МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені В. Н. КАРАЗІНА

І. В. Гущин
О. В. Киричок
В. М. Куклін

ВСТУП ДО МЕТОДІВ ОРГАНІЗАЦІЇ ТА ОПТИМІЗАЦІЇ НЕЙРОМЕРЕЖ

Навчальний посібник

Харків – 2021

УДК 004.8
Г 98

Рецензенти:

Б. М. Конорєв – доктор технічних наук, професор, лауреат державної премії УРСР, колишній начальник відділу програмного забезпечення бортових систем керування об'єктів ракетно-космічної техніки НВО «Хартрон», професор Національного аерокосмічного університету імені Н. Є. Жуковського «ХАІ»;

Г. М. Жолткевич – доктор технічних наук, професор, декан факультету математики та інформатики Харківського національного університету імені В. Н. Каразіна;

В. О. Філатов – доктор технічних наук, професор, завідувач кафедри штучного інтелекту Харківського національного університету радиоелектроніки.

Затверджено до друку рішенням Вченої ради
Харківського національного університету імені В. Н. Каразіна
(протокол № 15 від 28 грудня 2021 року)

Гущин І. В.

Г 98 Вступ до методів організації та оптимізації нейромереж : навчальний посібник / І. В. Гущин, О. В. Киричок, В. М. Куклін. – Х. : ХНУ імені В. Н. Каразіна, 2021. – 152 с.

ISBN 978-966-285-754-2

У книзі викладено в значній мірі розвинені в роботах попередників різні способи опису архітектури і проблем оптимізації нейронних мереж. Обговорюються методи навчання нейронних мереж з вчителем і без нього. Детально розглядаються різні технології градієнтного спуску при зворотному поширенні помилки. Обговорюється кластеризація при навчанні без вчителя і на її основі формування класифікацій. Розглянуто прості нейронні мережі і закладені в їх основу ідеї акредитації та асоціативної пам'яті. Представлені приклади нейронних мереж на основі нечіткої логіки. Значна увага приділяється вивченню багатопараметричних нелінійних систем і на цій основі обговорюється перехід до технологій глибокого навчання. Розглянуто різні види сучасних нейромереж. Обговорюються методи перетворення вхідних даних і принципи обробки текстів природної мови. Навчальний посібник є цікавим для аспірантів і студентів факультетів природничо-наукового профілю та комп'ютерних наук, які вивчають системи штучного інтелекту.

УДК 004.8

ISBN 978-966-285-754-2

Навчальне видання

Гущин Іван Валерійович
Киричок Олександр Віталійович
Куклін Володимир Михайлович

ВСТУП ДО МЕТОДІВ ОРГАНІЗАЦІЇ ТА ОПТИМІЗАЦІЇ НЕЙРОМЕРЕЖ
Навчальний посібник

Коректор *О. В. Анцибора*
Комп'ютерне верстання *О. С. Чистякова*
Макет обкладинки *Н. Є. Прудник*

Формат 60x84/16. Ум. друк. арк. 5,12. Наклад 100 пр. Зам. № 19/22.

Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб'єкта видавничої справи ДК №3367 від 13.01.2009
Видавництво ХНУ імені В. Н. Каразіна

Надруковано з готових оригінал-макетів у ФО-П Тітов Є. В.
61057, м. Харків, Харківська набережна, 9, кв. 23.
Свідоцтво про реєстрацію ВОО № 951823 від 18.01.1999 р.

ЗМІСТ

ПЕРЕДМОВА

Кафедра штучного інтелекту завдячує своїй появі рішенню першого декана факультету комп'ютерних наук Целуйка Олександра Федоровича, енциклопедично освіченої та креативної людини, якій довірили створення цього факультету. Треба сказати, що на кафедрах механіко-математичного та фізико-технічного факультетів і до появи факультету комп'ютерних наук було чимало викладачів та науковців, які активно користувалися методами чисельного моделювання та використовували інформаційні технології для вирішення складних завдань.Але все ж таки інтелектуальні системи були наступним кроком у освоєнні обчислювальних методів і технологій, що зажадало значних зусиль.

Велику методичну допомогу надав співробітникам кафедри декан механіко-математичного факультету Григорій Миколайович Жолткевич, який дозволив користуватися частиною своєї великої бібліотеки. Оскільки завданням кафедри було також навчання студентів перших курсів основам програмування та алгоритмізації, то процес формування курсів, які належали до навчання принципів та методів штучного інтелекту, йшов досить повільно.

Спочатку, відповідаючи завданням часу, було створено курс лекцій, який описує експертні системи. Його читав Є. В. Бєлкін, перший талановитий аспірант кафедри. Його колега, також колишня аспірантка Олександра Петренко, дуже обдарована випускниця ФКН, сформувала кілька лекцій з організації нейронних мереж, які також були включені до структури курсу «Експертні системи», лекції читали на четвертому році навчання. У цей час було надруковано інтегральний навчальний посібник кафедри, де було представлено ці та інші курси [1].

Там народився курс лекцій з основ логічних методів отримання рішень, який зараз називається «Методи та системи штучного інтелекту», підготовлений В. М. Куклініим, представлений у навчальних посібниках двома мовами [2, 3].

Після від'їзду Є. В. Бєлкіна до Канади та О. С. Петренко до Австралії, ці предмети почали читати випускники факультету О. В. Мишин та І. В. Гущин, які закінчили аспірантуру кафедри. Для розширення тематики

кафедри були запрошені відомі вчені, які здобули визнання світової наукової громадськості, В. А. Буц, В. І. Карась та В. В. Яновський, які забезпечили курси комп'ютерної фізики, альтернативних методів обчислень, зокрема з використанням квантових комп'ютерів [4].

Було освоєно методи паралельних обчислень за допомогою технології CUDA [5]. Пізніше ініціативу у викладанні комп'ютерного моделювання для природничих наук підхопив Ю. О. Аверков. З появою В. В. Яновського пожвавилася наукова діяльність у галузі вивчення роєвого інтелекту природних та штучних агентів, за допомогою А. В. Приймака, який успішно закінчив аспірантуру кафедри, була розвинена теорія еволюції великої кількості сценаріїв поведінки та їх носіїв [6].

Необхідність вивчення мов штучного інтелекту привела до появи циклів лекцій з мови «Пролог» та «Лісп», підготовлених відповідно А. В. Мішиним та А. Є. Споровим у рамках розширеного курсу «Експертні системи», який багато разів змінював назву і зараз називається «Теорія інтелектуальних систем та аналіз даних». Там же представлений цикл лекцій Є. В. Поклонського з вивчення експертних систем за допомогою нечіткої логіки.

Незважаючи на те, що деяке введення в роботу нейронних мереж залишилося в цьому курсі, підготовлене І. В. Гущиним, він разом з А. В. Мішиним та Є. В. Діденком, з деякою незначною участю В. М. Кукліна, підготували для магістрів першого року навчання сучасний курс «Розробка систем штучного інтелекту», який охоплював основні види машинного та глибокого навчання інтелектуальних систем.

Оскільки вже в роботі над дипломами бакалаврів студенти самостійно все частіше почали використовувати нейропакети, виникла потреба сформувати для третього року навчання курс «Нейронні мережі», для методичної підтримки якого представлено цей навчальний посібник.

Допомогу авторам посібника В. М. Кукліну та І. В. Гущину тут надав Є. В. Поклонський, який додатково підготував кілька лекцій та практичних занять саме з використання нейропакетів. Участь А. В. Киричка у створенні цього навчального посібника була необхідною для подальшого розширення тем навчання, які базуються на розвинених методах глибокого навчання, у сферу моделювання людської свідомості.

Автори вдячні рецензентам проф. Б. М. Конорєву, проф. Г. М. Жолткевичу та проф. В. О. Філатову за підтримку та корисні зауваження, а також проф. В. Т. Лазурику та проф. С. І. Шматкову за участь в обговоренні розлянутих тем.

Автори

Література до передмови

1. Введение в методы программных решений: учебное пособие / Е. В. Белкин, А. В. Гахов, А. М. Горбань, В. М. Куклин, В. М. Лазурик, А. С. Петренко, М. Ю. Силкин, В. В. Яновский; под ред. проф. В. М. Куклина. – Х.: ХНУ имени В. Н. Каразина, 2011. – 308 с.

2. Куклін В. М. Подання знань і операції над ними: навчальний посібник / В. М. Куклін. – Х.: ХНУ імені В. Н. Каразіна, 2019. – 164 с.

3. Куклин В. М. Представление знаний и операции над ними: учебное пособие / В. М. Куклин. – Х.: ХНУ имени В. Н. Каразина, 2019. – 180 с.

4. Яновский В. В. Квантовая механика алгоритмов. – Х.: ИСМА, 2009. – 272 с.

5. Гущин І. В., Куклін В. М., Мішин О. В., Приймак О. В. Моделювання фізичних процесів із використанням технології CUDA. – Х.: ХНУ імені В. Н. Каразіна, 2017. – 116 с.

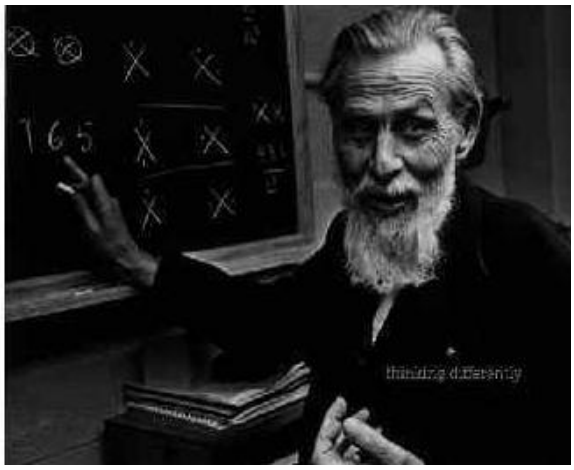
6. Kuklin V. M., Priymak A. V., Yanovsky V. V. A world of strategies with memory // Problems of theoretical physics. Scientific works. Issue 5 / V. A. But, et. al. Under the general edited by A.G. Zagorodny, N. F. Shulga, edited by V. A. Buts. – Kh.: V. N. Karazin Kharkiv National University, 2022.

ЧАСТИНА І

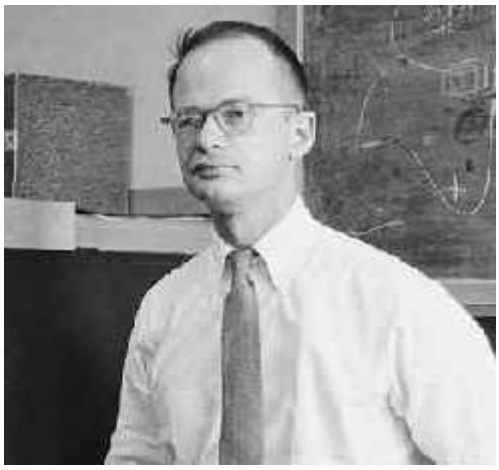
ВИНИКНЕННЯ НЕЙРОМЕРЕЖ

ВСТУП

Підхід до створення інтелектуальних систем – нейронних мереж спочатку принципово відрізнявся від побудови інтелектуальних систем і технологій на основі логіки. Творці цього науково-технічного напряму були нейрофізіологи, нейропсихологи і частково нейролінгвісти, добре ознайомлені зі структурою природної інтелектуальної системи – людського мозку. Причому багато з нейрофізіологів самостійно освоїли інженерні навички та програмування.



Воррен Маккалох – нейрофізіолог зі США (1898–1969)



Волтер Пітсс – нейрофізіолог-математик зі США (1923–1969)

Нейрофізіолог В. Маккалох (W. McCulloch) та нейролінгвіст і математик В. Піттс (W. Pitts) першими вирішили використовувати підказку природи і створили систему прийняття рішень, подібну роботі мозку (їх робота була опублікована у 1943 р.), – нейрони, представлені в запропонованій ними нейронній мережі, були порогові, тому їх називають нейронами Маккалоха–Піттса[1]. На вхід нейрона інформація надходила по

[1] Мак-Каллок У. С., Питтс В., Логическое исчисление идей, относящихся к нервной активности [1] (опубліковано у 1943 р.).

синапсах, яких у нейрона могло бути досить багато. Амплітуда сигналу множилася на вагу синапсу, тобто, взагалі кажучи, синапси були нерівноцінними і виконували роль параметрів системи. Це дозволило вважати нейронні мережі багатопараметричною обчислювальною системою з нелінійними операторами (подібно системі інтегродиференціальних операторів з різними параметрами, що визначають вагу–внесок кожного оператора), про що мова піде нижче.

Отже, саме В. Маккалох (W. McCulloch) і В. Піттс (W. Pitts) створили ***першу модель нейрона для нейромережі***[2] (1943). Синаптичні канали з позитивною вагою при позитивному вхідному сигналі стимулювали нейрон, а з негативними вагами його гальмували. Поріг b, при перевищенні якого нейрон реагував на вхідний сигнал, можна було вбудувати в штучний нейрон, наприклад, додавши нульовий синапс з вагою b, на нього подавався постійний одиничний сигнал.

Функція активації нейрона (результат дії нейрона) була подібна функції Хевісайда (або 0, або 1). Пізніше почали використовувати інші нелінійні функції активації, вид яких відрізнявся від форми функції Хевісайда, що робило вихідним сигналом вже сигнали не бінарного типу (і його можна було називати аналоговим[3]).

> ***Про рішення інтелектуальних завдань*** (місце нейронних мереж в структурі інтелектуальних систем). ***Знання і метазнання.*** Нагадаємо, що традиційне рішення інтелектуальних завдань у математичній логіці було засновано на формуванні понять (перший рівень – це літерали в теорії предикатів) і створенні правил-метазнань, за якими вони взаємодіють (наприклад, другий рівень – речення, пропозиції в теорії предикатів)[4]. Це так званий символьний опис (символьні підходи – symbolic reasoning). Правила взаємодії речень –пропозицій сформовані на основі деяких формалізмів, наприклад, в теорії предикатів на резолюції. Це еквівалентно твердженням, що рішення задач засновано на дедукції. ***Пряма і зворотна дедукція***. Пряма дедукція від фактів до висновків (іноді вони набувають сенсу цілей) має труднощі з вибором шляхів вирішення і не знайшла застосування в мовах логічного програмування. Однак вона корисна для самонавчання інтелектуальної системи, для отримання нового знання з раніше поданого. Зворотна дедукція (фактично – це доведення теорем) – від питань (цілей) до фактів, навпаки отримала розвиток у мовах логічного програмування (наприклад, ПРОЛОГ). ***Семантична павутина.*** Подальший розвиток логічних систем – це семантична павутина, а також формування предикатів високих порядків, далеких і близьких зв'язків-асоціацій між реченнями-пропозиціями***.***

[2] Нейрони цих мереж як створені, так і природні відносяться до структур типу лінійної ректифікації – розділення (rectified linear unit (ReLU)).

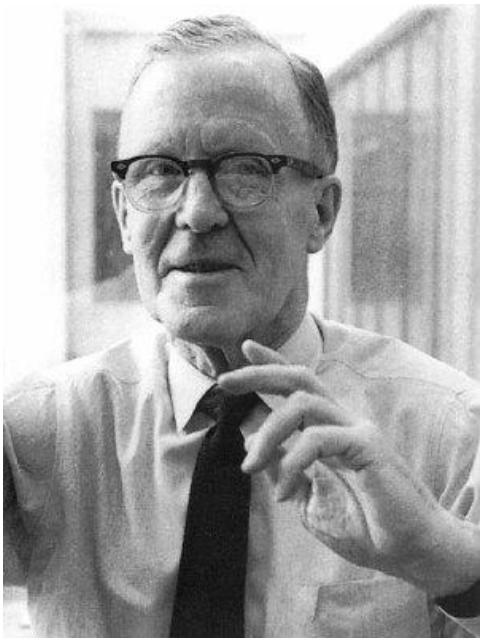
[3] Що представлені неперервною функцією.

[4] Див., наприклад, навчальні посібники [2] (http://dspace.univer.kharkov.ua/handle/123456789/15167).

Мови штучного інтелекту. У мові ПРОЛОГ – логічного програмування, заснованого на логіці предикатів першого порядку (і в перспективі на семантичних мережах, оформлених подібним чином), алгоритм вирішення (резолюція) побудований на узгодженні гілок (речень-пропозицій) явно або неявно створюваного машиною графа рішення (тобто це доказ теорем). Кваліфікація користувача може бути досить невисокою, що дозволяє застосовувати побудовані на цих мовах експертні системи повсюдно. Програміст може побудувати логічну систему зв'язків, орієнтуючись на тип завдання, між заданими в початкових умовах поняттями. І дозволити машині шукати умови узгодження початкових даних і обраної логічної схеми. По суті, рішенням є ці підібрані машиною умови. На основі функціональних мов (типу ЛИСП і його модифікації) з розвиненим логічним формалізмом, це цілком можна робити. Однак вимоги до кваліфікації програміста значно вище, ніж в разі застосування мов логічного програмування ПРОЛОГ. Оскільки роль програміста при використанні функціональних мов більш відповідальна, сама діяльність носить явно творчий характер, інтелектуалам цей підхід більше імпонує. ***У разі нейронної мережі*** все виявилося інакше. Перш за все, до цих пір неясно, як формується в цій мережі нове знання і створюється рішення. Що ж собою являли традиційні нейронні мережі? Тут на вхід мережі подається запит, і мережа реагує на нього розподіляючи на виході сигнали в n-вимірному просторі значень. Подібні (близькі за змістом) рішення локалізуються в якомусь одному місці простору виведення або навіть в глибині мережі. Для того, щоб розділяти рішення різного типу (класу, виду), потрібно, щоб області локалізації рішень різного класу не перекривалися. Саме тому перші нейронні мережі називалися персептрони, вони були орієнтовані на впізнавання об'єктів. Таким чином, в персептроні створюється перший рівень – система понять. З ускладненням нейронної мережі формується другий рівень – зв'язок між створеними в ній поняттями, тобто пропозиції–правила, чому теж доведеться мережу навчати. Подальший розвиток нейронної мережі – це вже формування в її середовищі семантичної павутини – прямих і зворотних (сильних) та асоціативних (слабких) зв'язків між пропозиціями–правилами. Нейронні мережі, як і людський інтелект, часто шукають рішення шляхом порівняння заданих умов завдання з успішної реєстрації аналогами, яких повинно бути достатньо багато. Якщо вдається знайти в масиві пам'яті аналогічні умови, що незначно відрізняються від заданих, то відповідні їм рішення стають рішеннями завдання. Нейронні мережі, як і людський інтелект, слід з обережністю класифікувати. Але вважати їх емпіричними системами, напевно, не слід. Природний інтелект (людський розум) – приклад подальшого розвитку і ускладнення нейронної мережі, де відбувається посилення можливостей системи зв'язків – «перколяції, – взаємопроникнення образів», що дозволяє формувати динамічні картини уявлень, те, що прийнято називати уявою. І вміння створювати гіпотези і аналізувати їх обґрунтування, що Ч. С. Пірс ще в XIX столітті назвав абдукцією, фактично здоровим глуздом [3]. Хоча висування гіпотез принципово не відрізняється від індуктивних методів побудови рішень.

Таким чином, зв'язок логіки і нервової діяльності підказав В. Маккаллоху та В. Піттсу в 1943 р. ідею нейронної мережі [4]. Поява книги «Кібернетика, або управління і зв'язок в тварині і машині» в 1948 р. Н. Вінера, батько якого був вихідцем з імператорської Росії, ознаменувала появу нової галузі знання – кібернетики. Здавалося, ці два напрямки повинні об'єднатися, що і сталося спочатку, але подальший розвиток теорії нейронних мереж У. Маккаллока призвів до відмови від об'єднання цієї теорії з кібернетикою.

Д. Хебб вже у 1949 році представляє алгоритми навчання нейронної мережі, але лише в 1958 році з'являється персептрон Ф. Розенблатта[5], в 1960 році Б. Уїдроу створив Адалін – адаптивний суматор (використовується в системах обробки сигналів) на керованих резисторах-мімісторах.



Дональд Олдінг Хебб – нейропсихолог з Канади (1904–1985)

Френк Розенблат – нейрофізіолог зі США (1923–1971)

Математики відзначили, що вид функції активації, яка передбачається обмеженою і нелінійною, якісно не впливає на характеристики мережі[6], для отримання потрібних рішень. У значно більшій мірі важливі зв'язки між нейронами. На відміну від природної нейронної мережі, штучна мережа, взагалі кажучи, не передбачає змін числа і стану ваг окремих нейронів,

[5] Rosenblatt F. Report 85–460–1, Cornell Aeronautical Laboratory, 1957.

[6] Спираючись на роботу Стоуна, Хехт-Нільсон довів, що функції загального типу можуть бути представлені двошаровою мережею з прямими повними зв’язками [5]. До речі, задля навчання мережі, скажімо, у рамках зворотнього розповсюдження помилки, знадобиться вже нелінійна і безперервно диференційована функція активації.

форми активаційної функції і умов на її вході[7] вже в процесі вирішення завдань. Одинокий нейрон дає можливість обчислити значення однієї нелінійної функції (активаційної), множина нейронів може обчислювати значення суперпозиції таких функцій.

Послідовно спрощуючи вид уявлення довільної функції як результату і суперпозиції стандартних функцій, спочатку А. Н. Колмогоров [6], потім Г. Лоренц [7], а потім і Д. Шпрехер [8] прийшли до висновку, що будь-яка функція може бути представленою кінцевою сумою деяких функцій, аргументом яких є кінцеві суми стандартних функцій. Іншими словами, шари мережі при певній її архітектурі формують суму своїх відгуків (кожен відгук у вигляді цієї деякої функції), аргументами якої є функції активації окремих нейронів. Математики підказали програмістам, що таким чином можна отримати в результаті будь-яку функцію, тобто мережа дозволяє розраховувати, що вона знайде рішення в будь-якому випадку[8].

Програмування нейронної мережі ***замінюється її навчанням***. Або з'ясуванням реакцій мережі на вхідні сигнали, що формує значну кількість виділених вихідних образів (в цьому полягає налаштування – оптимізація мережі), які потім аналізуються і формується їх набір. З цим набором потім порівнюються правильні рішення.

В результаті виникає можливість отримання правильного результату в ситуації, не розглянутої в процесі навчання або розпізнавання об'єктів попередньо налаштованою мережею.

Машинне навчання пропонує підходи, які навчають мережу знайти правильне рішення лише з певною ймовірністю. А створити систему навчання для всіх випадків не виявляється можливим. Але для деякого класу задач результат навчання може виявитися прийнятним.

Існує безліч технологій навчання, які можна розбити на кілька класів. Це, перш за все, традиційне навчання з учителем (supervised learning або fully supervised learning), де для кожного навчального прикладу (питання) є правильна відповідь.

Можливо навчання з частковим залученням вчителя (semi–superviser) або взагалі без вчителя (unsupervised learning). Взагалі кажучи, навчання – це завдання оптимізації нейронної мережі. У разі навчання з учителем це призводить до виправлення помилок – відхилень отриманого мережею рішення від заданого навчальною вибіркою відомого рішення. У разі навчання без учителя краще говорити про завдання класифікації (classification problem), коли за ознаками (feature), які ідентифікують, класифікують предмет-сутність, яка повідомила мережі свої вхідні дані, мережа знаходить відповідь.

---

[7] У випадку налаштування та навчання мережі такі зміни можливі.

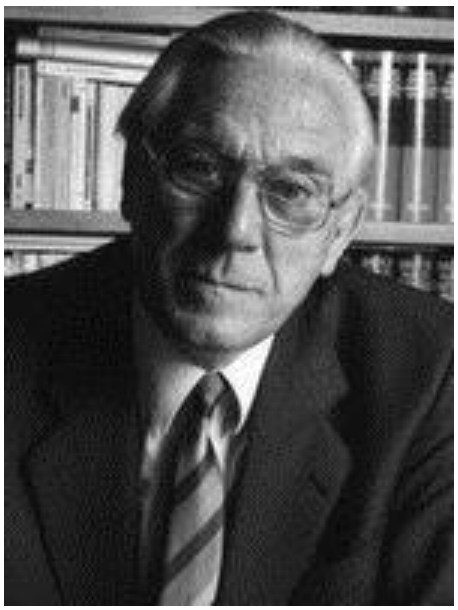
[8] Детально про математичну базу нейронних мереж див. [9].

У 1959 р. Д. Хьюбел і Т. Візель (D. Hubel, T. Wiesel) обговорили характер зберігання і обробки інформації у біологічних нейронних мережах.

