 33, Issue 2. P. 51–59.
5. MapReduce: Simplified Data Processing on Large Clusters by Jeffrey Dean and Sanjay Ghemawat https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
6. A computational model for TensorFlow: an introduction by Martín Abadi, Michael Isard, Derek G. Murray https://dl.acm.org/citation.cfm?doid=3088525.3088527

Одержано 18.04.2019

Про автора:

Білецький Борис Олександрович,
кандидат фізико-математичних наук,
старший науковий співробітник
Інституту кібернетики
імені В.М. Глушкова НАН України.
Кількість наукових публікацій в
українських виданнях – 13.
Кількість наукових публікацій в
зарубіжних виданнях – 2.
http://orcid.org/0000-0002-0667-8571

Місце роботи автора:

Інститут кібернетики
імені В.М. Глушкова НАН України.
03680, м. Київ,
Проспект Академіка Глушкова, 40.
Тел.: +38(044)526 2008.
E-mail: borys.biletskyy@gmail.com