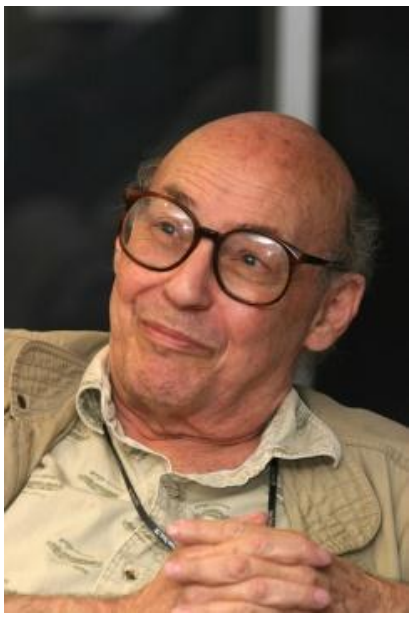
Девід Хьюбел – нейрофізіолог з Канади (1926–2013)

Візель Тетяна Григорівна (1938 р. нар.) – нейропсихолог з Росії

В. Уїдроу (W.Widrow) на початку 60-х рр. створив кілька нейрокомп'ютерів, К. Штайнбух (K. Steinbuch) запропонував асоціативні матриці. Але охолодження інтересу до нейромереж викликала претензійна книга М. Мінського і С. Пейперта (M. Minsky, S. Papert).

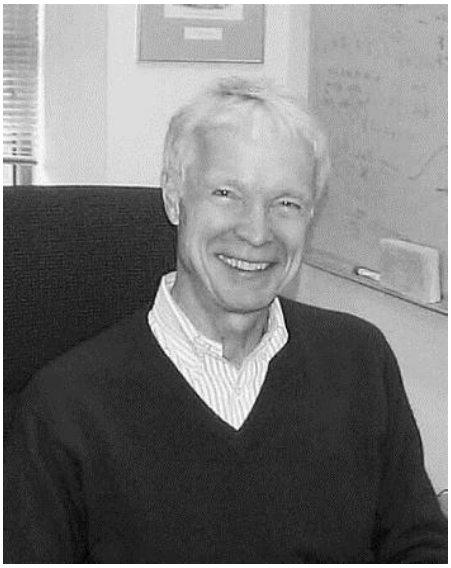


Карл Штейнбух – кібернетик з Німеччини (1917–2005)

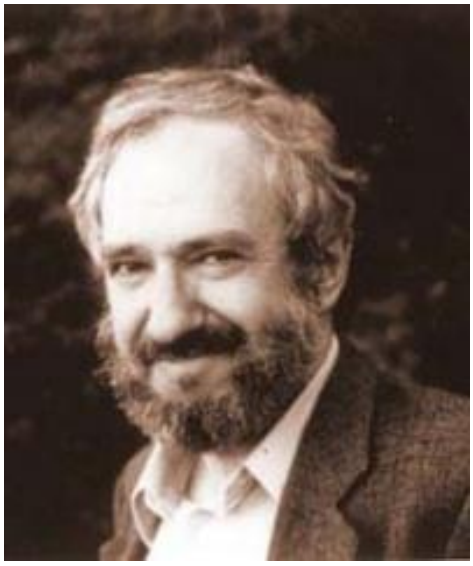


Мінський Марвін Лі когнітивіст зі США (1927–2026)

У 1982 р. Дж. Хопфілд (J. Hopfield) запропонував мережі, що моделюють асоціативну пам'ять і методи їх навчання.



Джон Джозеф Хопфілд – американский фізик зі США (1933 р. нар.)



Сеймур Пейперт – математик-програміст зі США (1928–2016)

У низці джерел (див., наприклад, [10]) першою (пороговою) моделлю біологічного нейрона вважали модель Л. Лапіка (1907), яка генерувала імпульси.

Минуло багато років, поки подібну, але вже позбавлену недоліків і відповідну природним властивостям біологічного нейрона модель запропонували А. Ходжкин і А. Хакслі [11], яку через складність в нейронних мережах не часто використовують. Спростив цю модель спочатку Р. Фітзхуг [12], який ввів кубічну нелінійність, потім цим зайнялися С. Моріс і Х. Лекар [13] та інші дослідники. Але нервова клітина знаходиться в оточенні сусідів, і її поведінка відрізняється від поведінки одиничного нейрона. Була виявлена так звана синаптична пластичність: зміна зв'язків-синапсів у відповідь на зовнішній вплив. Тому Д. Хебб сформулював правило, за яким при активації нейрона зв'язок посилюється (LTP – long-term potentiation). Але якщо нейрони дезактивовані, в тій же мірі зв'язки можуть послаблюватися (LTD – long-term depression). Синаптична ефективність, як виявилось, залежить від часу приходу на синапс збудження або пригнічення активності, хімічного стану оточення і т. п. Але слід мати на увазі, що при моделюванні роботи природних нейронних систем використовуються спеціалізовані нейромережі – нейрональні симулятори

[10] як з простою моделлю нейрона (PCSIM, NEST, Brain і NCS), так і з більш складними імітаторами нейронів (NEURON, GENESIS, SPLIT і MOOSE, MVASPIKE). Найбільш поширені програмні пакети NEST (Neuron Simulation Tool) і NEURON. Створюються також апаратно-реалізовані нейрональні мережі (наприклад, FACETS). Але в цій книзі обмежимося поданням нейрона в прийнятій найбільш спрощеній формі.

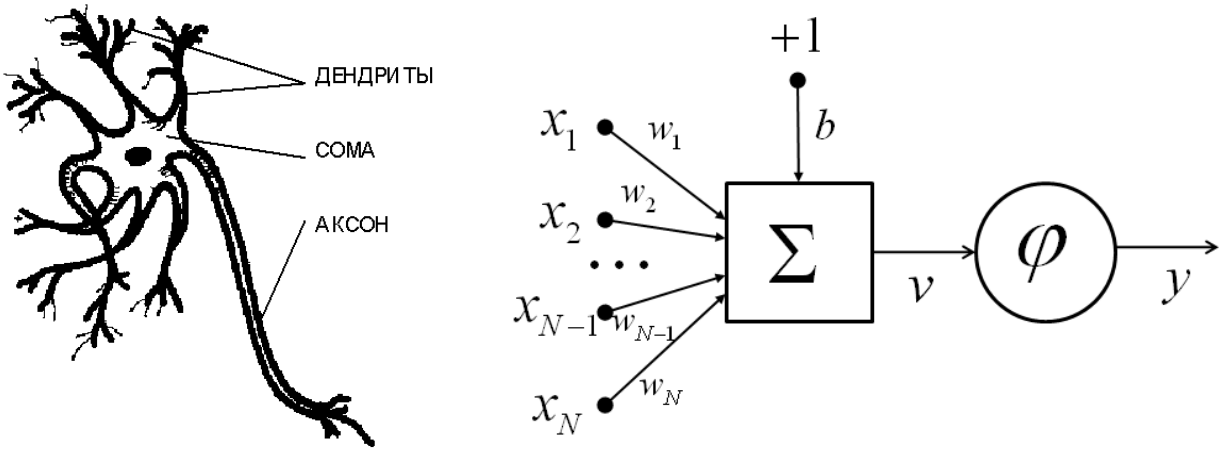


Рис. 1. Природний і штучний нейрони, $s = \sum_{i=1}^{n} w_i \cdot x_i + b$, $y = \varphi(s)$

При сигналах на входи нейронів $x_i, i = 1,...n$ формується сумарна зважена величина $\sum_i w_i x_i$, яка зазвичай обробляється активаційною (сигмоїдною) функцією $y = \varphi(\sum_i w_i x_i)$. Тут w_i – вага синапсу, b – зміщення (bias). Сигмоїдні функції звичайно представлені у вигляді $\varphi(s) = 1/[1+\exp\{-as\}]$, причому похідна $d\varphi(s)/ds = a \cdot \varphi(s)[1-\varphi(s)]$.

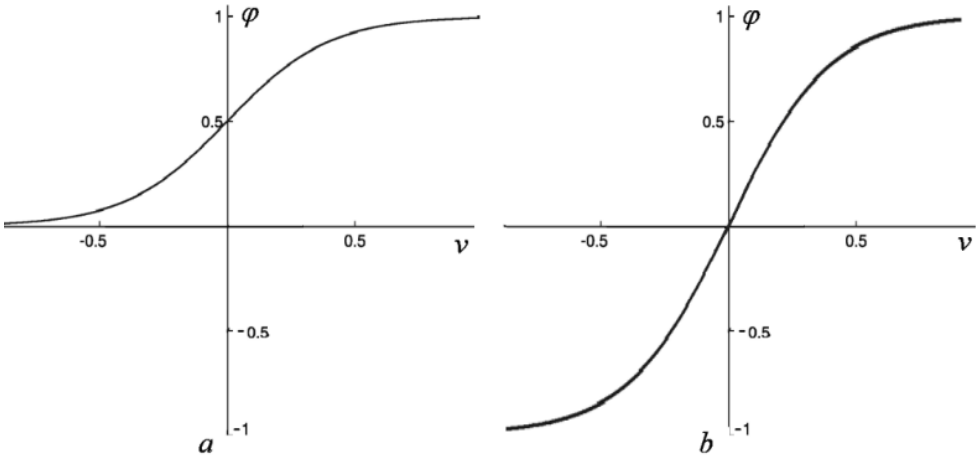


Рис. 2. Функції реакції нейрона на зовнішній вплив

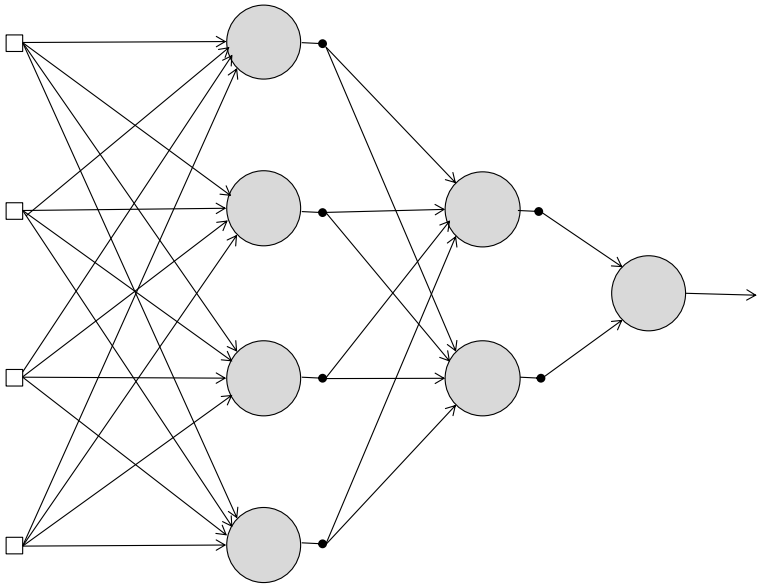


Рис. 3. Приблизна схема простої штучної нейронної мережи

Згодом оптимізм згас. Невдачі машинного перекладу (в основному з російської на англійську мову) змусили розчарованих керівників США в період холодної війни згорнути фінансування цієї програми.

Успіхи логічних – конекціоністських методів (символьні підходи – symbolic reasoning) штучного інтелекту відсунули на узбіччя підтримки чиновників прихильників коннективізму (винахідливого створення блоків з нейронів, як у мережі У. Піттса і У. Маккалоха) і успіх персептронів. Сумніви А. П. Петрова і М. М. Бонгард [14, 15] в можливостях персептрона в 1963 році, а також згадувана вище робота однокурсника Ф. Розенблатта М. Мінського і його колеги С. Пейперта [16] про неможливість вирішувати завдання з інваріантними уявленнями (див. також проблему єдності рішень) в 1969 році привели до першої зими в розвитку нейронних мереж. Хоча в цей час інтерес до цієї теми не згас (див., наприклад, [17, 18]). Т. Кохонен і Дж. Андерсон у 1972 р. запропонували використовувати нейронні мережі як системи пам'яті. Б. В. Хакімов у 1973 р. використовував модель з синапсами для вирішення завдань, П. Дж. Веброс, а також А. І. Галушкин у 1974 р. обговорювали алгоритм зворотного поширення помилки, але ці роботи залишилися без уваги.

Навіть ідея самоорганізованої мережі-когнітрону і потім неокогнітрону Фукусіми[9] (Fukushima, 1975, 1980) для розпізнавання образів не викликала інтересу, хоча через два десятиліття лягла в основу сучасної згорткової мережі.

[9] Цікаво, що архітектура мережі була подібна невідомої тоді згорткової мережі.

Чималу частку розчарування викликав негативний звіт Дж. Лайтхілла про можливості нейронних мереж. Підхід до підтримки досліджень, які створювали нові технології, корисні для держави, завжди був традиційний для чиновників, особливо військових. Як відомо, агентство передових оборонних досліджень «ARPA» (тепер «DARPA») дало можливість М. Мінському, Г. А. Саймону, А. Н'юелу та іншим авторитетам витрачати будь-які фінанси та засоби для прогресу в тих областях, які вченим здавалися перспективними.

Аллен Н'ювел зі США
(1927–1992)

Герберт Саймон зі США
(1916–2001)

Але вже в 1969 р. (поправка Менсфілда) від вчених і технологів зажадали вже цільових розробок. Звіти Дж. Лайтхілла та інших залучених військовими експертів привели до 1974 року в розпал економічної кризи до очікуваного згортання фінансування програм штучного інтелекту (ШІ). У цей період мільярдна галузь штучного інтелекту була завалена через розчарування чиновників від науки.

Закінчення першої зими ШІ (1982) ознаменувало появу в 1982 році мережі зі зворотними зв'язками Дж. Хопфілда і мережі Кохонена, причому остання навчалася без учителя і забезпечувала кластеризацію рішень самостійно. Потім у 1986 р. ціла когорта дослідників (західних – Д. І. Румельхарт, Дж. Е. Хінтон, Р. Дж. Вільямс практично одночасно з вченими красноярської групи на чолі з С. І. Барцевим) повернулися до методу зворотного поширення помилки.

Мережі спочатку були невеликими (незначні кількість шарів, кількість нейронів, кількість синапсів-зв'язків), і тому не припинялися

спроби створення їх формального – аналітичного опису, що пізніше призвело-таки до появи безлічі теорій глибокого навчання.

Теуво Калеви Кохонен з Фінляндії (1934 р. нар.)

Джеймс (Джим) А. Андерсон когнітивіст зі США (1940 р. нар.)

Девід Румельхарт – когнітивіст зі США (1942–2011)

Сергій Ігорович Барцев – математик-біофізик з Росії (1955 р. нар.)

У цей період стали домінуючими в усвідомленні проблем ролі нейрофізіологів і математиків. Перші, спираючись на знання систем головного мозку, пропонували все нові варіанти архітектури та способи конструювання рішень, а математики намагалися довести раціональність деяких, що сподобалися їм, мереж і методів.

Після десятиліття досліджень, у 1983 році DARPA запустили Ініціативу стратегічних обчислень (Strategic Computing Initiative).

Програмою досліджень керувало Управління технологій обробки інформації (IPTO). Через чотири роки розчаровані результатами чиновники все згорнули (друга зима ШІ 80-х років), крім систем управління боєм DART, де, до речі, штучний інтелект проявив себе прекрасно (якщо можна так казати) під час першої війни в Перській затоці у 1991 році.

Відомим став скандальний факт висловлювання колеги вищезазначених вчених Х. Моравека[10], який звернув увагу громадськості на схильність вчених до нереалістичних прогнозів, а чиновників – до завищеної вимогливості до результативності наукового пошуку.

Агентство в цей же час також скасувало фінансування дослідникам з університету Карнегі–Меллона, які працювали над програмою розпізнавання мови (на основі прихованих марківських моделей), але тільки при формуванні шарів мережі в певному порядку. Але, як водиться, розробки цих вчених допомогли розвинути ці методи лише через роки, хоча і до початку нового століття.

Стимулювали відродження інтересу до ШІ технології комп'ютерного зору (з 2000 року) та вимушений інтерес до обробки великих даних (з 2010 року), появи робіт групи Дж. Хінтона з глибоких мереж довіри[11] і розвиток підходів, названих пізніше глибоким навчанням.

Джефрі Хінтон – творець сучасних нейронних мереж, з Британії, зараз у північній Америці (1947 р. нар.)

У першому десятиріччі нового століття технології зберігання інформації і ETL привели до появи розробок з відкритим кодом Hadoop Mongo DB. Успіхи системи Watson (що стала переможцем у грі Jeopardy 11) і AlphaGo (перемогла людину у грі «го») сформували хвилю оптимізму, який спровокував можливо необґрунтований багато в чому інтерес і потік інвестицій.

Мабуть, після очевидних успіхів у застосуванні нейронних мереж в якості інтелектуальної начинки спершу

[10] Парадокс Моравека — «Порівняно легше забезпечити комп'ютер можливістю виконання таких складних операцій, як гра в шахи і тестування рівня продуктивності пристроїв, але майже неможливо навчити його навичкам навіть однорічної дитини, коли мова йде про швидкість обробки інформації».

[11] https://www.cs.toronto.edu/~hinton/absps/fastnc.pdf.

керованих руками машин, оптимізм інвесторів вже буде збережений і нових зим штучного інтелекту вже можна не очікувати. Але застосування машин, що володіють штучним інтелектом та працюють саме в оточенні людей, буде відбуватися дуже поступово через ефект «Connor Case», коли рішення цих машин, які б задовольняли людей[12], знайти взагалі не вдасться.

Слід відзначити, що для наближення ШІ до людської свідомості потрібно пройти ще довгий шлях до появи ***штучної істоти, що усвідомлює себе*** і навколишній світ, можливості якої будуть порівнянні з можливостями природного розуму.

Література до вступу

1. Мак-Каллок У. С., Питтс В. Логическое исчисление идей, относящихся к нервной активности // Автоматы; под ред. К. Э. Шеннона и Дж. Маккарти. – М.: Изд-во иностр. лит., 1956. – С. 363–384.
2. Куклін В. М. Подання знань і операції над ними: навчальний посібник / В. М. Куклін. – Х.: ХНУ імені В. Н. Каразіна, 2019. – 164 с. Куклин В. М. Представление знаний и операции над ними: учебное пособие / В. М. Куклин. – Х.: ХНУ имени В. Н. Каразина, 2019. – 180 с.
3. E. Larsen ‘The Myth of Artificial Intelligence: Why Computers Can’t Think the Way We Do./ Kindle Edition.
4. Мак–Каллок У. С., Питтс В. Логическое исчисление идей, относящихся к нервной активности. Архивная копия от 27 ноября 2007 на Wayback Machine // Автоматы / Под ред. К. Э. Шеннона и Дж. Маккарти. – М.: Изд-во иностр. лит., 1956. – С. 363–384. (Перевод английской статьи 1943 г.).
5. Hecht–Nielsen R. Kolmogorov’s Mapping Neural Network Existence Theorem // IEEE First Annual Int. Conf. on Neural Networks, San Diego, 1987, Vol. 3, pp. 11–13.
6. Колмогоров А. Н. О представлении непрерывных функций нескольких переменных суперпозициями непрерывных функций меньшего числа переменных // Известия АН СССР, 108 (1956), с. 179–182.
7. Lorentz George. Metric entropy, widths, and superpositions of functions // American Mathematical Monthly. 1962, vol. 69, p. 469–485.
8. Sprecher D. A. On the structure of continuous functions of several variables //Transactions of the American Mathematical Society. 1965, vol. 115, p. 340–355.
9. Яновский В. В. Коллективный интеллект. НТК «Институт монокристаллов» НАН Украины. – Киев: Наукова думка, 2020.

[12] Люди готові погодитися з існуванням технологічних катастроф через людський фактор (це загибель людей на транспорті, невдале лікування, зараження харчовими продуктами і т. п.), але вони не хочуть, щоб ці катастрофи відбувалися з вини машин. Як вийти з цього етичного глухого кута, стало завданням психологів і навіть філософів (див. також https://www.youtube.com/watch? v = G4w-aqIEV50).

10. Математическое моделирование нейродинамических систем / Прокин И.С., Симонов А.Ю., Казанцев В.Б. Электронное учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2012. – 41 с.

11. Hodgkin A.L., Huxley A.F. A quantitative description of membrane current and its application to conduction and excitation in nerve // The Journal of physiology. 1952. Vol. 117, № 4. P. 500–544.

12. Fitzhugh R. Impulses and Physiological States in Theoretical Models of Nerve Membrane. // Biophysical journal. 1961. Vol. 1, № 6. P. 445–466.

13. Morris C., Lecar H. Voltage oscillations in the barnacle giant muscle fiber // Biophysical journal. 1981. Vol. 35, № 1. P. 193–213.

14. Петров А. П. О возможностях перцептрона // Известия АН СССР, Техническая кибернетика. –1964. – №6.

15. Бонгард М. М. Проблемы узнавания. – М.: Физматгиз, 1967.

16. Минский М., Пейперт С. Перцептроны. – М.: Мир, 1971 (https://ru.wikipedia.org/wiki/ Перцептроны)

17. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. – 2 изд. – М.: Горячая линия – Телеком, 2002. – 382 с.

18. Куклин В. М. Особенности развития искусственного интеллекта на современном этапе // Вісник Харківського національного університету імені В.Н. Каразіна, серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». – 2018. – С. 34–40.

РОЗДІЛ 1. НАВЧАННЯ МЕРЕЖІ З УЧИТЕЛЕМ

Машинне навчання використовує алгоритми, які мінімізують похибки рішення мережею завдання методами (параметричної) оптимізації. Оптимізація змінює параметри мережі для кращого наближення до заданих рішень завдань навчальної вибірки або для більш чіткої класифікації (кластеризації) отриманих мережею рішень у відсутності навчальних прикладів.

Види навчання: навчання з учителем – на кожний вхідний образ є необхідна відповідь, і цього слід добиватися. Часто цей вид навчання називають ***параметричною оптимізацією***. Параметричною тому, що змінюють параметри, наприклад, ваги, а оптимізацією тому, що оптимізується вихід мережі з відомими рішеннями навчальної вибірки, тобто зменшується різниця між отриманим і потрібним рішенням.

Якщо необхідної відповіді немає, тобто немає навчальної вибірки, то потрібна хоча б оцінка правильності відповіді (жорстке навчання). При навчанні без учителя – це вже самонавчання мережі, задають безліч певних і усвідомлених користувачем образів, не знаючи відповідей мережі. Мережа сама формує на виході кластери – області згущення відповідей. Потім користувачеві доведеться розбиратися з цим набором кластерів, визначаючи їх зміст. Крім того, слід домагатися високого рівня відокремлення образів-кластерів. Це можна вважати настроюванням мережі. Є змішані варіанти навчання, коли використовуються обидва підходи.

Правила навчання: 1. Корекція помилки – ваги змінюються для зменшення різниці між виходом мережі і необхідним значенням. 2. Правило Больцмана – розподіл вагових коефіцієнтів може відповідати обраному розподілу ймовірностей. 3. Правило Хебба – зміна ваги синапсу зростає при збільшенні активності нейронів по обидві сторони цього синапсу. 4. Метод змагань – вихідні нейрони змагаються між собою, і нейрон з максимумом зваженої суми є переможцем. Решта нейронів пригнічуються (переможець забирає все). Часто навчаються тільки переможці (в основному вихідного шару), причому їх ваги модифікуються в сторону близькості до даного вхідного сигналу (до вибраного, наприклад).

Навчання з учителем. Для цього потрібні вхідні образи і еталонні вихідні. Набір цих величин називають ***навчальною вибіркою*** – набором прикладів вирішення цього класу задач. Експерти надають цю вибірку. Звичайно частину цих прикладів резервують для перевірки[13] ступеня навчання мережі і при безпосередньому навчанні не використовують. Когнітивні психологи, що моделювали процеси мислення, Девід Румельхард і Джеффрі Хінтон (1986) запропонували найбільш поширений нині метод

[13] Після використання цих резервних прикладів їх контрольний потенціал вичерпано і більше їх використання неможливе (детальніше див. [1]).

навчання мережі з учителем – метод зворотного поширення помилки: оцінивши ступінь відхилення від потрібної відповіді, вони рекомендували пройтися в зворотному порядку по верствах мережі і налагодити налаштування. До речі, пізніше вважали, що метод зворотного поширення помилки зазвичай менш ефективний, ніж безпосередня оптимізація «функції вартості» за допомогою градієнтного спуску [2].

Проблеми навчання мережі прямого поширення. Обговоримо, спираючись на міркування У. Мікелуччі [3], проблеми навчання мережі. Оскільки існували труднощі застосування відшукання мінімуму вартісної функції (або функції помилки) в мережі з більшим чи навіть величезним числом зв'язків – ваг синапсів, то були змушені використовувати чисельні методи. Налаштування ваги здійснюється методом градієнтного спуску $w_{n+1} = w_n - \gamma \nabla J(w_n)$, де $J(w_n)$ – вартісна функція, γ – гіперпараметр (тобто параметр, заданий до початку навчання), що визначає швидкість налаштування (rate of learning). У якості вартісної функції часто вибирають середньоквадратичну помилку (mean squared error, MSE). Якщо вартісна функція стає менше деякого необхідного значення, то мережу можна вважати налаштованою. Зазвичай число ітерацій налаштування дуже велике, що створює умови для зростання помилок. Гіперпараметр γ, якщо він завищений, не дасть налаштувати мережу, система буде коливатись близько потрібного значення мінімуму функції $J(w_n)$. Важливо співвідносити заучування з числом ітерацій, це визначає ефективність тренування. Значення гіперпараметра γ може зменшуватися автоматично (так зване покрокове ослаблення) або підкорятися обраному закону.

Пакетний градієнтний спуск. Спочатку, як і вище, обчислюють логіти l_j: $l_j = b_j + (\vec{x} \cdot \vec{w}_j)$. Потім обчислюються ймовірності $p(a)$: $p(a) = \sigma_a(l) = e^{l_a} / \sum_j e^{l_j}$, а потім оцінка втрат – негативний натуральний логарифм ймовірності правильної відповіді: $X(\Phi, x) = -\ln p(a)$. Це був прямий прохід (forward pass). Ці дані були потрібні для застосування зворотного проходу коригування ваг (backward pass). Після того як були обчислені всі поправки (в даному циклі – епосі, яких сотні), тільки тоді проводиться корекція ваг. Цей алгоритм є досить повільним. Існує мініпакетний спуск, коли спостережень, розрахунків десятки.

Але важливо відзначити, що істотні не пакети і більш загальні, що охоплюють корекцію всіх параметрів мережі, так звані епохи, а загальна кількість оновлень ваг. Відзначимо, що часто саме в послідовності епох відбувається зміна темпу заучування – швидкості навчання. Хоча зміна (зменшення) швидкості навчання може бути задана у вихідному коді навчання (ступеневе і покрокове послаблення). Може бути задана формула, зменшення швидкості навчання в залежності від різних умов і параметрів.

АЛГОРИТМ НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ ЗА ДОПОМОГОЮ ОБЕРНЕНОГО ПОШИРЕННЯ ПОМИЛКИ

Таблиця 1

1	Подати на вхід вхідний вектор–сигнал і розрахувати всі виходи і входи	$s_j^{(q)} = \sum_{i=1}^{L} y_i^{(q-1)} \cdot w_{ij}^{(q)}$, L – число нейронів в шарі (q–1), вихідний нейрон шару (q) має стан +1, на його *i*-й вхід поданий, $y_i^{(q-1)} \cdot w_{ij}^{(q)}$, де $y_r^0 = x_r$ – вхідний сигнал. $E(w) = \frac{1}{2}\sum_{j,k}(y_{j,k}^{(Q)} - d_{j,k})^2$ – функція помилки, $y_{j,k}^{(Q)}$ – вихідний стан нейрона j вихідного шару при подачі на вхід k-го способу, $d_{j,k}$ – потрібний вихідний стан
2	Розрахувати для вихідного шару δ_j^Q	$\delta_j^Q = (y_j^{(Q)} - d_j) \cdot \frac{dy_j}{ds_j}$ для вихідного шару, За визначенням $\delta_j^q = \frac{dE}{dy_j} \cdot \frac{dy_j}{ds_j}$ $\frac{dE}{dw_{ij}} = \frac{dE}{dy_j} \cdot \frac{dy_j}{ds_j} C = (\frac{dE}{dy_j} \cdot \frac{dy_j}{ds_j}) \cdot y_i^{(q-1)} = (\delta_j^q) \cdot y_i^{(q-1)}$ $\frac{dE}{dy_j} = \sum_r \frac{dE}{dy_r} \cdot \frac{dy_r}{ds_r} \cdot \frac{ds_r}{dy_j} = \sum_r \frac{dE}{dy_r} \cdot \frac{dy_r}{ds_r} w_{jr}^{(q+1)}$
3	Розрахувати для інших шарів $\delta_j^{(q)}$ і поправку до вагового коефіцієнта $\Delta w_{ij}^{(q)}$	Отримаємо рекурсивну формулу * $\delta_j^{(q)} = [\sum_r \delta_r^{(q+1)} w_{jr}^{(q+1)}] \frac{dy_j}{ds_j}$ ** Тут синапс з j в r і з шару (q) в шар (q + 1)* $\Delta w_{ij}^{(q)} = -\eta \frac{dE}{dw_{ij}^{(q)}}$ – зміна вагового коефіцієнта або інакше $\Delta w_{ij}^{(q)} = -\eta \cdot \delta_j^{(q)} \cdot y_i^{(q-1)}$ $w_{ij}^{(q)}$ синапс між *i*-м нейроном шару (q–1) та *j*-м нейроном шару (q)
4	Скорегувати всі ваги	$w_{ij}^{(q)}(t) = w_{ij}^{(q)}(t-1) + \Delta w_{ij}^{(q)}(t)$

Додаткова термінологія

Mnist – набір даних для подання в двовимірному масиві 28x28 кольору і яскравості. 60 тис. навчальних зображень-міток і 10 тис. в наборі розробки і тестування.

National Institute of Standards (NIST) – національний інститут стандартів США.

Label – зображення-мітки.

Supervised learning – навчання з учителем

Fully (semi–supervised) supervised leaming – повне (часткове) навчання з **учителем.**

Unsupervised learning – навчання без учителя.

Classification problem – завдання класифікації, тобто з набору даних (ознак) ідентифікується сутність.

Dot product – **скалярний** добуток.

Примітивний персептрон – це нейрон з 784 входів для зображення цифри в матриці 28x28.

Параметри – вага кожного входу і зміщення (bias).

Набори – development set – розробка, held–out set – обраний набір, validation set – перевірки, test set – тестовий набір.

Feed–forward neural networ – нейронна мережа з прямим зв'язком.

Gradient descent – градієнтний спуск.

Multilevel perceptron – багатошаровий персептрон.

Zero–one loss – двійкова функція втрат

Loss function – функція втрат

Cross–entropy loss – функція крос–ентропійних втрат, наприклад $X(\Phi, x) = -\ln p_{\Phi}(a_x)$, тут логарифмічна ймовірність мітки *x*. Крос–ентропія (cross entropy) двох розподілів є мірою того, наскільки вони різні.

Softmax – перетворення набору чисел (Logit – логіт) – в розподіл ймовірностей $\sigma(x)_j = e^{x_j} / \sum_i e^{x_i}$. Для логіт $p(l_i) = e^{l_i} / \sum_j e^{l_j}$.

Метод найшвидшого спуску – $L \equiv E(w) = \frac{1}{2}\sum_{j,k}(y_{j,k}^{(Q)} - d_{j,k})^2$ **–** функція втрат

$$\Delta\phi_i = -\varepsilon\frac{\partial L}{\partial\phi_i} \quad .$$

Batch size – розмір партії (в стохастичному градієнтному спуску m = 20). Оновлення параметрів відбувається при використанні тільки при розмірі партії багато меншою числа навчальної вибірки. Чим менше розмір партії, тим менше швидкість навчання ε.

Bayes error – Байесова помилка – найменша помилка будь-якого класифікатора з випадковим кінцевим результатом, подібна до невиправної помилки.

long short – tenn memory,

LSTM – нейрон з довгою короткостроковою пам'яттю.

Стохастичний градієнтний спуск (stochastic gradient descent SGD). У цьому випадку відновлення відбуваються частіше, після кожного визначення градієнтів функції помилки, тому обчислювальних помилок більше – кажуть, в системі вищий рівень шуму, що має також і позитивні властивості: вибиває систему з локальних мінімумів, полегшуючи пошук глобального мінімуму функції помилки.

Це один з методів зворотного поширення помилки, який менш надійний, ніж пакетний. Розглянемо технологію на основі підбору b_j. Спочатку $\frac{\partial X(\Phi)}{\partial b_j} = \frac{\partial l_j}{\partial b_j} \cdot \frac{\partial X(\Phi)}{\partial l_j}$, тут $X(\Phi, x)$ – функція крос-ентропійних втрат, тобто логарифмічна ймовірність мітки *x*. Нагадаємо, крос-ентропія (cross entropy) двох розподілів є мірою того, наскільки вони різні. Очевидно, $\frac{\partial l_j}{\partial b_j} = \frac{\partial}{\partial b_j}[b_j + (\vec{x} \cdot \vec{w}_j)] = 1$. Тоді $\frac{\partial X(\Phi)}{\partial l_j} = \frac{\partial p_a}{\partial l_j}\frac{\partial X(\Phi)}{\partial p_a}$ і $\frac{\partial X(\Phi)}{\partial p_a} = \frac{\partial}{\partial p_a}(-\ln p_a) = 1/p_a$, отримаємо $\frac{\partial p_a}{\partial l_j} = \frac{\partial \sigma_a(l)}{\partial l_j} = \begin{cases}(1-p_j)p_a, a = j \\ (-p_j p_a), a \neq j\end{cases}$, $\frac{\partial X(\Phi)}{\partial l_j} = \frac{1}{p_a}\begin{cases}(1-p_j)p_a, a = j \\ (-p_j p_a) a \neq j\end{cases}$ та $\Delta b_j = \varepsilon\begin{cases}(1-p_j), a = j \\ -p_j, a \neq j\end{cases}$, а вже тоді $\frac{\partial X(\Phi)}{\partial w_{i,j}} = \frac{\partial l_j}{\partial w_{i,j}}\frac{\partial X(\Phi)}{\partial l_j}$. Потім $\frac{\partial X(\Phi)}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}}[b_j + (w_{1,j}x_1 + \ldots w_{i,j}x_i + \ldots)] = x_i$ і остаточно $\Delta w_{i,j} = -\varepsilon x_i \frac{\partial X(\Phi)}{\partial l_j}$.

Важливо відзначити, що оновлення параметрів мережі може відбуватися і при використанні партії (batch size), розмір якої набагато менше числа завдань навчальної вибірки (мабуть, через часті оновлення виникає випадкова помилка–шум, крім того через довільний вибір партій і визначили назву методу: «стохастичний»). Чим менше розмір партії, тим менше повинна бути швидкість навчання.

Подальший розвиток методів зворотного поширення помилки. Про моделі машинного навчання. Всі моделі, такі як ResNet, RNN, Normalizing flows, модифікують приховані стани виду $h_{t+1} = h_t + f(h_t, \theta_t)$. Якщо є вектор $z(t)$ і функція $f[z(t), t]$ задані мережею з параметрами θ і входом $z(t_0) = z_0$, при цьому виходом будемо вважати $z(t_1)$, то можна скористатися моделлю (нейродіффуром) $\frac{dz}{dt} = f_\theta[z(t), t]$ з умовами $z(t_0) = z_0$. Рішення представимо як $z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_\theta[z(t), t] \cdot dt$ або

$z(t_1) = output = ODESolver(z_0, f_\theta, t_0, t_1)$. Визначимо вартісну функцію або функцію втрат $L[z(t_1)]$. Для навчання градієнтними способами потрібно диференціювати за параметрами функцію динаміки f_θ. Метод зворотного поширення помилки вимагає обчислення $\frac{\partial L}{\partial \theta}$. Можна використовувати чисельний метод на сітці $t_0 = x_0 < ... < x_n = t_1$ для знаходження рішення в її вузлах $z_i = z_{i-1} + (x_i - x_{i-1}) f_\theta(z_{i-1}, x_{i-1})$, $i = 1, 2, ..., n$. Однак бентежить необхідність тримати в пам'яті градієнти по вузлах сітки, крім того, оскільки це наближене рішення оберненої задачі, то накопичується при диференціюванні наближеного виразу вже власна помилка, тому має сенс використовувати підхід, запропонований в роботі «Neural Ordinary Differential Equations» [4], представлений нижче в наступному розділі.

Апроксимація градієнтів без диференціювання наближеного рішення. Вводять пов'язані (adjoint) функції $a_z(t) = \partial L / \partial z(t)$, $a_\theta(t) = \partial L / \partial \theta(t)$.

Автори згаданої роботи показали справедливість наступних співвідношень:

$$\frac{da_z(t)}{dt} = -a_z(t)\frac{\partial f_\theta[z(t), t]}{\partial z}; \quad \frac{da_\theta(t)}{dt} = -a_z(t)\frac{\partial f_\theta[z(t), t]}{\partial \theta}. \tag{1.1}$$

Тут функція втрат $L[z(t)]$ залежить від $z(t)$ і $\theta(t)$ при $t < t_1$. Можна показати, що

$$a_z(t) = \frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t+\varepsilon)}\frac{\partial z(t+\varepsilon)}{\partial z(t)} + \frac{\partial L}{\partial \theta(t+\varepsilon)}\frac{\partial \theta(t+\varepsilon)}{\partial z(t)},$$

$$a_\theta(t) = \frac{\partial L}{\partial \theta(t)} = \frac{\partial L}{\partial z(t+\varepsilon)}\frac{\partial z(t+\varepsilon)}{\partial \theta(t)} + \frac{\partial L}{\partial \theta(t+\varepsilon)}\frac{\partial \theta(t+\varepsilon)}{\partial \theta(t)},$$

де $\frac{\partial \theta(t+\varepsilon)}{\partial z(t)}$ – нульова матриця, оскільки $\theta(t+\varepsilon)$ не залежить від $z(t)$, а $\frac{\partial \theta(t+\varepsilon)}{\partial \theta(t)}$ – одинична матриця, оскільки $\theta(t+\varepsilon) = \theta(t)$. Скориставшись розкладанням $z(t+\varepsilon) = z(t) + \varepsilon f_\theta[z(t), t] + ...$, запишемо

$$\frac{\partial z(t+\varepsilon)}{\partial z(t)} = I + \varepsilon \frac{\partial f_\theta[z(t), t]}{\partial z(t)} + ...$$

$$\frac{\partial z(t+\varepsilon)}{\partial \theta(t)} = \varepsilon \frac{\partial f_\theta[z(t),t]}{\partial \theta(t)} + \ldots$$

З огляду на $\frac{\partial L}{\partial z(t+\varepsilon)} = a_z(t+\varepsilon)$ і $\frac{\partial L}{\partial \theta(t+\varepsilon)} = a_\theta(t+\varepsilon)$, отримаємо важливі співвідношення

$$a_z(t) = a_z(t+\varepsilon)[I + \varepsilon \frac{\partial f_\theta[z(t),t]}{\partial z(t)} + \ldots]$$
$$a_\theta(t) = a_\theta(t+\varepsilon)[\varepsilon \frac{\partial f_\theta[z(t),t]}{\partial \theta(t)} + \ldots] + a_\theta(t+\varepsilon).$$

Переходимо до меж

$$\partial a_z(t)/\partial t = \lim_{\varepsilon\to 0} \frac{a_z(t+\varepsilon) - a_z(t)}{\varepsilon} = \lim_{\varepsilon\to 0}[-a_z(t+\varepsilon)] \cdot$$
$$\cdot \frac{\varepsilon \frac{\partial f_\theta[z(t),t]}{\partial z(t)}}{\varepsilon} = -a_z(t) \frac{\partial f_\theta[z(t),t]}{\partial z(t)},$$

а також

$$\partial a_\theta(t)/\partial t = \lim_{\varepsilon\to 0} \frac{a_\theta(t+\varepsilon) - a_\theta(t)}{\varepsilon} = \lim_{\varepsilon\to 0}[-a_z(t+\varepsilon)]$$
$$\cdot \frac{\varepsilon \frac{\partial f_\theta[z(t),t]}{\partial \theta(t)}}{\varepsilon} = -a_z(t) \frac{\partial f_\theta[z(t),t]}{\partial \theta(t)}.$$

Праві частини (1.1.) легко обчислюються в процесі налаштування. Це матриці похідних виходів нейромережі по входах і параметрах. До цих рівнянь є умови

$$a_z(t_1) = \frac{\partial L}{\partial z(t)}, \qquad a_\theta(t_1) = 0. \tag{1.2}$$

Оптимізатори [5]. Модифікація градієнтного спуску дозволяє поліпшити ефективність навчання. Модифікації, звані оптимізаторами, активно використовуються для машинного навчання. Найвідоміші з них: Momentum, RMSProp і Adam.

Розглянемо ***оптимізатор Momentum***. Вводиться так зване «експоненціально зважене середнє» деякої величини θ_i $(i=1,...n)$, де n – номер поточного значення: $\theta=\theta_n$: $v_n=(\theta*b)_n=\sum_{i=1}^{n}\theta_i\cdot b_{n-i}$. Особливістю цієї величини, тобто згортки послідовностей $b_k=(1-\beta)\beta^k$ (для $k=1,...n$, причому сума всіх членів цієї функції при $n\to\infty$ дорівнює одиниці) і θ_i, є наступна характеристика. Поточне значення $\theta=\theta_n$ в цьому ряді множиться на одиницю, а всі попередні значення θ_i входять в суму з (експоненціально) зменшеною вагою b_k. Тобто основний внесок в експоненціально зважене середнє дає поточне значення. Це середнє є також ковзним середнім членів ряду.

Якщо при градієнтному спуску ваги оновлювалися відповідно до $w_{m+1}=w_m-\gamma\frac{\partial J(\vec{w})}{\partial w_m}$, то при застосуванні експоненціального середнього рівняння для зміни ваги набуває вигляду $w_{m+1}=w_m-\gamma v_m$, де $v_{m+1}=\beta v_m-(1-\beta)\frac{\partial J(\vec{w})}{\partial w_m}$. До речі, поправки зсуву при цьому виявляються неістотними. Цей результат для поновлення ваг дозволяє враховувати всі попередні дані (тобто зберігається пам'ять про попередні ітерації).

Обмежена машина Больцмана (Restricted Boltzmann Machine, RBM або ОМБ). Обмежена машина Больцмана[14] довгий час була одним із найпоширеніших будівельних блоків глибоких імовірнісних моделей. ОМБ є двошаровою мережею, перший шар якої видимий, а другий прихований. При цьому кожен нейрон видимого шару пов'язаний з кожним нейроном прихованого шару і навпаки. Кількість нейронів у прихованому шарі зазвичай менше, ніж у вхідному. Вхідні дані, що надходять на вхід, передаються в прихований шар і повертаються (реконструюються). При цьому навчання відбувається без вчителя і полягає в тому, що ваги та зміщення підбираються таким чином, щоб розподіл, що породжується прихованими змінними на видимих, був якомога більше схожим на розподіл вхідних даних.

[14] Перша обмежена машина Больцмана була побудована в 1986 р. Полом Смоленськи під назвою Harmonium [6], але набула популярності тільки після винаходу Хінтоном швидких алгоритмів навчання в середині 2000-х років.

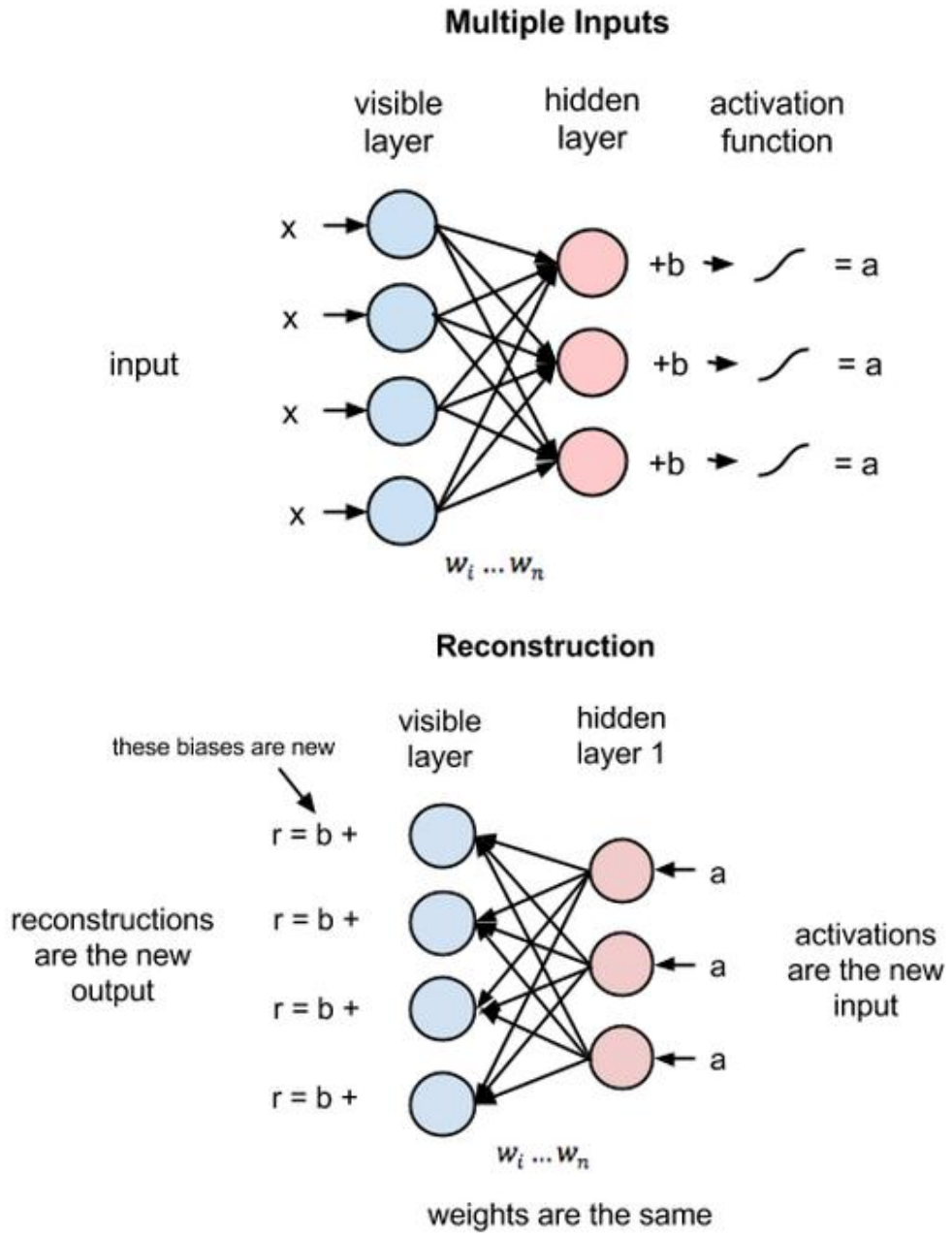


Рис. 4. Архітектура обмеженої машини Больцмана (а) та процес реконструкції вхідних даних (б)

Таким чином, ОМБ є алгоритмом, корисним для зменшення розмірності вхідних даних та їх попередньої класифікації. Але все ж таки основне призначення ОМБ і подібних алгоритмів – це підбір початкових значень ваг і зміщень, що дозволяє при подальшому глибокому навчанні методом зворотного поширення помилки набагато швидше приходити до результату.

Алгоритм, який таке навчання реалізує, одержав назву contrastive divergence[15]. Цей алгоритм теж придуманий Хінтоном, причому ще

[15] В україномовній та російськомовній літературі цей термін не отримав загальноприйнятої назви.

2002 року, до початку революції глибокого навчання, а 2005 року було сформульовано ключову ідею навчання: розпочати з contrastive divergence, щоб навчити початкову точку, та донавчити результат за допомогою звичайного навчання з вчителем. Так само можна побудувати і глибоку модель: спочатку навчити перший шар як обмежену машину Больцмана, потім використовувати приховані вершини першого шару як видимий шар другого рівня, потім так само з третім і т.д., а в кінці об'єднати все в одну велику модель з кількома шарами, використовувати ваги машин Больцмана як початкове наближення та донавчити результат з учителем. Така конструкція одержала назву глибокої машини Больцмана (Deep Boltzmann machine, DBN).

Технологія застосування (алгоритм): Вводиться деяка функція T, яка визначає «температуру». Взагалі кажучи, евристична. 1). На вхід подається великий набір початкових векторів, визначаються виходи мережі для них і на основі цих виразів конструюють цільову функцію C. 2). Міняють вагу в одній зв'язці і знаходять зміну всіх величин і відповідно цільової функції. Якщо цільова функція C зменшилася, ця вага зберігається. Якщо ні, то збереження визначається з імовірністю P (C) = exp (–C / kT), де k – константа. Для випадкового числа r (від нуля до одиниці), при P (C)> r**,** вага зберігається, інакше повертаємося до старого значенням. Процедура перебору ваг відбувається при постійному зменшенні «температури», для зменшення цільової функції. Процедура повторюється для всіх векторів навчальної вибірки. Цей метод вимагає великого часу, але якщо він відпрацьований, то виявляється вельми ефективним.

Ініціалізація Ксав'є. Повноцінне навчання за допомогою обмежених машин Больцмана в сучасній практиці зустрічається рідко, але все одно дуже важливо ініціалізувати ваги правильно. І основним результатом [7] став простий і добре мотивований спосіб ініціалізації ваг, що дозволяє суттєво прискорити навчання та покращити якість. На честь одного з авторів його часто називають ініціалізацією Ксав'є (Xavier initialization).

Основна ідея ініціалізації ваг Ксав'є полягає в спробі зробити дисперсію вихідних даних шару рівною дисперсії його вхідних даних. Звичайно, ця ініціалізація залежить від функції активації шару. В оригінальній роботі [7] як функція активації використовувався логістичний сигмо ід.

Пізніше виявилося, що активатор ReLu підходить краще, оскільки дозволяє вирішити проблему зникаючих та вибухових градієнтів. В результаті з'явився новий метод ініціалізації, який використовував ту ж ідею (балансування дисперсії активації) до цієї нової функції активації. Він був запропонований Каймінгом Хе [8], і тепер його часто називають ініціалізацією Хе.

Література до розділу 1

1. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика – 2 изд. – М.: Горячая линия – Телеком, 2002. – 382 с.

2. Pennington J., Socher R., Manning C. D. GloVe: Global Vectors for Word Representation https://nlp.stanford.edu/pubs/glove.pdf.

3. Микелуччи У. Прикладное глубокое обучение. Подход к пониманию глубоких нейронных сетей на основе метода кейсов: Пер. с англ. – СПб.: БХВ–Петербург, 2020. – 368 с.: ил.

4. Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. CoRR, abs/1806.07366, 2018.

5. S. Ruder "An overview of gradient descent optimization algorithms" ("Обзор оптимизационных алгоритмов градиентного спуска"), см. https://goo.gl/KgKVgG.

6. Smolensky P. Information processing in dynamical systems: Foundations of harmony theory. – Colorado Univ at Boulder Dept of Computer Science, 1986.

7. Glorot X., Bengio Y. Understanding the Difficulty of Training Deep Feedforward Neural Networks // International conference on artificial intelligence and statistics, 2010. – P. 249–256.

8. He K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification //Proceedings of the IEEE international conference on computer vision. – 2015. – С. 1026-1034.

РОЗДІЛ 2. НАВЧАННЯ БЕЗ УЧИТЕЛЯ

Когнітивний психолог (нейропсихолог) Дональд Хебб, який запропонував навчання мережі без учителя (причому не без налаштування мережі деякою програмою), вважав, що зв'язок між двома нейронами повинен змінюватися в залежності від частоти їх збудження (як у природних нейронних мережах). При цьому вага їх синапсів (зв'язки) зростає. При процедурі навчання без учителя мале початкове значення ваг синапсів при цьому виді навчання замінювалося керуючою програмою налаштування на зважене – множину вхідного і вихідного сигналу такого зв'язку. Тобто була обрана така реалізація правила Хеба. Після зменшення відхилень вихідного сигналу мережі до деякого заданого малого значення вважали, що мережа налаштована.

Т. К. Кохонен використовував модифікацію ваги, представляючи її у вигляді попереднього значення, до якого додавав зважену різницю між вихідним сигналом, що надходить від попереднього шару, і значенням ваги. Ці види формування ваги синапсів також слідують формально правилу Хебба (див. «Правила навчання»), хоча і здаються досить довільними, але, тим не менше, їх застосування виявилося ефективним.

Формування понять – кластерів у вихідному шарі. Кластеризація – це метод навчання без учителя, в якому вимірюється відстань між об'єктами і схожі об'єкти збираються разом. Зрештою об'єкти, що концентруються навколо n центроїд, вважаються належними одній групі – кластеру. Метод K–середніх – один з найвідоміших алгоритмів кластеризації в машинному навчанні.

Мережа на виході, використовуючи навчальну вибірку, формує кластери рішень, які відображають певні образи, поняття, висновки і т. п. Ці кластери являють собою згущення в просторі вихідних значень мережі (і містять невелику кількість збуджених нейронів у вихідному шарі). Далі при поданні на вході мережі нового образу, рішення на виході може потрапити в околицю одного з вже сформованих кластерів. Тим самим мережа віднесе це рішення до виділеного експертами класу. Іншими словами, навчальна вибірка нав'язує набір понять, висновків, і від неї потребується віднести новий образ до одного зі сформованих кластерів або класів[16]. Тобто в цьому випадку цікавим є поділ даних на виході за

[16] Забігаючи наперед, відзначимо, що мережі можуть бути породжуючі і дискримінантні. Породжуючі навчаються спільного розподілу ймовірності входу x і виходу y, тобто P (x, y). Дискримінантні визначають умовну ймовірність P (x | y), тут вихід тільки класифікує вхід.

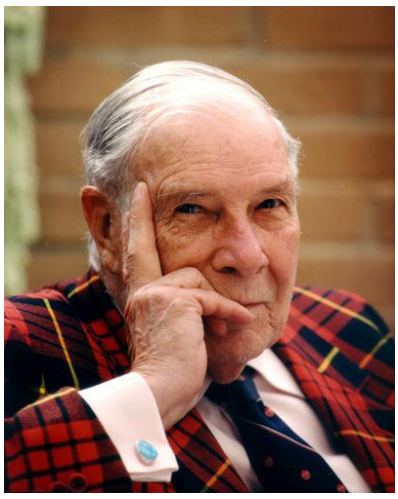
класами-кластерам, тому цей метод налаштування мережі полягає в більш чіткому виділенні класів, тобто в класифікації. Збираються в класи-кластери вихідні дані і забезпечують процес кластерізації.

Зведення вихідного кластера виходу до одного нейрона. Тобто якщо в мережі не сформована експертами за рахунок застосування навчальної вибірки прикладів система кластерів (тобто відмовилися від навчання з учителем), мережа може самостійно сформувати набір кластерів (класів) рішень. Однак при цьому доведеться розбиратися, які поняття і уявлення відображає кожний кластер. Для прискорення створення класифікації образів, представлених на вхід мережі, кожен образ часто пов'язують з нейроном вихідного шару (його спочатку потрібно знайти), що отримав на своєму вході максимальний сигнал, і домагаються (за допомогою керуючої налаштуванням програми) на його виході максимуму. При цьому пригнічують всі вихідні сигнали його сусідів, для цього послаблюють всі їх синапси. Цей метод навчання прискорює в структурі мережі створення системи образів. Потім з ними можна буде порівнювати нові вхідні дані.

Автоматичне виділення нейрона-переможця у вихідному шарі. Процес виділення «нейрона-переможця» може відбуватися і без втручання користувача (тобто керуючої програми користувача): нейрони, які отримали на вхід сильний сигнал, звичайно за замовчуванням пов'язані з сусідами латеральними (бічними) зв'язками, що гальмують та пригнічують сусідів.

Крім того, всі нейрони мають позитивні зворотні зв'язки зі свого виходу на свій вхід. Співвідношення між цими зв'язками і їх інтенсивність підбирають при формуванні мережі.

Розглянемо найбільш відомі ***метод Хебба і алгоритм Кохонена.***



Річард Уеслі Хеммінг – математик зі США (1915–1998)

Реймонд Курцвейл – вчений-винахідник зі США (1948 р. нар.)

СИГНАЛЬНИЙ МЕТОД ХЕББА І АЛГОРИТМ КОХОНЕНА НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ БЕЗ УЧИТЕЛЯ

Таблиця 2

1	Ініціалізація: всім ваговим коефіцієнтам привласнюють малі випадкові значення	
2	На вхід подається вхідний сигнал, і в обсязі мережі на вхід нейрона подається зважена сума, до якої застосовується його активаційна функція	$y = \varphi(\sum_i w_i x_i)$ активаційна функція, для нейронів вхідного шару, аргумент активаційної функції для нейронів наступних шарів $s_j^{(q)} = \sum_{i=1}^{L} y_i^{(q-1)} \cdot w_{ij}^{(q)}$
3	Відбувається зміна вагових коефіцієнтів 1. Використовуйте попередню нормалізацію вхідних образів і вагових коефіцієнтів, наприклад $x_i \to x_i / \sqrt{\sum_{j=1}^{n} x_j^2}$. 2. Частку вхідних образів у початковий момент роблять рівними, а потім поступово вони набувають характеру, що прискорює навчання	$w_{ij}(t) = w_{ij}(t-1) + \alpha y_i^{(q-1)} y_j^{(q)}$ – метод Хебба (1) $w_{ij}(t) = w_{ij}(t-1) + \alpha \cdot [y_i^{(q-1)} - w_{ij}(t-1)]$ – алгоритм Кохонена (2) з акредитацією – $\min D_j$ нейрона, де $D_j = \sqrt{\sum_{i=1}^{R} (y_i^{(q-1)} - w_{ij})}$. Цей нейрон і його сусіди в околиці (яка зменшується в процесі навчання) навчаються за правилом (2). Нейрони-переможці можуть виключатися з мережі для зменшення дискредитації інших нейронів. $y_j^{(q)}$ – вихід нейрона j шару (q)
4	Далі цикл з кроку 2, поки не стабілізуються вагові коефіцієнти. Потім на вхід пред'являють інші образи, відповіді на які невідомі	При тестуванні визначається топологія кластерів вихідного шару

РОЗДІЛ 3. ОСОБЛИВОСТІ ОПТИМІЗАЦІЇ МЕРЕЖ

Проблеми простих нейронних мереж. При експлуатації різноманітних мережевих рішень з'ясувалося, що серйозні проблеми виникли з навчанням: знадобилася велика кількість правильно вирішених завдань-прикладів (для цього навіть створювали колективи розробників таких прикладів). При збільшенні кількості шарів вплив коригувальних поправок прогнозовано послаблюється, слід було проводити багато операцій з налаштування – в режимі ітерацій з поверненнями. Виявилося, наприклад, корисним попередньо готувати мережу для обробки даних певного типу – метод попереднього навчання мережі Д. Хінтона (запропонований у 2006 р.).

Якщо після навчання все нові тести показували, що помилки стали допустимо менше, то навчання вважали успішним. Якщо ж виявляли після навчання помилки на тестових прикладах, значить, мережа просто запам'ятала все, що їй повідомили під час навчання, а реально нічому не навчилася, що політкоректно назвали перенавчанням (overfitting). Вимога математиків домагатися єдності рішення здавалася в ці часи непорушною, і на неї орієнтувалися творці нейронних мереж, як, втім, і всі розробники обчислювальних систем і систем штучного інтелекту.

Перенавчання великих мереж було очікувано, тому що збільшення вільних керуючих параметрів еквівалентно збільшенню ступенів свободи. Недостатнє число навчальних завдань дозволяє забезпечити правильність необхідних рішень цих завдань. Однак інші подібні завдання могли бути вирішені неправильно. Крім того, не можна було забезпечити пряму відповідність початкових умов кінцевими результатами. Невеликі зміни початкових умов могли приводити до якісних змін рішень (одна з форм некоректності завдань). Боротьба з перенавчанням виявилася складною, але деякі підходи виявилися вдалими. Наприклад, часто застосовували ослаблення впливу сусідніх нейронів один на одного (для кількісних критеріїв такого ослаблення використовували відомі в математиці методи регуляризації). Можна було навіть відключати під час навчання великі ділянки мережі – оригінальна методика навчання Дж. Хінтона Dropout (2012). Це тимчасово зменшувало складність мережі, створюючи умови ослаблення перенавчання, а потім при підключенні спочатку виключених ділянок їх адаптація до нового стану поліпшувалася. Дійсно, людський мозок знайшов же спосіб боротися з ефектом перенавчання у формі отримання безлічі відповідей на просте запитання (нехай навіть не так успішно, як хотіли б математики), відключаючи цілі відділи кори головного

мозку, що не потрібні для вирішення простого завдання. Але якщо все це не допомагало, тоді мережу вважали нездібною і відмовлялися від неї.

Хоча всі усвідомлювали, що є й інша причина появи несподіваних і множинних відповідей. Коли мережа досить потужна, вона здатна вийти з нав'язаного їй програмою навчання класу рішень і запропонувати свої, дещо інші, які експериментатори не припускали. Для них мережа порушує сформовану картину світу[17]. Така кмітливість мережі, обговорювана ще М. Мінським, теж не віталася, і від цієї надмірно розумної мережі теж відмовлялися. Каменем спотикання залишалася вимога математиків домагатися єдності рішення (тобто рішення повинно бути тільки одне). Хоча вони самі вже зіткнулися з неінтегріруємістю, тобто неможливістю отримання єдиного рішення так званих необчислюваних завдань, тобто які не можливо обчислити.

Допоміг вийти із замкнутого кола (єдність рішень, недостатнє число навчальних прикладів обмежували складність і обсяги мереж) прогрес в розвитку систем обчислення та сміливість в організації та оптимізації мереж, яка значно підвищилась. Недостатні обсяги і недостатня складність не дозволяли збільшувати пам'ять і вирішувати складні завдання [1]. Мало обізнані в математиці інженери і технологи, що не вірили у важливість вимоги єдності рішення, створили нейронні мережі з величезним числом елементів – нейронів і синапсів. Навчивши ці мережі усвідомлювати природну мову і тексти (в більшій частині англійські, де мова досить структурована, чого вартий, наприклад, порядок слів у реченні), вони виявили здатність мережі вирішувати завдання, які не можливо коректним чином обчислити. Вирішити їх вдалось спираючись на величезні запаси попередньо інформації, яку мережа засвоїла. Те, що такі рішення подібних задач не відповідають критерію єдності, цих людей не турбувало, оскільки, вирішуючи такі завдання, людина сама отримує безліч варіантів рішення, з яких вона вибирає лише таке, що їй здалося привабливим.

---

[17] У книзі «Про користь роздумів» була висловлена гіпотеза «необхідного розвитку». Будь-яка інтелектуальна система (зокрема цивілізація), що еволюціонує, весь час збільшує свої інтелектуальні ресурси (тобто підключаються все нові і нові блоки інтелектуальної мережі), зростає її ускладнення. І колишній обсяг навчальних завдань і відповідно запропонованих рішень незабаром виявляється недостатнім. В тому сенсі, що при цьому виникають проблеми з формуванням певних рішень, підвищуються неузгодженості, які сприймаються як помилки. Тобто порушується стійкість картини світу. Єдиним виходом тут може бути збільшення обсягу відомих задач, що забезпечують ефективне навчання глобальної інформаційної системи. Це збільшення може бути підтримано саме наукою, виходом із замкнутого кола колишніх понять і уявлень.

Але не так все просто[18]**.** Зі збільшенням числа нейронів та їх зв'язків- синапсів виникли проблеми з навчанням і налаштуванням мереж. Крім використання технологій ослаблення і придушення впливу сусідніх нейронів один на одного, почали використовувати відключення під час навчання великих ділянок мережі – оригінальна методика навчання винахідливого Дж. Хінтона ***Dropout*** (2012). Це тимчасово зменшувало складність мережі, а потім при підключенні перш виключених ділянок поліпшувалася їх адаптація до нового стану. Та й наш мозок, відключаючи цілі відділи кори, непотрібні для вирішення простих завдань, часто спрощував вибір нашої поведінки. К. Фукусіма (1975) навіть створив мережу – ***когнітрон***, де зв'язки організовані подібно зоровій корі головного мозку.

При цьому нейрони з наступного шару пов'язані лише з обмеженою областю попереднього шару (областю зв'язку), але нейрони вихідного шару підключені так, щоб реагувати на повне вхідний поле.

Проблеми недостатнього навчання. Для вирішення цього завдання використовують алгоритми оптимізації і помилок. Кожен даний набір ваг передбачає деяку гіпотезу зв'язку вхідних сигналів з виходом мережі (формуванням міток у просторі рішень). Пошук потрібного визначення ваг пов'язаний з алгоритмами оптимізації та помилок (такими як градієнтний спуск[19]). Значення ваг – це параметри, які потрібно оптимізувати.

Для налаштування навчання використовують заздалегідь налаштовані і незмінні гіперпараметри (наприклад, швидкість змін в оптимізації ваг-синапсів). Таким чином, зміна параметрів – ваг відбувається при мінімізації помилки на навчальній вибірці, причому завдання оптимізації полягає у виборі напрямку процесу для зменшення помилки.

Проблеми навчання з зайвою точністю (іноді це називають перенавчанням – тут це дуже мала частота помилок на навчальному масиві – зайва (як виявилося) точність оптимізації). Хоча це явище не цілком відповідає раніше визначенню overfitting. Досвід показує, що в цьому випадку часто спостерігаються помітні помилки на генеральній сукупності. Це призводить до необхідності послабити ступінь наближення результатів до навчальної вибірки, дати деякий лаг, відхилення від зайвої

[18] Джеффрі Хінтону репортери приписують таке зауваження: «На більшості конференцій говорять про введення невеликих змін замість того, щоб гарненько подумати і запитати себе: “Чому те, що ми робимо зараз, не виходить? З чим це пов'язано? Давайте зосередимося на цьому”».

[19] Характеристиками спуску є якобіан-матриця перших часткових похідних по векторах і гессіан-матриця подібних похідних другого порядку по векторах.

точності. Цю проблему, як і проблему перепідгонкі, вирішує регуляризація.

Перепідгонка [2]. Під час навчання або налаштування мережі виникає шум, і він сприймається мережею як елемент налаштування та входить в результат навчання. Іншими словами, мережа сприймає шум і обчислювальні помилки як елемент навчального набору. Боротьба з цим явищем також непроста, і зазвичай навчальну множину розбивають на кілька фрагментів.

І оцінка за допомогою порівняння показників кожного фрагмента дозволяє принаймні інтуїтивно зрозуміти, що не так і спробувати краще узгодити структуру навчання.

Регуляризація. Одним з методів боротьби з перепідгонкою і навіть з перенавчанням (мається на увазі зайва за точністю оптимізація) в наведеному вище сенсі є регуляризація. Цей термін погано визначено, але він виконує безліч функцій, серед яких, як не дивно, ослаблення близькості налаштування до тренувального набору даних. Виявляється, практика показала, що зайва близькість до результатів обчислення мережі до цього набору послаблює можливості роботи мережі на нових даних.

Для цього, наприклад, додають регуляційний член до вартісної функції, тобто до функції помилки. Розглянемо популярну регуляризацію.

Введемо норму вектора $\vec{x}$ з компонентами x_i, такими що $\| \vec{x}_p \| = \sqrt[p]{\sum_i | x_i |^p}$, для функції помилки використовуємо середньоквадратичну помилку (MSE): $J(w) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$, тут виміряна змінна y_i, а потрібне її значення $\hat{y}_i$, m – кількість спостережень. Уточнена функція помилки $\tilde{J}(w) = J(w) + \frac{\lambda}{2m} \| \vec{w} \|_2^2$, де λ – регуляризаційний (гіпер) параметр. При цьому $w_{i,n+1} = w_{i,n} - \gamma \frac{\partial \tilde{J}(\vec{w}_n)}{\partial w_i} = w_{i,n} - \gamma \frac{\partial J(\vec{w}_n)}{\partial w_i} - \frac{\gamma\lambda}{m} w_{i,n}$, причому тут використано співвідношення $\frac{\partial \| \vec{w} \|_2^2}{\partial w_i} = 2w_i$. Отримаємо нове рівняння для поновлення ваг $w_{i,n+1} = w_{i,n}(1 - \frac{\gamma\lambda}{m}) - \gamma \frac{\partial J(\vec{w}_n)}{\partial w_i}$. Це зрушує ваги до нуля, що допомагає в боротьбі з перепідгонкою і навіть з перенавчанням.

Трансферне навчання[20] полягає у використанні частини донорської навченої мережі, звідки беруться шари (зазвичай кілька перших) і підставляються згідно з розробленою технологією в розроблену мережу. Використовуються деякі протоколи для оцінки нейронних мережевих рівнів (шарів) для такого перенесення (трансферту) навчання.

Оптимальне керування мінімізує міру поведінки системи. В кінці 1950-х років Р. Беллман з колегами, розвиваючи теорію Гамільтона–Якобі (XIX століття)[21], розглянув поведінку системи спільно зі з'ясуванням оцінки цієї поведінки шляхом визначення функції цінності (опімальної плати, витрат). Функціональне рівняння, що описує цей процес, назвали рівнянням Беллмана (R. Bellman).

Методи вирішення оптимального управління назвали ***«динамічним програмуванням»***. Можна використовувати стохастичний варіант опису оптимального управління – т. зв. марковські методи (процеси) прийняття рішення. Розроблено метод ітерацій (наближень) у виборах стратегій (Howard 1960). Проблемою динамічного програмування є експоненціальне зростання обчислень з ростом змінних.

Навчання з підкріпленням. В основі лежить метод проб і помилок[22]. Це може бути представлено як «закон впливу» (Thorndike, 1911) на навчання найбільш вдалих операцій взаємодії з середовищем. Цей процес є вибірковим (відбір альтернатив поведінки) і асоціативним (удачі зв'язуються з даними подіями). З іншого боку, присутні пошук – перевірка різних варіантів і пам'ять – запам'ятовуються найкращі ситуації.

Була представлена «виборча самоналагоджувальна адаптація» – форма навчання, що використовує сигнали про успіхи та невдачі, замість навчальних прикладів (Widrow, Gupta, Maitra, 1973). Інтерес до методів навчання з підкріпленням (метод проб і помилок) відродив Харрі Клопф

---

[20] Обговорення цього методу проведено, наприклад, в роботі [3]. А практичні методи застосування цього підходу розвинули автори статті [4].

[21] У класичному варіаційному численні і аналітичній механіці, в яких завдання знаходження екстремумів (інтегрування гамільтонової системи рівнянь) зводиться до рівнянь з частковими похідними 1-го порядку. Основи були розроблені У. Гамільтоном (W. Hamilton) в 20-х рр. XIX ст., а в 1837 р. К. Якобі (C. Jacobi) застосував цей метод для загальних завдань класичного варіаційного числення.

[22] У 1950-х роках цей підхід обговорювали (A. Turing), були розроблені стохастичний нейро-аналоговий калькулятор з підкріпленням (SNARC – Stochastic Neural-Analog Reinforcement Calculation – автор M. Minsky). Відома система STeLLA (автор J. Andreae). Зниження інтересу і перемикання дослідників на методи навчання з учителем було пов'язано, мабуть, з тим, що метод проб і помилок був хороший для просунутих «учнів», новачкам потрібно було вчити «з нуля».

(Klopf, 1972–1982), який звернув увагу на важливість наполегливості в досягненні мети управління зовнішнім середовищем. До 1989 року всі найцікавіші напрями навчання на основі проб і помилок спробувала об'єднати в Q-навчанні дисертація К. Уоткінс (Chris Watkins), у чому йому, мабуть, допоміг П. Вербос (P. Werbos, 1977–1987). Успіх програми (G. Tesauro, 1992) для гри в нарди ще більш посилив інтерес до цього напрямку.

Оптимальне керування – розробка поведінки системи, що мінімізує функцію або функціонал, що описує динамічну систему. Для цього визначають стан динамічної системи і функцію цінності (плати, мабуть, за отримання цього стану). Рівняння, що зв'язує ці величини, в досить загальному випадку отримано Р. Беллманом (використав підхід до опису динаміки механічної системи на основі теорії Гамільтона–Якобі) і носить його ім'я. Алгоритми рішення цього рівняння назвали динамічним програмуванням. Причому для спрощення опису стохастичного варіанта управління (тобто пошуку потрібної поведінки системи) пам'ять про попередні події не використовується, тобто все розглядається в рамках марківського процесу. Однак навіть такий підхід для складних систем вимагає великого числа обчислень. Збільшення розмірності системи призводить до вибухового зростання операцій, названого «прокляттям розмірності». Виграш реальний або очікуваний в цінності або оплаті є механізмом вибору потрібного рішення, або точніше поведінки. Це є істотною ідеєю навчання поведінки методом проб і помилок. Взагалі кажучи, до цього методу можна додати перевірку і відбір альтернатив (вибірковість або пошук), і якщо альтернативи пов'язувати з конкретними навчальними ситуаціями, то можна вважати його асоціативним, що дозволяє формувати пам'ять. Саме зв'язок між пошуком і формуванням пам'яті про успіхи і є основою методу навчання з підкріпленням [5]. Важливо, що машині дають можливість самій шукати потрібну систему поведінки, вибираючи з альтернатив найкращу і запам'ятовуючи зв'язок між цим вибором і конкретною ситуацією. М. Мінський в докторській дисертації 1954 р. описав машину (SNARC), що вирішує завдання на основі методу проб і помилок, фактично використовуючи ідею навчання з підкріпленням. А пізніше в 1961 році, коли вже терміни навчання з підкріпленням визначилися, сформулював питання: «Як розподілити значення коефіцієнтів, що описують впевненість в успіху, між багатьма можливими варіантами вирішення». Пошуком відповідей на це питання зараз займаються теорія і практика навчання з підкріпленням.

Бум інтересу до глибокого навчання нейронних мереж, який збігся з ростом обчислювальних можливостей, зокрема використання графічних процесорів, дозволив даному напрямку отримати безліч відчутних результатів. Автомобілі з автопілотом від Tesla, автоматичні системи посадки космічних ракет від SpaceX, людиноподібні роботи від

BostonDynamics, AlphaGo для гри го від DeepMind – всі ці результати стали логічним розвитком алгоритмів навчання з підкріпленням для вирішення рівняння Беллмана для всіляких комбінацій параметрів.

Література до розділу 3

1. Куклин В. М. Особенности развития искусственного интеллекта на современном этапе // Вісник Харківського національного університету імені В.Н. Каразіна, серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління», 2018 С. 34–40.
https://periodicals.karazin.ua/mia/article/view/12563/11962

2. Bumham, K. P., Anderson, D. R. Model Selection and Multimodel lnference – 2nd ed. – New York; Springer–Verlag, 2002.

3. Sinno Jialin Pan and Qiang Yang. A survey on transfer learning // IEEE Transactions on knowledge and data engineering, 22(10):1345–1359, 2010.

4. Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Advances in neural information processing systems, pages 3320–3328, 2014.

5. Саттон Р. С., Барто Э. Г. Обучение с подкреплением. М.: Бином. Лаборатория знаний, 2014. – 402 с.

Розділ 4. НАБОРИ ТРАДИЦІЙНИХ ІДЕЙ, ЯКІ ВИЗНАЧИЛИ РОЗВИТОК НЕЙРОННИХ МЕРЕЖ

Про кількість нейронів у шарах і в цілому в мережі. Важливим параметром в рішенні проблеми є число заучувальних параметрів (ваг і зміщень у шарі m Q_m і в мережі $Q = \sum_m Q_m$) [1]. Підбір архітектури бажано проводити на малих кількостях нейронів в мережі. Лише тоді можна починати міняти Q. Проблемою часто є невелике число нейронів або точніше Q_m в окремому шарі. Рекомендують звернутися до досвіду розробників, який представлений в мережі Інтернет.

Зміни числа нейронів у прихованих шарах. У звичайних нейронних мережах іноді слід підбирати загальне число нейронів.

1. ***Алгоритми скорочення*** (pruning algorithms) – видаляють синапси (обнуляють вагові коефіцієнти, які потрапили в даний інтервал значень).

2. ***Конструктивні алгоритми*** (constructive algorithms) – додаються нейрони. Замість одного ставлять два (розщеплення – splitting). Причому вагові коефіцієнти нового нейрона рівні відповідним коефіцієнтам старого з додаванням шуму. Рекомендують значення зв'язків виходів нових нейронів з нейронами наступних шарів зменшити вдвічі (варіант регуляризації).

Прабатьки нейронних мереж – персептрони. Ці прабатьки нейронних мереж складалися з суматорів з багатьма входами-синапсами, на які подавався вектор вхідного сигналу, кожен компонент якого множився на ваговий коефіцієнт.

Якщо ця сума виявлялася більше деякого порога, вихід приймав значення одиниці, інакше – нуль. Персептрон (perceptron) спочатку був задуманий для двійкової класифікації (нуль або одиниця) – binary classification problem.

Розглянемо найпростіший варіант персептрона [2]. Нехай він складається з одного нейрона, який описаний вектором ваг (weight) $\vec{w} = [w_1, ...w_n]$ по одному на кожен вхід, і спеціальної ваги b, званої зміщенням (bias). Тут $\Phi = (\vec{w} \cup b)$ _ параметри (parameters) персептрона. При цих параметрах персептрон обчислює функцію $f_\Phi(x)$, яка дорівнює одиниці при $b + \sum_{i=1}^{l} x_i \cdot w_i > 0$ (ці вирази називають лінійними блоками, логітами) і нулю в іншому випадку. Розглянемо алгоритм персептрона (perceptron algorithm) – для навчання персептрона набору навчальних

прикладів (training example). Вказано приклад, що обговорюється з верхнім індексом, тому введення для k-го прикладу буде $\vec{x}^k = [x_1^k, ...x_l^k]$ і правильна відповідь (мітка) – a^k. Для двійкового класифікатора, такого як персептрон, відповіддю є 1 або 0, що вказує на членство в класі або його відсутність (при класифікації за n класами відповіді будуть від 0 до n-1). Навчання тут є по суті апроксимацією функцій (function approximation). Зазвичай існує три набори прикладів завдань. Перший – це навчальний набір (training set), що використовується для налаштування параметрів моделі. Другий – це набір розробки (development set), що використовується для перевірки моделі, коли ми намагаємося її покращити (вибраний набір held-out set або набір validation set). Третій – це тестовий набір (test set). Властивість алгоритму персептрона: якщо існує набір значень параметрів, який дозволяє персептрон правильно класифікувати весь навчальний набір, алгоритм гарантовано його знайде (що в загальному випадку для нейронних систем не характерно). Псевдокод цього алгоритму представлений нижче:

1. Встановити b та всі $\vec{w}$ в 0.
2. Для N ітерацій або до тих пір, поки ваги не перестануть змінюватися

(а) для кожного навчального прикладу x_1^k з відповіддю a^k

і. якщо a^k - f(x} = 0
продовжити

іі. в іншому випадку для всіх ваг w_i ; $\Delta w_i = [a_i^k - f(x_i^k)]x_i$.

Перший із двох рядків визначає: якщо виведення персептрона має правильну мітку, нічого робити не потрібно. Другий вказує, як змінити вагу w_i.

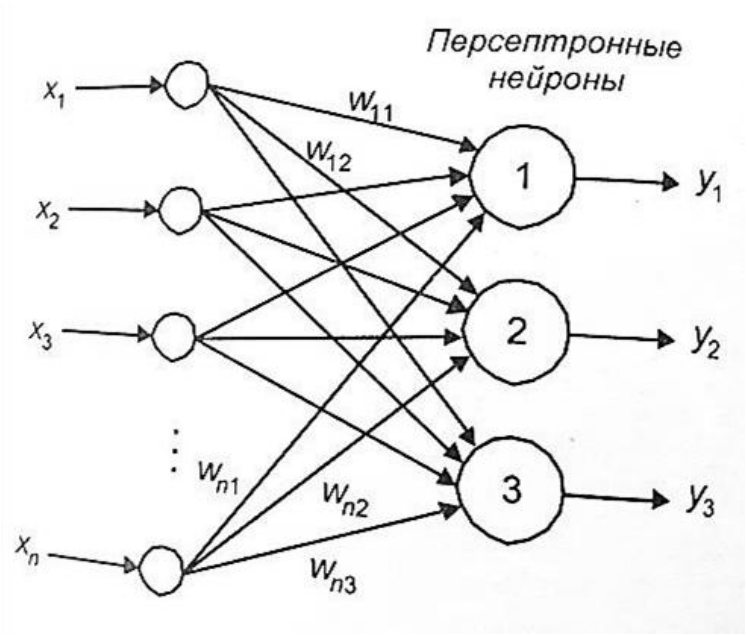


Рис. 5. Персептрон Ф. Розенблатта (див. [5])

Ф. Розенблатт показав, як навчати персептрони. В. Уїдроу продемонстрував це на практиці. Але пізніше М. Мінський цілком обґрунтовано вказав на суттєві обмеження щодо застосування таких одношарових нейронних систем, що спровокувало втрату надовго інтересу до нейронних систем [3] (т. зв. «перша зима»).

Одношаровий персептрон (single layer perceptron) [4, 5][23].

Якщо відмовитися від обмежень двійкової класифікації, то можливості налаштування збільшуються. Після ініціалізації (малі випадкові синаптичні ваги і зміщення b) на вхід персептронного (має поріг передаточної функції) нейрона пред'являється вектор з компонентами x_i і потрібно отримати на виході d. На виході визначають вихідний сигнал і отримують $y(t) = \varphi(\sum_{i=1}^{N} w_i(t)x_i - b)$. Налаштування ваг полягає в процедурі $w_i(t+1) = w_i(t) + \eta[d - y(t)] \cdot x_i$. Тобто поправка до ваги в процесі налаштування $\Delta w_i = \eta \frac{\partial L}{\partial w_i} = \eta \cdot [d - y(t)] \cdot x_i$, де L – функція втрат ($\eta$ – learning rate). При повному налаштуванні ваги вже не змінюються.

Корисно використовувати функцію крос-ентропійних[24] втрат (cross–entropy loss). Якщо вихід серед всіх значень має найбільшу можливу величину, то можна ввести розподіл ймовірності цієї величини (probability distribution). Але перш ніж це робити, потрібно вирішити, як бути з негативними значеннями виходу. Тут краще використовувати функцію, яка перетворює значення виходу незалежно від знаку в розподіл ймовірностей – ***softmax***:

$$\sigma(x)_j = \exp(x_j) / \sum_i \exp(x_i).$$

Функція крос-ентропійних втрат [2]. Використовуючи уявлення для функції softmax $\sigma(\vec{x})_j = \exp\{x_j\} / \sum_j \exp\{x_j\}$, для ненормованих чисел – логітів $L = \{l_j\}$, $l_j = b_j + \vec{x} \cdot \vec{w}_j$, можна отримати ймовірності $p(a) = \sigma(\vec{l}) = \exp\{l_j\} / \sum_i \exp\{l_i\}$. При цьому часто використовують уявлення – функцію крос-ентропійних втрат (cross-entropy loss) $X(\Phi, x) = -\ln p_a$. Тут Ф – позначення набору використаних параметрів. Ця функція завжди зменшується в міру покращення моделі, тому її

[23] Розроблено Ф. Розенблаттом в 1959 році.

[24] У теорії інформації крос-ентропія відповідає мірі відмінності двох розподілів.

використовують у ряді випадків як функцію втрат, іноді вибираючи лише один логіт. Для прикладу оцінимо градієнт. Процедура розрахунків така:

$$\frac{\partial X(\Phi)}{\partial l_j} = \frac{\partial p_a}{\partial l_j}\frac{\partial X(\Phi)}{\partial p_a}; \quad \frac{\partial X(\Phi)}{\partial p_a} = \frac{\partial(-\ln p_a)}{\partial p_a} = p_a^{-1};$$

$$\frac{\partial p_a}{\partial l_j} = \frac{\partial \sigma_a}{\partial l_j} = \begin{matrix}(1-p_j)p_a...(a=j)\\ -p_j p_a...(a\neq j)\end{matrix};$$

$$\frac{\partial X(\Phi)}{\partial l_j} = \frac{1}{p_a}[\begin{matrix}(1-p_j)p_a...(a=j)\\ -p_j p_a...(a\neq j)\end{matrix}]; \quad \Delta b_j = \eta\cdot[\begin{matrix}(1-p_j)...(a=j)\\ -p_j...(a\neq j)\end{matrix}];$$

$$\frac{\partial X(\Phi)}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}}(b_j + \sum_i x_i w_{i,j}) = x_i; \quad \Delta w_{i,j} = \eta x_i \frac{\partial X(\Phi)}{\partial l_j}.$$

Чим менший розмір партії, тим меншою має бути встановлена швидкість навчання η (Learning Rate).

Псевдокод для навчання набуває вигляду:

1. Для j та і встановити b_j і $w_{i,j}$ випадковими (але близькими до нуля).

2. Поки точність розробки не припинить зростати, слід

(a) для кожного навчального прикладу k у партіях з m прикладів

і. виконати прямий прохід, використовуючи рівняння

$$X(\Phi, x) = -\ln p_a;\ p(a) = \sigma(\vec{l}) = \exp\{l_j\}/\sum_i \exp\{l_i\};\ l_j = b_j + \sum_i x_i w_{i,j}.$$

іі. Виконати зворотний прохід, використовуючи рівняння

$$\Delta w_{i,j} = \eta x_i \frac{\partial X(\Phi)}{\partial l_j};\ \Delta b_j = \eta\cdot[\begin{matrix}(1-p_j)...(a=j)\\ -p_j...(a\neq j)\end{matrix}];\ \frac{\partial X(\Phi)}{\partial l_j} = \frac{\partial p_a}{\partial l_j}\frac{\partial X(\Phi)}{\partial p_a}.$$

ііі. у кожних m прикладів модифікувати всі Ф сумарними доповненнями.

(b) обчислити точність моделі, виконавши прямий прохід для всіх прикладів у корпусі розробки.

3. Вивести Ф з ітерації до зниження точності розробки.

Акредитація Кохонена. Сенс акредитації вченого в області обчислювальної техніки і нейронних мереж Теуво Калев Кохонена у виборі нейрона, який добре реагує на вхідний сигнал.

Потім для даного вхідного образу використовується метод навчання саме цього нейрона і деяких нейронів в його околиці. При цьому реакція цих нейронів посилюється, сигнал на них збільшується, а інші нейрони

шару при цьому мимоволі пригнічуються. Причому пригнічення їх відбувається 1) через малі початкові значення їх синапсів, 2) слабкого впливу на них і 3) переважної дії[25] так званих бічних – латеральних зв'язків між ними, зокрема в даному шарі.

Мережа Кохонена (Kohonen's Neural Nenwork = Kohonen's Self Organizing Feature Map). Ця одношарова мережа ділить вектори вхідних сигналів на підгрупи. Вхідні сигнали подаються на входи всіх нейронів.

АЛГОРИТМ:

1. Привласнення малих випадкових значень всіх ваг (ініціалізація).

2. Подаються на вхід послідовно початкові сигнали – вхідні образи.

3. Для кожного окремого сигналу x_i обчислюється відстань (часто говорять, від вхідного сигналу до цього нейрона) $d_j = \sum_{i=1}^{N}(x_i - w_{ij})$ кожного нейрона.

4. Для прискорення цього утворення кластерів знаходять нейрон, у якого ця відстань мінімальна.

5. Потім налаштовують тільки цей нейрон і його сусідів за правилами $w_{ij}(t) = w_{ij}(t-1) + \alpha \cdot [x_i - w_{ij}(t-1)]$ і рекомендують повернення до пункту 2.

Мережа Хопфілда[26] (Hopfield Network) являє собою один шар нейронів, число яких n відповідає числу входів і числу виходів. Всі виходи нейронів пов'язані з їх входами тут $w_{ij} = w_{ji}$ $w_{ii} = 0$ (що відповідає умовам теореми про стійкість такої мережі, доведеної Хопфілдом і Гросбергом).

Вагові коефіцієнти розраховуються і встановлюються до початку роботи мережі (наприклад, ініціалізація для автоасоціативної пам'яті $w_{ii} = 1$, $w_{ij} = \sum_{k=1}^{M} x_i^k x_j^k$, де x_i^k – i-й елемент κ-го вектора-зразка), потім вони можуть повільно змінюватися, підлаштовуючись до обраних початкових умов [4].

$$y_j(0) = x_j, \quad y_j(t+1) = \varphi[\sum_{i \neq j}^{N} w_{ij}(t) y_i(t)].$$

---

[25] У природному нейронному середовищі також спостерігається посилення збудження нейронів в малій локальній області, причому нейрони в околиці порушених нейронів-«переможців» останніми пригнічуються.

[26] Д. Хопфілд запропонував її в 1982 р., чим спровокував інтерес до нейронних мереж після нищівної критики М. Мінського і «першої зими».

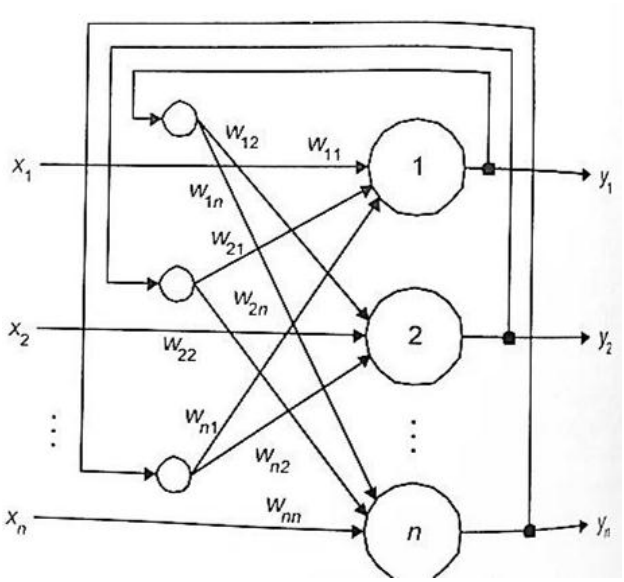


Рис. 6. Мережа Хопфілда (див. [4])

Мережа сама формує на виході т. н. кластери – еталонні рішення (зазвичай не більше 15n). Для аналізу використовують функцію $E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} y_i y_j - \sum_{j=1}^{n} x_j y_j + \sum_{j=1}^{n} \theta_j y_j$ – «енергію мережі» (використовують і інші подібні функціонали, наприклад, $E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} y_i y_j$), яка відношення до реальної енергії не має. Тут x_j, y_j – вхід і вихід *j*-го нейрона.

Після будь-якої ітерації зміна «енергії мережі» $\Delta E = -\sum_{j=1}^{n}[\sum_{j \neq i}(w_{ij} y_i) + x_j - \theta_j] \cdot \Delta y_j$. Мережа налаштована, якщо «енергія мережі» мінімальна і не змінюється. Якщо образи – еталони виявляються рознесені, вони всі різні і не корелюють, тоді новий сигнал відповідає одному з еталонів або не знаходить свого еталона. Якщо еталони близькі і кореляція помітна, можна говорити про їх ***асоціативний зв'язок***. Сенс цих еталонів визначається користувачем цієї мережі.

Мережа Хеммінга трохи складніше, складається з вхідного і вихідного шарів. Останні шари містять число нейронів, що дорівнює заданому користувачем числу еталонних рішень.

Вхідний шар практично не бере участь у формуванні рішень і тільки виконує передавальну функцію (тому його, взагалі кажучи, можна було б прибрати). Прихований (2) шар нейронів тільки підсумовує з вагою компоненти вхідного вектора x_i, тобто на вхід нейрона k прихованого шару

надходить x_i, відбувається підсумовування з вагою $s_k^{(2)} = \sum_{i=1}^{n} w_{ik} x_i + \theta_k$, і він, не перетворюючи це значення, передає його на вхід вихідного шару. Тобто сигнал на виході прихованого шару дорівнює $y_k^{(2)} \equiv s_k^{(2)}$. На входи нейронів третього (3) вихідного шару надходить $y_k^{(2)} \equiv s_k^{(2)}$. Активаційна функція створює результат на виході нейрона третього вихідного шару $y_k^{(3)} = \varphi[s_k^{(2)}]$. Цей сигнал разом з сигналами від усіх інших нейронів цього шару надходить на вхід даного нейрона $s_k^{(3)}(t+1) = y_k^{(3)}(t) - \varepsilon \cdot \sum_{\substack{j=1 \\ j \neq k}} y_j^{(3)}(t)$.

Перший доданок визначає ***позитивний зворотний зв'язок*** (вплив виходу нейрона на його вхід), другий – ***негативний зворотний зв'язок***. Тобто всі нейрони вихідного шару пов'язані негативними (інгібіторними) зв'язками. Але вхід кожного нейрона вихідного шару все ж пов'язаний з його виходом позитивним зворотним зв'язком. При стабілізації сигналів на виходах нейронів зовнішнього шару налаштування припиняється.

Якщо потрібно пов'язувати кожен вектор зовнішнього сигналу з отриманими в результаті налаштування еталонами, то корисно оцінити близькість двох послідовностей (наборів компонент векторів). Для цього У. Хеммінг запропонував користуватися мірою, яка почала називатися хеммінговою відстанню:

$$\rho(x, x') = bc\{(\overline{x}_i \wedge x_i') \vee (x_i \wedge \overline{x}_i')\},$$

де bc – визначає число приймаючих значення 1 (логічної одиниці) членів ряду. У разі близькості цих послідовностей можна говорити про те, що вхідний вектор відноситься до даного стандарту.

Допоміжні мережеві пристрої [4]. Мережа MAXNET[27] (MAXNET with Competition throw Lateral Inhibition). Здійснює в чистому вигляді пригнічення нейронами-переможцями своїх сусідів. Розмірність вхідного вектора N. Число синапсів $N(N-1)$, $w_{ii} = 1$, $w_{ij} = -\varepsilon \leq 1/N$.

Процедура: $y_j(0) = x_j$, $y_j(t+1) = \varphi[y_j(t) - \varepsilon \sum_{i \neq j}^{N} y_i(t)]$ – виділяє найбільший вхідний сигнал.

[27] Мережа-максимізатор, належить до групи мереж "winner takes all".

Нейрон Гросберга – ***вихідна зірка*** (outstar), самостійного інтересу не представляє, включена в загальну систему нейронної мережі. Вагові вхідні коефіцієнти налаштовуються за допомогою процедури $w_i(t+1) = w_i(t) + \eta \cdot [y_i - w_i(t)]$. Вихідний сигнал зазвичай статичний і визначає навчальний набір для інших нейронів.

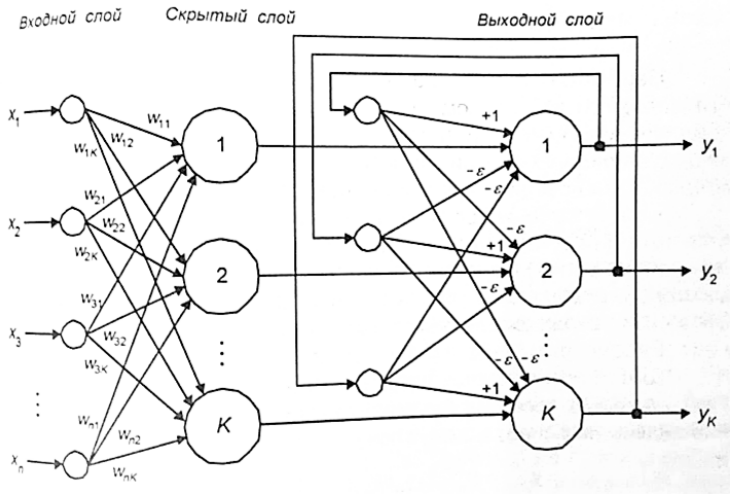


Рис. 7. Мережа У. Хеммінга (див. [4])

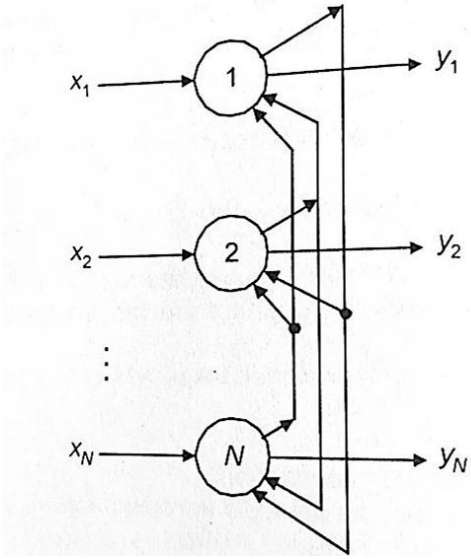


Рис. 8. Мережа MAXNET (див. [4])

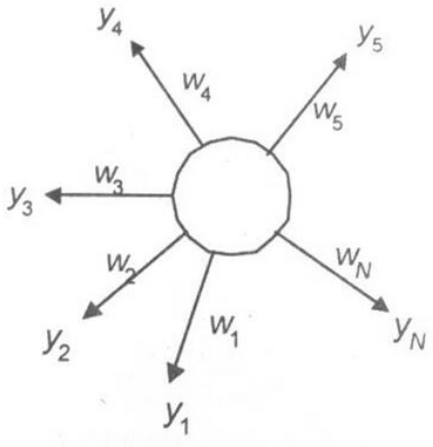


Рис. 9. Вихідний нейрон Гроссберга (див. [4])

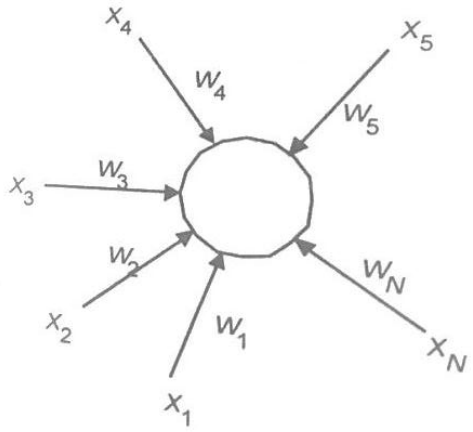


Рис. 10. Вхідний нейрон Гроссберга (див. [4])

Вхідна зірка (instar) – це інший нейрон Гроссберга. Самостійно інтересу не представляє, включений в загальну систему нейронної мережі. Налаштування $w_i(t+1) = w_i(t) + \eta \cdot [x_i - w_i(t)]$. Вхідна зірка налаштована на певний вхідний вектор, причому її для цього навчають. Вихід (не позначений на рисунку) – згортка вхідного вектора з ваговим вектором.

Література до розділу 4

1. Микелуччи У. Прикладное глубокое обучение. Подход к пониманию глубоких нейронных сетей на основе метода кейсов / Пер. с англ. – СПб.: БХВ-Петербург, 2020. – 368 с.: ил.

2. Черняк Е. Введение в глубокое обучение / Пер. с англ. – СПб.: Диалектика, 2020. –192 с. : ил.

3. Минский М., Пейперт С. Перцептроны. — М.: Мир, 1971 (https://ru.wikipedia.org/wiki/ Перцептроны).

4. Rosenblatt F. The perceptron — a perceiving and recognizing automaton // Report 85–460–1, Cornell Aeronautical Laboratory, 1957.

5. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. – 2 изд. – М.: Горячая линия - Телеком, 2002. – 382 с.

Розділ 5. ДЕЯКІ АНАЛОГІЇ В ОПИСІ БАГАТОПАРАМЕТРИЧНОЇ СИСТЕМИ

Людина створила в своєму розумі безліч образів і понять і хотіла би усвідомити, як ці образи взаємодіють один з одним[28]? Як пов'язані поняття? Саме ці завдання і повинні були вирішити в процесі створення систем штучного інтелекту.

Формалізоване знання. Друга сигнальна система дозволила перейти від усвідомлення понять до абстракцій, що породило формалізоване знання, тобто знання на основі теорій, методів розрахунків, що дозволяють прогнозувати поведінку спочатку простих об'єктів, а потім вже більш складних[29].

В принципі це знання дозволяє підготовленій і навченій людині вирішувати завдання з кінцевим числом варіантів. При цьому їй допомагають це робити обчислювальні пристрої, яким потрібно надати програму розрахунків – алгоритм і бази даних, відомості про процеси та їх учасників. Формалізовані знання дозволяють вирішувати так звані обчислювані завдання. Для цих завдань можна записати алгоритм, створити базу даних і вирішувати їх, використовуючи сучасні обчислювальні потужності. Таких завдань безліч, вони забезпечують високу точність, для них вистачає невеликого часу для їх постановки і рішення. Теорії і формалізовані процедури-методи дозволяють представити результат в такій же формальній формі.

Погано формалізовані дані. Деякі знання представлені наборами правил і процедур, які погано піддаються формалізації у вигляді теорій. В цьому випадку для пошуку рішень необхідно переглядати безліч даних, зіставляти їх і виділяти потрібні, що містить значну суб'єктивну складову. Природа для того і створила таку велику природну мережу – головний мозок, щоб накопичувати безліч даних і, порівнюючи їх, вибирати прийнятні рішення.

Для об'ємних, складних і масштабних завдань, які, наприклад, визначають поведінку людини в суспільстві, створення бази даних стало непідйомним завданням, яка вимагала б, зокрема, великих витрат людської праці з формалізації даних і формування великої бази цих даних.

---

[28] Кожне поняття описує деякий об'єкт, який має внутрішню структуру, але часто нам цікава поведінка цього об'єкта як цілого, а внутрішні особливості в розрахунок часто не приймаються.

[29] Теорії дозволяють від загальних принципів, за допомогою розроблених процедур, знайти приватні рішення, загальний обсяг яких надзвичайно великий і у багато разів перевершує обсяги баз даних і знань даної теорії.

Обсяги даних, які збирає людина за все життя, формуючи свою пам'ять, аналог бази даних машини, настільки великі і настільки різноманітні, що формалізувати їх навряд чи вдасться. Такі необчислювані завдання, які не піддаються формальним методам, вирішують інакше – обробляється і сприймається величезне число даних.

В результаті виходить вже усвідомлення матеріалу за рахунок оптимізації реакцій і рефлексій, порівняння отриманих рішень з існуючими подібними рішеннями в більш ніж великій базі даних, на рівні більше–менше – саме так формується людське розуміння[30]. Для вирішення подібних завдань виявилися корисні штучні нейронні мережі. Побічним результатом розвитку інформаційних технологій стали деякі несуворі поняття і підходи, які внесли плутанину в загальну структуру представлення знань і визначення процедур отримання рішень.

Методи логічного висновку в сучасному поданні вже не жорстко пов'язані з класичною математичною логікою (що оперує поняттями істина і неправда), а використовують також строго обґрунтовану математично нечітку логіку (з її функціями належності). Причому це зовсім не евристичний якісний опис, а скоріше інша мова опису та процедур, які цілком обґрунтовані математиками і є в своїй галузі застосування строгими[31]. Неточність і наближеність тут залишилася в уявленні предметів і явищ на основі опису набором функцій приналежності, тобто наявність певних властивостей задано значенням відповідної функції приналежності.

Важливо, що кожен об'єкт може належати до різних множин в різному ступені, тобто у одного об'єкта може бути досить великий набір функцій приналежності. Хоча слід зазначити незвичність опису процесів у рамках нечіткої логіки, до якої користувачі поки не звикли, тому дуже докладне освоєння цього методу опису потребує часу.

Імітаційні моделі, що засновані на суб'єктивному сприйнятті реальності їх творців, потрібно відрізняти від систем рівнянь, коректною мірою на основі суворої математики, отриманих зі створених і апробованих людством теорій. В електродинаміці – це рівняння Максвелла, в гідродинаміці – рівняння Нав'є–Стокса. Це рівняння спеціальної теорії відносності

---

[30] Це, наприклад, реалізовано в програмі IBM Debater, де реакції програми підганяли під реакції людини. Конкурс з такою програмою поки виграла людина (Хариш Натарайан), але, як вважали експерти, більше за рахунок недоступної машині емоційної подачі матеріалу, а не наукової суворості.

[31] Відносити їх до м'яких обчислень, розуміється як об'єднуючі неточні, наближені методи розв'язування задач, які практично неможливо вирішити кінцевим числом процедур (тобто кінцевий час, кінцевим числом кроків) не зовсім правильно, оскільки рішення нечіткої логіки здійснюються кінцевим числом формальних операцій і є точними в прийнятих термінах.

А. Ейнштейна, рівняння Е. Шредінгера і Дірака і безліч інших. Ці рівняння, або мовою програмістів – моделі, багаторазово перевірені в експериментах, спостереженнях і в області своєї застосовності є правильними. В цьому випадку вживати термін «імітаційні» для подібних апробованих практикою моделей не некоректно.

Імітаційні моделі створюються на основі досвіду, використання аналогій, побачених в коректних описах, про які йшлося вище. Автори імітаційних моделей вибирають значення параметрів, відповідні на їх погляд диференціальні й інтегральні (і навіть варіаційні) оператори, що описують, за їхніми суб'єктивними уявленнями, потрібний характер впливу на змінні величини. Важливо відзначити, що області застосування, тобто області зміни параметрів і фізичних (та інших) величин імітаційних моделей, вкрай вузькі, помітна зміна величини окремих параметрів призводить до неадекватності моделі і модельованого явища. Застосування таких імітаційних моделей жорстко вимагає додаткової інформації для користувача про інтервали допустимої зміни параметрів і фізичних (та інших) величин, що апріорі обґрунтувати досить складно[32].

До речі, цього немає в моделях, виведених з перших принципів, отриманих з багаторазово повторюваних експериментів і спостережень, тобто апробованих багаторічною практикою. Тому до імітаційних моделей слід ставитися з великою обережністю. Їх використання раціонально при відсутності можливостей знайти підходящу апробовану теорію, обумовлено необхідністю створення простого і зрозумілого для інженерів та виконавців опису[33].

Еволюційне моделювання – це програмний опис процедур взаємодії агентів у заданих умовах, що цілком піддається строгому математичному аналізу.

Якщо взаємодія із середовищем окремого агента спирається на апробовані і перевірені практикою теорії, це й опис некоректно називати евристичним. Цей опис цілком строгий. І від Дарвіна тут залишився тільки підхід–опис, в якому можна виявити домінування найбільш пристосованого агента до заданих умов і його оточення. Інший раз творці еволюційних моделей відмовляються від традиційних апробованих і багаторазово підтверджених практикою підходів і описів поведінки окремого агента і його впливу на сусідів і середовище в цілому. При цьому вони шукають на

[32] Взагалі кажучи, ці інтервали допустимих величин повинні визначатися в натурних експериментах, оскільки аналітичні методи визначення допустимих відхилень параметрів і величин в цьому випадку навряд чи застосовні. Це одна з причин, чому творці імітаційних моделей ухиляються від обґрунтування меж застосовності їх творінь.

свій страх і ризик не традиційний шлях уявлення взаємодії агентів ансамблю або рою, але тоді визначення їх моделей як «евристичні» і «імітаційні» цілком підходять.

Зауважимо, що сучасні вимоги до опису безлічі агентів і середовища включають також більш загальний принцип самоузгодження. Дотримуючись цього принципу, потрібно враховувати не тільки як середовище впливає на поведінку агентів, а також яким чином агенти впливають на середовище.

Недетерміновані функції (програми) в програмуванні визначаються як функції, які на один і той же вхідний сигнал дають різні результати. Природу цього бачать в обліку мінливих обставин і реакції програми на ці зміни, або це може бути наслідок існування в тілі програми генераторів випадкових чисел. Але не можна виключати і виникнення стохастичності в рішеннях, тобто існування в математичній моделі «дивних» атракторів (що, взагалі кажучи, може виявитися несподіванкою). Недетермінований алгоритм використовує методи проб і помилок, повторів або випадкового вибору [1]. Локальна детермінованість – кожен крок обчислень є обумовленим і певним. Глобальна недетермінованість[34] – відсутній результат повного обчислення.

Системи інтегродиференціальних рівнянь і нейронна мережа. Система інтегродиференціальних рівнянь (в часткових похідних) може бути представлена як набір пов'язаних між собою операторів. Параметри цієї системи рівнянь (коефіцієнти перед операторами) визначають відносну вагу цих операторів. Існує деяка аналогія з нейронною мережею, де нейрони виконують функцію операторів, а синапси забезпечують зв'язок між ними. Причому ваги синапсів фактично визначають значення параметрів мережевої системи.

Роль зворотного зв'язку. В електроніці зворотні зв'язки, як відомо, призводять до таких наслідків. Негативний зворотний зв'язок, коли вихідний сигнал подається на вхід пристрою зі зворотним знаком, використовується для ослаблення шумів і забезпечення чистоти сигналу, що підсилюється. Позитивний зворотний зв'язок, коли вихід і вхід пов'язані, причому з виходу на вхід пристрою сигнал подається без зміни, нехай навіть ослаблений, здатний забезпечити самозбудження (використовується для створення генераторів).

---

[34] Подібна невизначеність виникає в так званих ad hoc мережах, «створених на льоту», які виникають при випадковому або навіть довільному об'єднанні (типу «клунж») окремих обчислювальних структур.

Застосування нейронних систем для вирішення інтегро-диференційних рівнянь безперервних середовищ і / або їх аналогів в дискретному описі почалися порівняно недавно. Цьому заважало уявлення, що штучні нейронні системи більше пристосовані для задач розпізнавання, якісної оцінки, але не точних кількісних рішень. Хоча всі розуміли, що людина, природний розум якої побудований за таким же принципом, як і нейронна мережа, безсумнівно здатна до вирішення завдань з високою точністю. Отже, не існує принципової неможливості навчити штучну нейронну мережу рішенням систем рівнянь, які застосовуються в багатьох теоретичних описах. Звичайне рішення рівнянь – багатокрокове застосування з початкових або граничних умов в рамках покрокової сітки методу Рунге–Кута або йому подібних. Звичайно, потрібно було визначати стан системи послідовно на кожному кроці, і для пізніх кроків слід знати всі попередні. В роботі M. Dissanayake і N. Phan–Thien представили свій сітковий алгоритм [2] застосування нейронної мережі для вирішення диференціальних рівнянь. Надалі нейронні мережі, які вирішували диференціальні рівняння, почали називати DENN (Differential equations neural networks).

Підходи до випадкової вибірки доменів. Замість сітки Дж. Сіріньяно і Л. Спіліопулос [3] запропонували випадковим чином вибирати домени з цієї сітки, що знижує обчислювальну складність (для сіткових підходів складність зростає експоненціальним чином). Виявилося, що ці методи продемонстрували свою перевагу перед традиційними в ряді випадків. З іншого боку, ці підходи дозволяють навчати нейронні мережі новими методами.

Можливості верифікації. Оскільки традиційно нейронні мережі використовують для вирішення завдань, які погано визначені, які не інтегруються відомими стандартними методами і часто взагалі не інтегровані (це обробка тексту і мови, машинний зір і т. п.), то там доводиться йти навпомацки. Добре, що в задачах з інтегрування диференціальних і інтегродиференціальних рівнянь багато рішень відомі, причому навіть аналітичні, що спрощує верифікацію, дозволяє домогтися кращої методики навчання мереж.

Навчання на основі кінцевого набору рішень. Пізніше І. Лагаріс з колегами запропонував спочатку навчити мережу працювати з даною системою диференціальних рівнянь, змінюючи початкові і граничні умови. А потім, вже не вирішуючи цю систему, «приблизне значення рішення в будь-якій точці в межах діапазону навчання можна обчислити за постійний час без необхідності обчислювати попередні» [4]. Тобто поки робота у формуванні підходів для вирішення нейронною мережею систем інтегродиференційних рівнянь триває.

Системи інтегродиференціальних рівнянь, які описують збурення середовища, з наявністю параметрів (зв'язків між операторами) неявно мають зворотні зв'язки, як негативні, так і позитивні. Оскільки всі оператори

зв’язані між собою, сигнали на межах середовища породжують вимушене збудження, а потім формують вимушене рішення мережі. Ці процеси відносяться до граничних задач. Начальні стани в об’єму середовища породжують рішення начальних задач, до речі, задач Коші.

Однак обернені зв’язки призводять до появи власних, автомодельних рішень інтегродиференційних рівнянь. Виниклі в глибині середовища, яке описано системою рівнянь, ці рішення, притаманні системі, можуть слабо, а то і зовсім не залежати від початкових і пограничних умов. Їх математики назвали автомодельними рішеннями, автохвилями. Часто подібні власні рішення виникають поблизу областей простору змінних, званих атракторами.

Мешканці нелінійного середовища. Що нам відомо про системи нелінійних інтегродифереціальних рівнянь? Це добре розвинена область математики і математичної фізики, і відомо про неї досить багато. Перш за все, ці системи рівнянь описують поширення (існування) якихось збурень в деякому середовищі. Саме параметри цієї системи і її конструкція (вид операторів і їх розташування і зв'язки) визначають це середовище. Рішення цієї системи можуть залежати від початкових і граничних умов. Але якщо кордони далеко і відлік часу почався давно, можливе існування стаціонарних рішень – їх називають автомодельними рішеннями, автохвилями, хоча це не обов'язково хвилі.

Для цих рішень – автомодельних рішень, автохвиль середовище, що представлено системою рівнянь, – це місце їх існування, вони там можуть існувати без особливих змін скільки завгодно довго, якщо немає механізмів втрати їх енергії (консервативні середовища), тобто немає механізмів дисипації. Таких автомодельних рішень, автохвиль може бути багато, і їх число залежить від складності системи (її розмірності – числа змінних) і від кількості параметрів[35]. Для простих систем знайдені рішення для набору автомодельних рішень, автохвиль, їх явний символьний вид, але якщо система дуже складна, багатопараметрична, знайти явний вигляд всіх автомодельних рішень, автохвиль цієї системи стає непідйомним завданням. Математики відмовилися від цього наміру вже давно.

Крім автохвиль, область локалізації яких може бути довільною, існує більш численна спільність реакцій системи, які можна назвати атракторами. В околиці атракторів – притягуючих центрів, власні рішення систем рівнянь

[35] Математики знайшли спосіб визначення виду цих автохвиль, використовуючи формалізм теорії груп. Представляючи деяке обурення як рішення, вводимо мале (інфінітезимальне) відхилення всіх величин і вимагаємо, щоб отриманий вираз задовольняв системі, тобто перетворення повинно зберегти інваріантним систему.

збираються, накопичуються. У нейронній мережі роль атракторів можуть виконувати області мережі, де зосереджуються власні рішення певного виду. Власне, спочатку творці перших нейронних мереж спеціально створювали (використовуючи навчання з учителем) такі області-атрактори для кожного класу рішень або знаходили (при навчанні мережі без вчителя) їх і використовували в подальшій роботі. Творці перших нейромереж з асоціативними зв'язками створювали такі області-шари-атрактори для кожного класу рішень та використовували їх у своїй роботі.

Взагалі кажучи, власні рішення (зокрема автохвилі в середовищах, що описуються системами інтегродиференційних рівнянь) можуть взаємодіяти між собою і обмінюватися енергією, тобто змінювати свою амплітуду – величину, особливо не змінюючи свій вигляд – свою індивідуальність[36]. Якщо внести в систему збурення–шум, обмін енергією між власними рішеннями середовища посилюється. Цей шум може бути непереборним власним шумом системи – флуктуаціями, які пов'язані з існуванням і взаємодією безлічі власних рішень. А може бути внесений характером – процедурою рішень, наприклад, дискретним машинним розрахунком, який виконується з деякою точністю.

Зовнішні впливи на середовище. Тепер подивимося, як впливають на систему та її мешканців–власних рішень зовнішні впливи (зокрема граничні і початкові умови). Кожне зовнішнє обурення провокує зростання (або придушення) власних рішень. Одні рішення-автоволни ростуть, інші можуть пригнічуватися. Крім цього, оскільки складно зовнішнім впливом вибірково збільшувати одне з власних рішень, то навіть при певному резонансі з ним будуть порушуватися, можливо дещо слабше, інші власні рішення і напевно почне посилюватися деякий побічний шум. Наприклад, експериментально виявили, що зовнішній імпульс здатний викликати синхронну активацію багатьох нейронів і сформувати просторово-часові патерни (synfire chains) [5-7].

Режими без урахування автомодельних рішень. У багатьох випадках, наприклад, в електроніці в режимах посилення, рішення, отримані на виході, є вимушеними в тому сенсі, що система просто реагує на сигнал на вході (гранична задача). Саме такий режим роботи нейронних систем реалізується в мережах прямого поширення. Тоді існування власних автомодельних збуджень часто не враховується.

Зворотні зв'язки в нейронних мережах. Якщо нейронна мережа обрана так званого прямого поширення, наступні шари мережі

[36] Оскільки вид автохвиль досить консервативний, можна вважати їх квазічастинками і розглядати газ таких квазічастинок, їх взаємодію в цьому середовищі і характер розподілу по енергії.

(відлічувані від вхідного, куди подається сигнал) можуть не впливати на попередні. В цьому випадку зворотних зв'язків немає і мережа підсилює і перетворює вхідний сигнал. Це подібно режимам підсилення в електроніці.

Якщо ж внести зворотні зв'язки в мережу, в ній здатні виникати власні рішення, взагалі кажучи, які слабко залежать від вхідного сигналу. Але вхідний сигнал може по-різному провокувати виникнення і посилення цих властивих мережі її власних рішень. Це явище лежить в основі навчання мережі без вчителя.

Ці рішення, що слабко або взагалі не залежать від вхідних сигналів та блукають у товщі мережі, аналогічні автохвилям в середовищах, які представлені системами інтегродиференціальних рівнянь. Поки що ці явища дослідників не цікавлять.

Навчання системи і користувачів. Якщо є механізм зміни параметрів системи, що дозволяє збільшити резонанс, тобто домогтися того, щоб зовнішній вплив в основному збуджував одне власне рішення (автовхвилю), то можна говорити про цілеспрямовані зміни системи рівнянь (структури і виду власних рішень як в системі рівнянь, так і в наборі довготривалих збуджень в нейронній мережі). Тобто можна говорити про зміну нелінійного середовища. Це подібно до навчання нейронної мережі.

Але можна обійтися і без навчання, вважаючи, що кожний зовнішній вплив певного виду потрапить в резонанс з одним з автомодельних рішень системи. Тоді цей вибір системи стане нашим вибором, і ми будемо орієнтуватися на цю суперпозіцію рішень. Як реакцію на даний зовнішній вплив. Це подібно до навчання без вчителя в теорії нейронних мереж. Власне, це самі вчителі навчаються ототожнювати реакції системи з вхідними сигналами (зовнішнім впливом).

Виявилося також, що Клітинні Нейронні Мережі (Cellular Neural Networks), які використовували правило Хебба і були реалізовані у вигляді інтегральних мікросхем, могли за рахунок паралельного перетворення вхідних сигналів допомогти вирішувати завдання розпізнавання образів, але, головне, формували в своєму середовищі різні автохвилеві структури для генерації ритмів. Експериментально виявлені були повторювані послідовності імпульсів (спайки), які активні нейрони формують, і просторові структури, що вважають результатом кодування інформації і формуванням пам'яті.

Значні роботи з Large-scale моделювання мозку активно проводилися Е. Іжікевічем і нобелівським лауреатом Дж. Едельманом. Основою була феноменологічна модель нейрона, запропонована Е.

Іжікевічем. У процесі моделювання було використано 22 типи нейронних клітин, які отримують шляхом зміни параметрів моделі Іжікевіча [5-7]. Серед модельних нейронів було виділено 8 типів нейронів, що збуджуються, і півмільярда синапсів з короткочасною синоптичною пластичністю. Цікавим також є проект Blue Brain, розпочатий в 2005 році (École Polytechnique Fédérale de Lausanne – EPFL and IBM). Метою проекту було моделювання окремих нейронів і утворених ними типових колонок неокортексу мозку – неокортикальних колонок [5].

Про нові підходи до опису штучного нейронного середовища. Тепер розглянемо системи, прямо не ототожнюючи їх з набором нелінійних рівнянь або з нейронною мережею. Тобто абстрагуємося від конкретного її виду. Можна вважати автомодельне збурення, автохвилю або довгоживучі збурення в мережі деяким віртуальним чином, який система і іноді її користувач (при навчанні без учителя) можуть ототожнювати з певними поняттями. Зрозуміло, що кожен образ-поняття (автохвиль в системі рівнянь або образ-збурення в нейронній мережі) здатний взаємодіяти з іншими образами-поняттями. Ці взаємодії – властивості вже самої системи. Величезне число зв'язків і параметрів (численних синапсів з різною вагою в нейронній мережі) здатні породити дуже значне число власних рішень (автохвиль в системі великого числа рівнянь і довгоживучих збуджень в нейронній мережі). Зовнішні впливи (сигнали) призводять до появи (або стимулюють їх появу) цих довгоживучих об'єктів – понять, які пов'язані з один з одним. Виявити їх існування і зв'язкі між ними напевно можна, організовуючи інтерфейс, що дозволяє їх побачити і усвідомити користувачеві. Потрібно розуміти, що виникнення цих образів-понять в глибині системи далеко не завжди буде представлено у формальному, явному вигляді користувачеві, але це може виявитися необов'язковим. Головне, що ці внутрішні образи здатні прояснити користувачеві картину їх взаємодії, що в багатьох практичних випадках і було б вирішенням завдання. Важливо, що вхідні сигнали (уявлення, вектори, що визначають образи, які користувач ототожнює з певними поняттями) виявляються пов'язані з образами, що формуються як вимушено, так і самостійно в глибині системи. І ця відповідність дозволяє стежити за грою цих образів, переводячи ці події на мову звичних користувачеві понять.

1. Кожне поняття з великого набору і відомі взаємодії (зв'язки) цих понять, усвідомлені користувачем, представляються у вигляді початкових векторів, зручних для введення в систему. Для цього використовуються класифікатори, мови, різні форми представлення даних.
2. Система, реагуючи на початкові або граничні умови (вектори), створює, крім вимушених рішень, самостійно власні рішення (автомодельні збудження в системі рівнянь і довгоживучі збудження в нейронній мережі). Зрозуміло що задані користувачем початкові вхідні вектори, що формують вимушені рішення, в більшій чи меншій мірі можуть бути пов'язані зі

сформованими перш в системі образами і їх взаємодією. За допомогою різних видів оптимізації формуються зв'язки між заданими користувачем вхідними даними і образами, створеними системою.

3. Крім того система сама здатна формувати зв'язки між цими образами, підказаними користувачем і, крім того, виявленими системою самостійно.

4. Якщо організований інтерфейс, що дозволяє продемонструвати ці рішення користувачу, то останній отримує відповідь.

5. При необхідності система з дуже великим набором різних або однакових операторів (нейронів), великим числом зв'язків між ними (синапсів) і різних їх ваг (значень параметрів) може підключатися до безперервного джерела інформації, створюючи величезне число пов'язаних образів у своєму середовищі, спостереження за якими (за допомогою інтерфейсу) може дозволити виявити багато рис процесів і особливості, які раніше вислизали від уваги користувача.

Але дослідникам цікаво розібратися і в роботі людського інтелекту, для цього також створюються нейромережі з біологічно реалістичних елементів і зв’язків.

Література до розділу 5

1. Зиглер К. Методы проектирования программных систем. М.: Мир, 1985.

2. M. W. M. G. Dissanayake and N. Phan–Thien. Neural–network–based approximations for solving partial differential equations. Communications in Numerical Methods in Engineering,10(3):195–201, March 1994.

3. Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. arXiv preprint arXiv:1708.07469, 2017.

4. I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks, 9(5):987–1000, September 1998.

5. Прокин И. С., Симонов А. Ю., Казанцев В. Б. Математическое моделирование нейродинамических систем: Электронное учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2012. – 41 с.

6. Izhikevich E.M., Gally J.A., Edelman G.M. Spike-timing dynamics of neuronal groups. // Cerebral cortex (New York, N.Y.: 1991). 2004. Vol. 14, № 8. P. 933–944.

7. Diesmann M., Gewaltig M.O., Aertsen A. Stable propagation of synchronous spiking in cortical neural networks. // Nature. 1999. Vol. 402, № 6761. P. 529–533.

Розділ 6. ОРГАНІЗАЦІЯ АСОЦІАТИВНОЇ ПАМ'ЯТІ

Збуджений нейронний слід в корі головного мозку володіє меншим опором і по ньому зазвичай іде і новий процес. Він же формує і підтримує основну пам'ять (тут пам'ять – це конфігурація вагової матриці) У пам'яті будь-якого виду є образи – об'єкти і відносини між ними (відносини обумовлені участю одних і тих же нейронів у реалізації різних образів).

Асоціації. Оскільки нейрони можуть брати участь в різних нейронних слідах, з'являється зв'язок між різними сегментами пам'яті, між сформованими образами. Якщо цей зв'язок сильний, система формує пропозиції (предикати), які також можна вважати складними образами, хоча це вже відносини декількох образів. Не обов'язково цей зв'язок буде очевидним для користувача[37]. Важливо зауважити, що ці комплексні образи, що включають відносини між об'єктами, будуть відразу збуджуватися в навченій або налаштованій мережі, тобто не вимагають спеціальних методів збудження, на відміну від асоціативних слабких зв'язків.

Слабкі асоціації. У разі якщо зв'язок слабкий і вимагає спеціальних методів для його виявлення, то можна говорити про асоціативний зв'язок. Зазвичай асоціативний зв'язок неявний і часто прихований для користувача. Зв'язок цей може бути між об'єктами і відносинами одного типу або зовсім різного типу. Це – асоціативна пам'ять[38].

Асоціативний спосіб доступу – взаємодія образу на вході з образами після успішної реєстрації мережею. Автоасоціативний спосіб орієнтується на кореляцію частин об'єктів, які розшукуються і вже представлені. Гетероасоціації таких очевидних кореляцій не мають. Можна ввести в обговорення прямі і непрямі асоціації. Перші мають прямий зв'язок між об'єктами, другі пов'язані через інші об'єкти (причому для користувача цей зв'язок може бути не очевидним).

Пошук за фрагментом і згадки. Асоціативна пам'ять потрібна для пошуку за окремим фрагментом або за згадуванням. Складається з двох каналів введення – по першому подають ключ (ознака), по другому контент, який треба запам'ятати. Для ініціювання запису потрібен тільки ключ.

---

[37] В цьому проявляється сильна і слабка сторони нейронних мереж. Сильна в тому, що, формуючи образи і їхні стосунки, користувач виявляє створення мережею і нових відносин, що для нього буде приємним (сподіваємося) сюрпризом. Слабка в тому сенсі, що користувач заздалегідь не може припустити, які додаткові зв'язки і відносини виникнуть в мережі.

[38] Зв'язок між психічними утвореннями або образами – асоціація. (Дж. Локк, 1698 р.)

Приклад з іншої області. Нехай є середовище, де для збурень виконується кілька рівнянь виду

$$\frac{\partial^2 F_i}{\partial t^2}+\frac{\partial^2 F_i}{\partial x^2}+\frac{\partial^2 F_1}{\partial y^2}+\frac{\partial^2 F_i}{\partial z^2}+\Omega_i{}^2(F_i)F_i=0\,. \qquad (3)$$

Іншими словами, саме виконання цих рівнянь для збурень і визначає це середовище. Обурення, описувані цими рівняннями, є власними рішеннями рівнянь (3) цього середовища і, взагалі кажучи, якщо немає втрат, можуть там існувати як завгодно довго. Розглянемо одне з рішень

$$F_1=A_1^{(1)}exp(i\vec{k}\cdot\vec{r})+A_2^{(1)}exp(i2\vec{k}\cdot\vec{r})+...+A_n^{(1)}exp(in\vec{k}\cdot\vec{r})+... \qquad (4)$$

Припустимо, ми якось зменшимо значення $A^{(1)}{}_n$, тоді, щоб не міняти загальної форми збурення – автохвилі, всі інші члени $A^{(1)}{}_{j\neq n}$ в (4) також повинні пропорційним чином зменшитися, оскільки форма рішення ніяк не може змінитися, середовище цього дозволяти не повинно.

При збільшенні $A^{(1)}{}_n$ всі $A^{(1)}{}_{j\neq n}$ в (4) також збільшуються. Зрозуміло, що і в першому, і другому випадку ми повинні відібрати або додати в систему енергію, щоб ці процеси мали місце.

Якщо система крім рівняння (3) при $i=1$ допускає виконання рівняння, подібного рівнянню виду (3), при $i=2$ у формі рішення

$$F_2=A^{(2)}{}_1exp(i\vec{k}\cdot\vec{r})+A_2{}^{(2)}exp(i2\vec{k}\cdot\vec{r})+...+A^{(2)}{}_nexp(in\vec{k}\cdot\vec{r})+..., \qquad (5)$$

то тут можлива взаємодія рішень (4) і (5) при виконанні умови

$$A^{(1)}{}_n=A^{(2)}{}_n. \qquad (6)$$

Тоді, порушивши в середовищі обурення у вигляді одного рішення (4), ми неодмінно створимо обурення, яке описується рішенням (5). Через зв'язок (6), звичайно. Зрозуміло, що доведеться витратити більше енергії, оскільки створюючи (4), ми, самі того, можливо, не помічаючи, створюємо і (5).

Повернемося до нейронної мережі. Нейронна мережа – це те ж середовище.

Оператори – тут це окремі нейрони. Рішення – збудження у тривимірному графі (вузли – нейрони, зв'язки – синапси). Саме матриця ваг-синапсів разом з активаційною функцією нейронів і створює середовище для існування рішень.

Якщо мережу налаштувати (або вона сама спочатку була налаштована) так, щоб нейрони на виході генерували рішення, подібне (4), і одночасно рішення, подібне (5) при початкових умовах, обраних вагах і даній активаційній функції нейронів, то умова (4) тут еквівалентна вимозі, що один і той же нейрон бере участь у двох рішеннях відразу. Домагаючись рішення (4), ми викликаємо до життя рішення, подібне (5), що і пояснює механізм асоціативного зв'язку.

З виходу знімається інформація і одночасно організовується зворотний зв'язок (зворотний зв'язок потрібен для багаторазового запам'ятовування і для посилення – провокування – дії ключа) процесу зчитування. Спільно діючі сигнали зворотного зв'язку і ключа підсилюють ефективність вилучення інформації, яку запам’ятовано. Подивимося, як реалізуються ці ідеї в нейронних мережах.

Відновлення інформації, яку запам’ятовано. Двонаправлена асоціативна пам'ять – мережа Коско (B. Kosko) – реалізує в основному гетероасоціативну пам'ять. Мережа складається з вхідного, прихованого і вихідного шарів. Тут F – активаційна функція нейронів, X і Y – вхідний і вихідний вектори.

Вихід нейрона вихідного шару безпосередньо пов'язаний зі входом відповідного нейрона вхідного шару. Спочатку мережа навчається. При цьому налаштовуються ваги прихованого шару W^T, а матриця ваг-синапсів вихідного шару в результаті навчання – настроювання представляється як [1].

$$W = \sum_{i=1}^{N} X_i^T \cdot Y_i \tag{7}$$

Нагадаємо, що стан ваг формує довгострокову пам'ять. Для відновлення запам’ятованої інформації вектор X або його частину короткочасно встановлюють на виходах прихованого шару. Після відключення X результат на виході вихідного шару $Y = F(X \cdot W)$ надходить на вхід вхідного шару $X = F(Y \cdot W^T)$, поки не досягається стійкий стан для векторів $X; Y$. В результаті отримуємо після успішної реєстрації вхідний вектор X і рішення Y, які запам’ятовано.

Модель адаптивно-резонансної теорії (Карпентер–Гросберг) – початкові значення ваг-синапсів шару розпізнавання випадкові і невеликі.

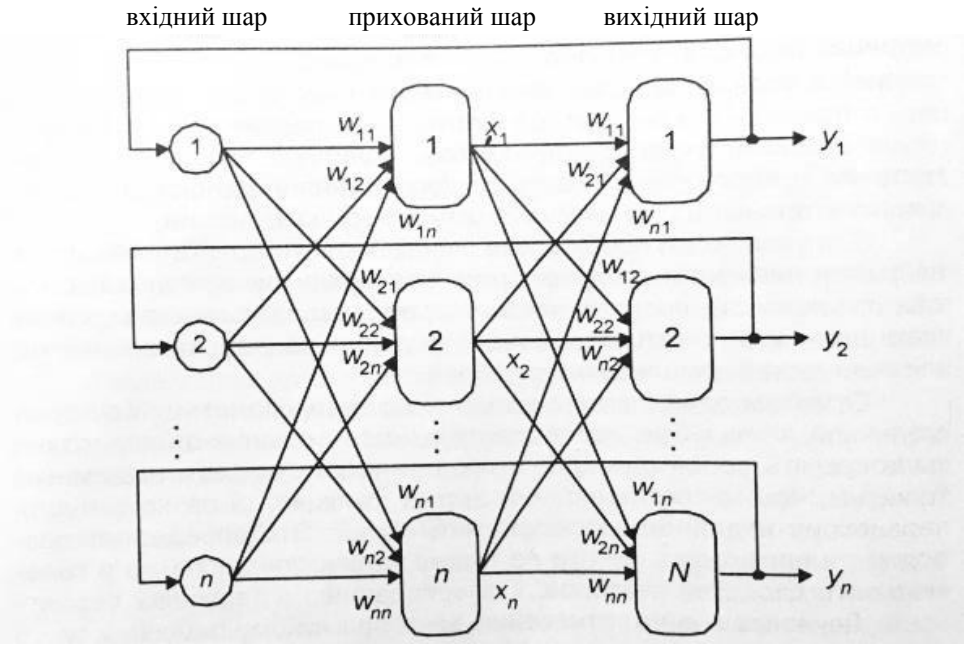


Рис. 11. Мережа Коско (див. [1])

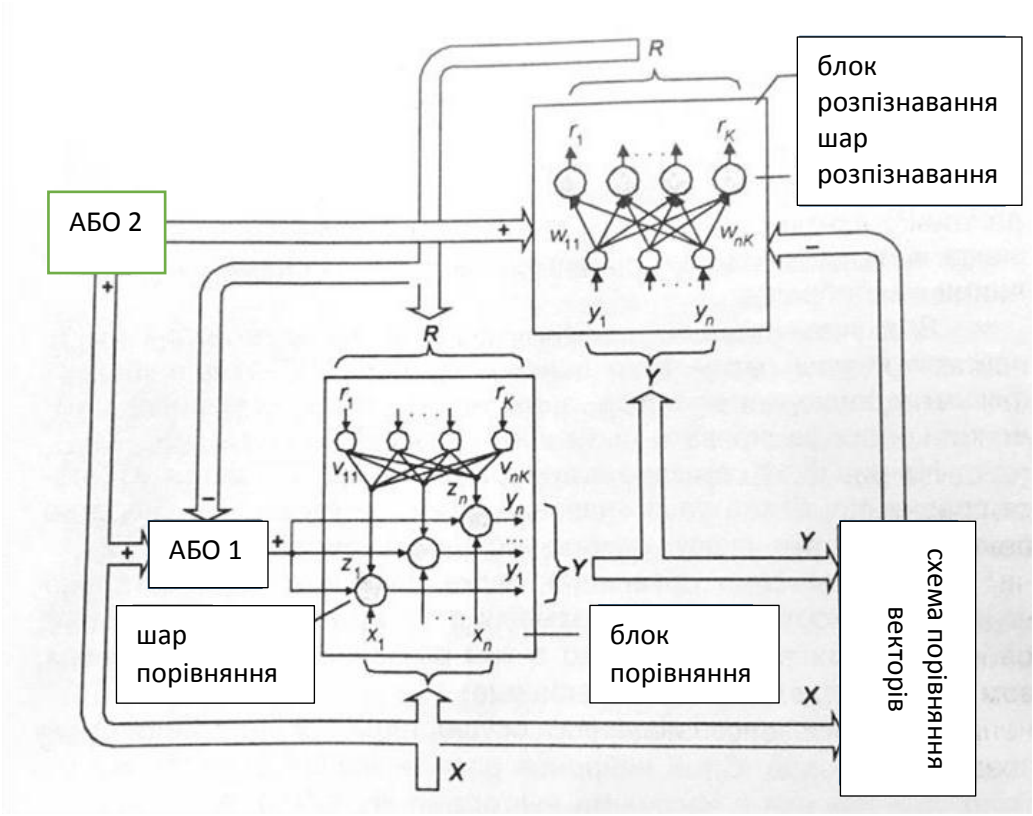


Рис. 12. Мережа АРТ (див. [1])

Вводять значення параметра подібності (0–1), вагові вектори усі рівні. Нейрони шару порівняння (блок порівняння) діють в режимі мажоритарного порівняння (в ідеалі вихід нейрона дорівнює одиниці тільки при одиницях на двох з його трьох входів). У блоці розпізнавання вхідні вектори класифікуються. Матриця ваг нейронів шару розпізнавання

$W_k = \{w_{ik}\}$ і всі нейрони взаємодіють латерально-гальмуючою схемою, коли в кожен момент збуджується тільки один нейрон-переможець. Число нейронів K шару розпізнавання дорівнює числу образів, що запам'ятовуються. ***Схема визначення подібності векторів порівнює*** X ***і*** Y. Якщо подібність менше встановленого порогу, збудження нейрона шару розпізнавання знімається.

Зв'язки-синапси на початку малі, нейрони не активовані порівняно з рівнем нормального збудження, що очікується після навчання. Значення всіх вагових векторів $V_k = \{v_{ik}\}$ дорівнює одиниці. Параметр відповідності $\rho \in (0,1)$ залежить від того, який ступінь відповідності влаштовує користувача.

Навчання без вчителя. При ***повільному навчанні*** вхідні вектори пред'являються короткочасно і послідовно, система реагує на всю спільність початкових умов: вагові коефіцієнти матриці $V_k = \{v_{ik}\}$ в масиві нейронів беруть безперервні значення. Розподіл значень матриці в цьому випадку відповідає середньому виду всіх вхідних сигналів. Це реакція на увесь масив векторів входу.

При ***швидкому навчанні*** довго не змінний вхідний вектор дає можливість системі сформувати стійкі значення (бінарні, тобто 1 і 0) матриці вагових коефіцієнтів $V_k = \{v_{ik}\}$ (див. блок порівняння) ***за рахунок літеральних негативних зворотних зв'язків і позитивного зв'язку між входом і виходом кожного нейрона*** (при цьому формується спільність «нейронів-переможців»). Це адекватно до кожного з сигналів входу налаштування.

Після послідовної подачі всіх вхідних векторів у шарі порівняння також послідовно формується вектор Y, який подається у блок розпізнавання та ініціює відповідний нейрон блоку. Надалі при навчанні змінюється вагова матриця $W_k = \{w_{ik}\}$ за такими правилами [1]:

$$w_{ik} = \frac{H \cdot y_i}{H - 1 + \sum_{j=1}^{n} y_j}, \tag{8}$$

де $1 < H \equiv Const < 2$ деяка постійна величина, $n-$ число компонентів вхідного вектора, причому число нейронів у шарі розпізнавання (див. блок розпізнавання) дорівнює P. У формулі y_i – компонент вектора $Y = \{y_i\}$, а нейрон, який виявився збудженим, має номер k. Також змінюємо

матрицю $V_k = \{v_{ik}\}$ так, щоб $v_{ik} = y_i$. На цьому навчання закінчується. Важливо відзначити, що вхідний сигнал мережею сприймається за рахунок навчання матриці $W_k = \{w_{ik}\}$.

Розпізнавання: Виходи системи АБО–2 мають всі нульові значення. Це також робить виходи нейронів шару розпізнавання нульовими. На вхід мережі подається ненульовий вектор X, що встановлює рівень одиниці на виході схеми АБО–2. Це забезпечує проходження X на виходи нейронів шару порівняння без змін, при цьому $Y = X$.

Для кожного k-го нейрона шару розпізнавання (в блоці розпізнавання) визначається згортка вагового вектора W_k з вектором Y. При максимальному значенні згортки на виході якогось нейрона (що відповідає більшій відповідності до вхідного вектора), він переходить в активний – одиничний стан. Цей резонанс пригнічує сусідів. Так (через латерально-гальмуючу схему взаємодії) формується – визначається нейрон-переможець.

Порівняння: Вихід k-го збудженого нейрона шару розпізнавання подається на вхід кожного i-го нейрона шару порівняння (з вагою його синапсу), тобто на входи z_i нейрона шару порівняння подається або сильний сигнал, або слабкий (1 або 0). Оберний зв'язок за допомогою ненульового вектора $R = \{r_i\}$ виводить АБО1 на нуль у тому сенсі, якщо компоненти векторів $X = \{x_i\}$ та $Z = \{z_i\}$ не збігаються.

Якщо $y^{(m)} = \sum_i c_i \cdot y_i$ (причому $\sum_i c_i = 1$) і x_i великі і навіть рівні одиницям (тобто компоненти векторів Z і X практично збігаються), то вихідний сигнал у шарі розпізнавання Y залишається, сигналізуючи, що вектор відповідає збереженому зразку X (це і називають класифікацією).

Якщо Z і X сильно відрізняються, тобто подібності недостатньо, то збудження з відповідного нейрона шару розпізнавання знімається блоком АБО 1. Потрібно шукати інший нейрон в цьому шарі, який забезпечить необхідний рівень подібності запам'ятованого вектора з введеним вектором X або його частиною.

Література до розділу 6

[1] Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. – 2-е изд. – М.: Горячая линия - Телеком, 2002. – 382 с.

Розділ 7. НЕЙРОННІ МЕРЕЖІ НА БАЗІ НЕЧІТКОЇ ЛОГІКИ

Дуже вчасно професором Каліфорнійського університету Лотфі Заде була створена теорія нечітких множин та нечіткої логіки (1965). Нечітка логіка застосовується для задач слабо формалізованих і задач з ненадійними умовами і з невизначеними виразами; розташовується між формальним і природним описами; описує − апроксимує будь-яку математичну модель (теорема FAT − Fuzzy Approximation Theorem, B. Kosko, 1993); але в дійсності вихідний набір даних часто не тільки неповний, а й злегка суперечливий, введені експертами змінні входу і виходу можуть виявитися такими, що не цілком адекватно описують реальність; тому з необхідністю від систем вимагають адаптивності, тобто вони повинні допускати корекцію знань і параметрів у процесі вирішення завдань.

Введена була ***характеристична функція – функція належності*** елемента до множини, що приймає значення від нуля до одиниці або до іншої величини − верхньої межі (в чітких множинах тільки 0 і 1), тобто $\mu_A(x)$ − це характеристична функція, що в свою чергу приймає значення на множині *M*. ***Нечіткі (fuzzy) множини*** (L. Zadeh, 1965) – це множини елементів, що описуються характеристичними функціями.

Лотфи А. Заде (L. Zade)

Барт Коско (B. Kosko)

Ж.–С. Р. Чанг (J.–S. Roger Jang)

Мережі на основі нечіткої логіки. Об'єднання ближчої до реальності природної мови і сформованих людиною понять системи нечіткої логіки зі штучними нейронними мережами було вперше виконане Ж.–С. Р. Чангом з Тайванського університету.

Нечіткі нейронні мережі побудовані за таким принципом: висновки робляться на основі апарату нечіткої логіки, а функції належності підлаштовуються з використанням алгоритмів навчання нейронних мереж (див., наприклад, [1, 2]).

Роль нечіткої логіки виявляється настільки великою, що дозволяє повернутися до одвічної проблеми усвідомлення пошуку рішень нейронними мережами загалом і людським розумом зокрема. Справа в тому, що запропонована давніми вченими система логічних і математичних методів містить далекі для нейронних мереж способи опису і процедури розв'язань. Нейронна мережа шукає відповіді, використовуючи інші принципи і підходи, засновані на порівняннях «більше–менше». Дійсно, ступаючи на проїжджу частину, людина ніколи не вирішує в розумі завдання зустрічі, спираючись на математичний апарат, а оцінює можливість наїзду автомобілем, що наближається, на основі порівняння спостережуваної ситуації з попереднім досвідом. Природа створила унікальний інструмент − людський мозок, який є нейронною мережею з сукупністю допоміжних систем, що забезпечують його ефективне функціонування. Він діє приблизно так, як описано нечіткою логікою, де кожен предмет, явище або дія має численні сторони та грані. Оцінка цих елементів, що становлять предмет, явище або дію, може бути представлена деякими характеристичними функціями, що визначають ступінь належності до певного поняття. Схоже, що пошук відповіді в нейронній мережі відбувається подібно до того, як описав у своїй математичній теорії Лотфі А. Заде. Безсумнівно, що людство неодмінно усвідомить важливість такого опису і спробує зрозуміти, як саме влаштована система прийняття рішень природним і штучним інтелектом у формі нейронних мереж. І тоді, освоївши мову нейронних мереж, прийнявши на озброєння методи його функціонування, люди знайдуть можливість спілкуватися один з одним[39], не використовуючи збіднені

[39] І з клонами свідомості інших людей, навіть померлих. Про можливість такого спілкування йшлося на конференції CES Asia в Шанхаї в 2016 році, за допомогою додатків, спроби

емоціями і почуттями сучасні мови, що не переводять інтуїтивні образи і здогади у формальні символьні описи. Залишаючи останнім тільки прямі розрахунки і обчислення, де інтуїція і здогади вже виконали свою роль.

Наприклад, у тришарових мережах перший шар вводить нечіткість (fuzzification) на основі функцій належності (характеристичних функцій), другий застосовує нечіткі правила, третій приводить до чіткості (defuzzification). Можна використовувати меншу кількість шарів. Варто звернути увагу, що сигмоїдні функції нейронів цілком здатні виконувати роль операторів нечіткої логіки. Нагадаємо, що при сигналах на входи нейронів формується сумарна зважена величина, яка зазвичай обробляється активаційною (сигмоїдною) функцією.

Важливо також звернути увагу на неперебoрну неоднозначність відповіді в формі результуючої характеристичної функції, отриманої в процесі застосування нечіткої логіки. Тобто, взагалі кажучи, з позицій чіткої логіки, відсутня однозначність виводу. Оскільки перехід до чіткості строго не визначений і вибір конкретного значення характеристичної функції при переході до чіткості є більше мистецтвом, ніж методикою. Як і в нейронних мережах, які замкнуті на великі аудиторії користувачів і агентів (великі мережі, що створені останнім часом), так і в природному інтелекті, більшість завдань вирішується методами, подібними методами нечіткої логіки, що не забезпечує однозначність відповідей. Вибір відповіді часто здійснюється на нестрогих міркуваннях (або перевагах), на оптимальному (нехай навіть і нестрогому) узгодженні обраного рішення з великим набором інших раніше отриманих в тій чи іншій мірі подібних рішень, що знаходяться в глобальній базі даних (пам'яті).

Приклад: Найпростіша нечітка нейронна мережа, яка використовує два правила:

П1: якщо x є A1, то $y = z_1$

і

П2: якщо x є A2, то $y = z_2$.

Введення нечіткості: нечіткі поняття тут представлені сигмоїдними функціями належності $A_1(x) = \{1 + \exp[-b(x-a)]\}^{-1}$ і $A_2(x) = \{1 + \exp[b(x-a)]\}^{-1}$, до речі, сума яких в даному випадку дорівнює одиниці (хоча це не настільки важливо).

Апарат нечітких множин: ступені істинності правил визначаються $\alpha_1 = A_1$ і $\alpha_2 = A_2$.

розробок яких вже робляться великими комп'ютерними фірмами світу. Для цього потрібно навчитися спілкуватися з нейронною мережею її мовою.

Перехід до чіткості: результат на виході визначається зваженим середнім: $o = \dfrac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2} = \dfrac{A_1(x)\cdot z_1 + A(x)\cdot z_2}{A_1(x) + A(x)}$.

Відзначимо, що ***вибір цього переходу неоднозначний***.

Якщо функція помилки $E_k = \dfrac{1}{2}[o_k - y_k]^2$ і множина правильних рішень для навчання $Y = \{y_k\}$, то корекція окремих рішень кожного правила ***при навчанні*** відповідає процедурам

$$z_1 := z_1 - \eta \frac{\partial E_k}{\partial z_1} = z_1 - \eta \cdot (o_k - y_k) \cdot A_1(x_k)$$

та $$z_2 := z_1 - \eta \frac{\partial E_k}{\partial z_2} = z_2 - \eta \cdot (o_k - y_k) \cdot A_2(x_k).$$

У процесі навчання підлаштовувати треба також параметри *a* і *b*, виходячи з виду функції помилки $E_k = \dfrac{1}{2}[o_k - y_k]^2$), тобто шляхом присвоєння

$$a := a - \eta \frac{\partial E_k}{\partial a} = a - \eta \cdot (o_k - y_k)(z_1 - z_2) \cdot b \cdot A_1(x_k) \cdot A_2(x_k),$$

$$b := b - \eta \frac{\partial E_k}{\partial b} = a - \eta \cdot (o_k - y_k)(z_1 - z_2)(x_k - a) \cdot A_1(x_k) \cdot A_2(x_k).$$

Норма і конорма. Визначимо такі операції $\min(\mu_A, \mu_B)$; $\mu_A \cdot \mu_B$; $\max(0, \mu_A + \mu_B - 1)$; . Вони належать до так званої трикутної норми (t-норма), якщо всі $\mu_A(x)$ знаходяться в інтервалі x [0,1].

Властивості норми:

1. $T(0,0) = 0; T(\mu_A, 1) = T(1, \mu_A) = \mu_A;$ – обмеженість;
2. монотонність;
3. $T(\mu_A, \mu_B) = T(\mu_B, \mu_A);$ – комутативність;
4. $T(\mu_A, T(\mu_B, \mu_C)) = T(T(\mu_A, \mu_B), \mu_C);$ – асоціативність.

Для операцій $\max(\mu_A, \mu_B)$; $\mu_A + \mu_B - \mu_A \cdot \mu_B$; $\min(1, \mu_A + \mu_B)$; корисно ввести поняття трикутної конорми (t-конорма).

Властивості конорми:

1. $S(1,1) = 1; S(\mu_A, 0) = S(0, \mu_A) = \mu_A;$ – обмеженість;

2. монотонність;

3. $S(\mu_A, \mu_B) = S(\mu_B, \mu_A)$; − комутативність;

4. $S(\mu_A, T(\mu_B, \mu_C)) = S(T(\mu_A, \mu_B), \mu_C)$; − асоціативність.

Нечіткі нейрони. Нехай x − вхідні сигнали, а w − синаптичні ваги нейрона.

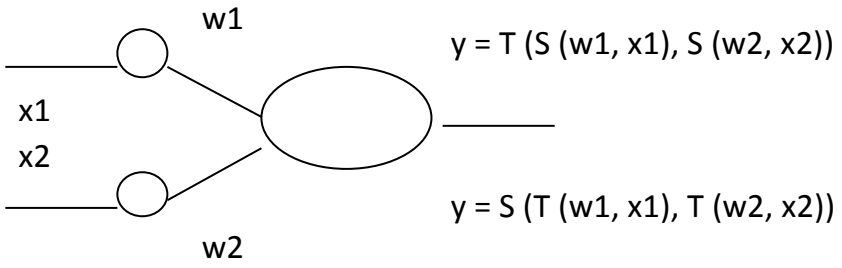


Рис. 13. Нечіткі нейрони

Визначимо нейрон «І» як $y = T(p1, p2) = T(S(w1, x1), S(w2, x2))$, $p1 = S(w1, x1)$.

Визначимо нейрон «АБО» як $y = S(p1, p2) = S(T(w1, x1), T(w2, x2))$, $p1 = T(w1, x1)$.

Роботу нейрона І можна проілюструвати таким чином. Спочатку вибираються сигнали великої кількості агентів, які перевищують поріг (у кожного агента він може бути різним) їх сприйняття нейроном. Цей поріг заданий синоптичною вагою. З тих сигналів, які перевищили поріг, і з фону інших синоптичних ваг обирається мінімальне значення, яке і є реакцією нейрона[40]. Нейрон АБО – це, навпаки, вибір максимального значення з безлічі сигналів, що не перевищують допустимий поріг (визначається для кожного агента синоптичною вагою) інших синоптичних ваг[41].

Взагалі кажучи, можна простежити зв'язок між принципами нечіткої логіки і роботою нейронних мереж. Багато алгоритмів нечіткої логіки побудовані на кон'юнкції вхідних даних, яка набуває форму вибору мінімального сигналу (жорсткий зв'язок, жорсткий відбір) або змішування сигналів (множення характеристичних функцій − м'який зв'язок, м'яка

[40] Це подібно до того, як владна структура вибирає свою мінімальну реакцію на почуті вимоги активних людей з натовпу.

[41] Тут це схоже на те, як влада вибирає найбільш ефективне рішення з усіх пропозицій, які не порушують деякі обмеження, наприклад, відкидають надмірно радикальні і неприйнятні пропозиції.

взаємодія, участь), потім композиція результатів набуває форму диз'юнкції (вибір або максимального сигналу, або складання сигналів). У реальної нейронної мережі слабкі вхідні сигнали посилюються, що не можна сказати про сигнали сильні. Подальші операції подібні до композиції, синтезу сигналів.

Подання нейронної мережі з нечіткими нейронами (реалізує алгоритм Tsukamoto).

П1: якщо x1 є L1, якщо x2 є L2, якщо x3 є L3, то y є G,
П2: якщо x1 є H1, якщо x2 є H2, якщо x3 є L3, то y є P,
П3: якщо x1 є H1, якщо x2 є H2, якщо x3 є H3, то y є R,

де явний вигляд функцій

$$L_j(x_i) = 1/[1-\exp\{b_j(x_i - c_j)\}],$$
$$H_j(x_i) = 1/[1+\exp\{-b_j(x_i - c_j\}],$$
$$G(z_1) = 1/[1+\exp\{-b_4(z_1 - c_4 + c_5)\}],$$
$$P(z_2) = 1/[1+\exp\{-b_4(z_2 - c_4)\}],$$
$$R(z_3) = 1/[1+\exp\{-b_4(z_3 - c_4)\}],$$

анілітичне представлення використання правил

$$z_1 = G^{-1}(\alpha_1) = c_4 + c_5 + \frac{1}{b_4}\ln\frac{1-\alpha_1}{\alpha_1},$$
$$z_2 = P^{-1}(\alpha_2) = c_4 + \frac{1}{b_4}\ln\frac{1-\alpha_2}{\alpha_2},$$
$$z_3 = R^{-1}(\alpha_3) = c_4 + \frac{1}{b_4}\ln\frac{1-\alpha_3}{\alpha_3}.$$

Визначимо рівні відсікання як (другий шар мережі)

$$\alpha_1 = L_1 \wedge L_2 \wedge L_3,$$

$$\alpha_2 = H_1 \wedge H_2 \wedge L_3,$$

$$\alpha_3 = H_1 \wedge H_2 \wedge H_3.$$

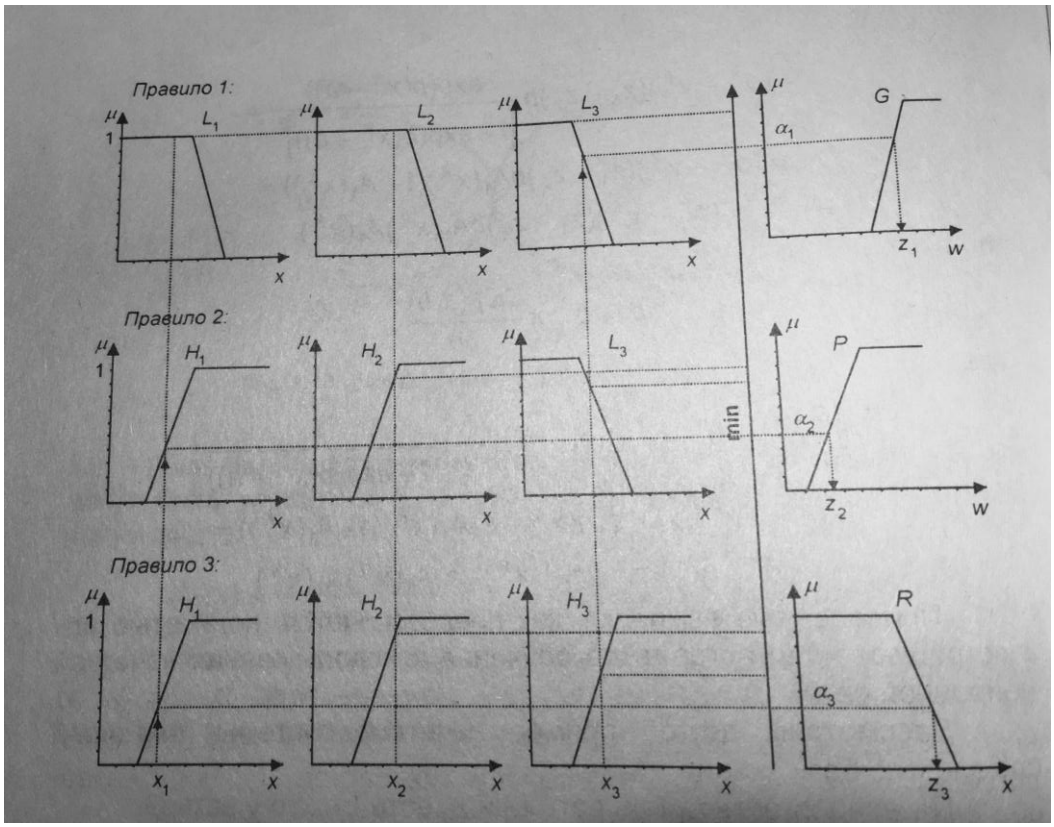


Рис. 14. Ілюстрація використання правил П1-П3 [3]

Тут рівні відсікання $\alpha-$ це є результат дії нейронів «І», наприклад, мінімум всіх змінних L або H. Наприклад, дію нейрона «І» для отримання $y \Rightarrow \alpha_1 = L_1(x1) \wedge L_2(x2) \wedge L_3(x3)$ з урахуванням синаптичних ваг даного нейрона на кожному каналі введення можна представити у вигляді

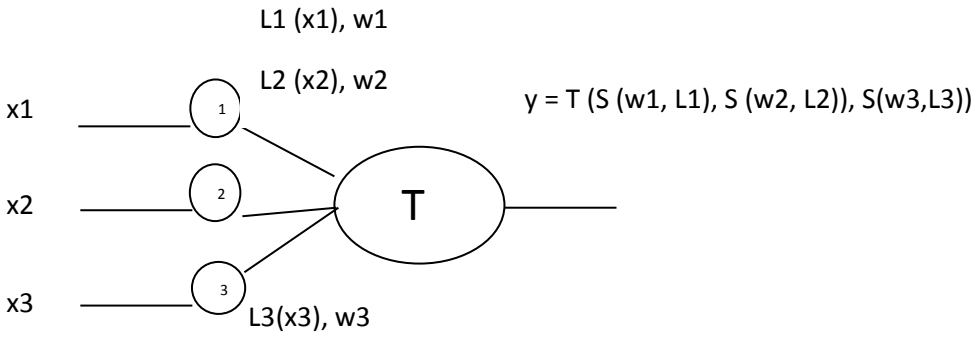


Рис. 15. Нейрон для створення мережі

До речі, наявність синаптичних ваг дозволяє розширити можливості алгоритму, ***фактично перетворюючи його в нейронну мережу***. Нейронна мережа з урахуванням наведеного вище пояснення набуває вигляду:

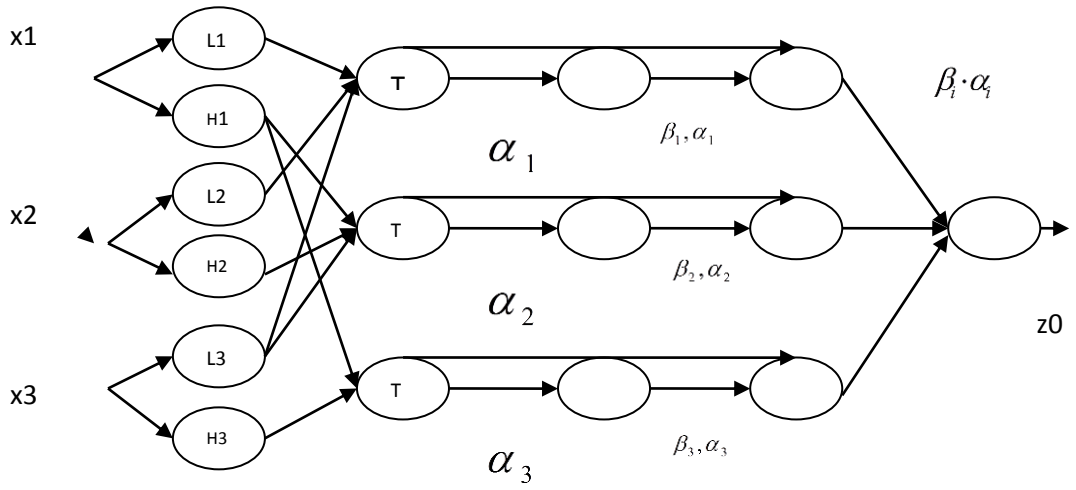


Рис. 16. Нейронна мережа на основі нечіткої логіки

Визначимо також відображення (четвертий шар мережі)

$$z_1 = G^{-1}(\alpha_1),$$

$$z_2 = P^{-1}(\alpha_2),$$

$$z_3 = R^{-1}(\alpha_3),$$

де, наприклад, $G^{-1}-$ функція, зворотна функції G.

А також приведення до чіткості

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3}{\alpha_1 + \alpha_2 + \alpha_3},$$

для чого корисно визначити величини виду $\beta_i = \frac{\alpha_i}{\alpha_1 + \alpha_2 + \alpha_3}$ (третій шар мережі).

Таку мережу можна вчити, для цього представимо корекцію функцій z_i, якщо є навчальний набір $\{y_k\}$ для функції помилки $E_k = \frac{1}{2}(z_0 - y_k)^2$,

$$z_i := z_i - \eta \cdot \frac{\partial E_k}{\partial z_i} = (z_0 - y_k)\frac{\alpha_i}{\alpha_1 + \alpha_2 + \alpha_3},$$

а також відповідну корекцію параметрів в такому вигляді:

$$b_4 := b_4 - \eta \cdot \frac{\partial E_k}{\partial b_4} = b_4 - \frac{\eta}{b_4^2} \delta_k \cdot \frac{\alpha_1 + \alpha_2 - \alpha_3}{\alpha_1 + \alpha_2 + \alpha_3},$$

$$c_4 := c_4 - \eta \cdot \frac{\partial E_k}{\partial c_4} = c_4 + \eta \cdot \delta_k,$$

$$c_5 := c_5 - \eta \cdot \frac{\partial E_k}{\partial c_5} = c_5 \cdot \delta_k \frac{\alpha_1}{\alpha_1 + \alpha_2 + \alpha_3}.$$

До речі, той факт, що в основі такої мережі лежить деякий цілком певний алгоритм, може викликати сумніви в застосовності для таких систем поняття «нейронна мережа». Навіть наявність нечітких нейронів не дає підстави вважати їх повноцінними нейронами, хоча тут є помітна їх схожість з нейронами природними і штучними. Але треба не забувати, що варіанти нейронних мереж (спеціалізованих для різних застосувань), зібрані у величезних бібліотеках, подібні в принципі мережам, побудованим на алгоритмах нечіткої логіки. Оскільки вибір архітектури спеціалізованої штучної нейронної мережі подібний вибору алгоритму мережі на нечіткій логіці, а деяке свавілля відповідно архітектури та алгоритму формують вже нейрони, що настроюються. Об'єднує природні і штучні нейронні мережі і мережі на основі нечіткої логіки розмиття істинності і хибності у формі функцій належності. А також можливість налаштування нечітких нейронів, тобто можливість навчання мережі, нехай і в певних межах, пов'язаних з вибором базового алгоритму.

Введені в опис так звані нечіткі нейрони дозволили зрозуміти характер роботи таких нейронних систем з нечіткою логікою, і з'явилася можливість переглядати процес вирішення. Ці системи дозволяють, правда на силу, але все-таки розуміти, що відбувається зі знанням у процесі розв’язання такою нейронною мережею задач. Таким чином відбулося ***історичне об'єднання штучних нейронних мереж і експертних систем прийняття рішень на основі логіки***. Цей процес ще не завершений, але його темп вселяє впевненість в його успішності.

Стандарти застосування

ANFIS (adaptive network–based fuzzy inference system) – 1. на вході − значення переводяться в функції належності, 2. T − норма, 3. нормалізація

значення, 4. формується нечіткий висновок, 5. дефазифікація. Налаштування за допомогою алгоритму зворотного поширення помилки.

NNFLC (neural−network−based fuzzy logic control system = controller) − об'єднання нечіткого контролера і нейронної мережі.

Література до розділу 7

1. Куклін В. М. Подання знань і операції над ними: навчальний посібник / В. М. Куклін. Х.: ХНУ імені В. Н. Каразіна, 2019. 164 с.

2. Куклин В. М. Представление знаний и операции над ними: учебное пособие / В. М. Куклин. Х.: ХНУ имени В. Н. Каразина, 2019. 180 с. (http://dspace.univer.kharkov.ua/handle/123456789/15167)

3. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. – 2-е изд. – М.: Горячая линия - Телеком, 2002. – 382 с.

ЧАСТИНА ІІ

СУЧАСНИЙ РОЗВИТОК НЕЙРОМЕРЕЖ

Розділ. 8. РЕВОЛЮЦІЯ У ВИВЧЕННІ ТА ЗАСТОСУВАННІ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

Вибухове зростання числа нейронів в сучасних мережах, різні способи їх з'єднання (локально пов'язані групи і шари нейронів, рекурентні зв'язки і т. д.), ускладнення архітектури, зростання обчислювальної потужності привели до появи нових поколінь мереж і розширили можливості їх оптимізації та навчання, що стало називатися глибоким навчанням. Зміст глибокого навчання весь час піддається корекції, різні автори монографій і авторитетні фахівці не можуть домовитися про основні акценти.

Найбільш поширені архітектури – це мережі, навчені з учителем і без учителя; згорткові мережі; рекурентні і рекурсивні мережі. Глибоке навчання з підкріпленням тепер вже розуміється як метод дати свободу мережі експериментувати, лише оцінюючи її дії. Пропонуються вхідні дані і результати винагороджуються в більшій чи меншій мірі. На дії нейронної мережі важко тепер накласти обмеження. Відмітна особливість глибокого навчання на загальну думку – це перехід від ручного формування ознак[42] до автоматичного створення великої кількості ознак вищого порядку при внесенні на вхід мережі необроблених даних, для спрощення класифікації.

Нові технічні можливості дозволили на багато порядків збільшити кількість ознак, які мережа знаходить в даних за рахунок ускладнення та збільшення кількості різноманітних процедур. Це дозволило значно краще робити порівняння та обчислення. Це революційне ***збільшення можливостей мережі обчислювати велику кількість ознак та їх взаємозв'язок*** зробило можливим значний прогрес у розвитку штучного інтелекту.

Що ж таке глибоке навчання і чому воно справило революцію у цій галузі знання – штучному інтелекті? Глибоке навчання (англ. Deep

[42] Хоча повністю від цього відмовитися не вдається, особливо при роботі з природною мовою.

Learning) — сукупність методів машинного навчання (з учителем, з частковим залученням вчителя, без вчителя, з підкріпленням), що ґрунтуються на навчанні уявлень, а не на спеціалізованих алгоритмах під конкретні завдання.

Глибинна нейронна мережа (ГНМ, англ. DNN – Deep Neural Network) – це штучна нейронна мережа (ШНМ) з кількома шарами між вхідним та вихідним шарами. DNN знаходить коректний метод математичних перетворень, щоб перетворити вхідні дані на дані на виході, незалежно від лінійної або нелінійної кореляції. Мережа просувається шарами, розраховуючи ймовірність кожного виходу. Наприклад, DNN, яка навчена розпізнавати породи собак, пройде за заданим зображенням і обчислить ймовірність того, що собака на зображенні належить до певної породи. Користувач може переглянути результати та вибрати ймовірності, які має відображати мережа (вище певного порога, наприклад), та повернути до мережі запропоновану мітку. Кожне математичне перетворення вважається шаром, а складні DNN мають багато шарів, звідси й назва «глибинні» чи «глибокі» мережі.

Глибоке навчання характеризується як клас алгоритмів машинного навчання, який:

• використовує багатошарову систему нелінійних фільтрів для отримання ознак із перетвореннями. Кожен наступний шар одержує на вході вихідні дані попереднього шару. Система глибокого навчання може поєднувати алгоритми навчання з учителем і без вчителя. Скажимо, аналіз зразка є навчанням з вчителем, а класифікація — навчання без учителя;

• має кілька шарів виявлення ознак або параметрів подання даних (навчання без вчителя). При цьому ознаки організовані ієрархічно, ознаки вищого рівня похідні від ознак нижчого рівня;

• є частиною ширшої галузі машинного навчання вивчення подання даних;

• формує у процесі навчання шари на кількох рівнях уявлень, які відповідають різним рівням абстракції; верстви утворюють ієрархію понять.

Багато методів глибокого навчання були відомі ще в 1980-ті і навіть раніше, але результати були невражаючими, поки просування в теорії штучних нейронних мереж (перекриття нейромереж за допомогою спеціального випадку ненаправленої графічної моделі, так званої обмеженої машини Больцмана) і обчислювальні потужності середини 2000-х років (у тому числі ті, що використовують графічні прискорювачі, програмовані користувачем вентильні матриці та різні форми нейронних процесорів) не

дозволили створювати складні технологічні архітектури нейронних мереж, що володіють достатньою продуктивністю і дозволяють вирішувати широкий спектр завдань, наприклад, у комп'ютерному зорі, машинному перекладі, розпізнаванні промови, причому якість рішень мережі можна порівняти з ефективністю людини, а в деяких випадках вона краще.

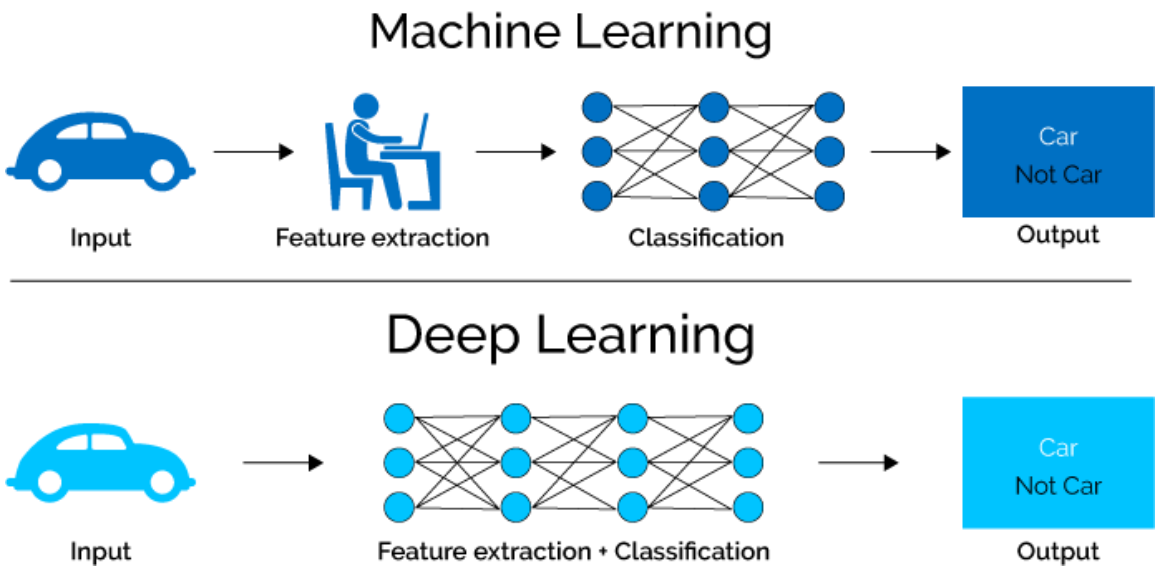


Рис. 17. Різниця між машинним та глибоким навчанням: у першому випадку користувач повідомляє алгоритму ознаки, якими можна відрізнити автомобіль від інших об'єктів. У разі глибокого навчання алгоритм сам виділяє різні ознаки різних об'єктів і навчається розділяти об'єкти за цими ознаками

Івахненко А. Г. (1913–2007) – радянський український вчений в галузі управління

Галушкин О. І. (1940–2016) – радянський та російський вчений в галузі інтелектуального управління

Ідеї глибоких нейронних мереж мають майже таку ж довгу історію, як і самі штучні нейронні мережі. Глибокі мережі з'явилися ще в середині 1960-х років. Скоріше за все перші справжні глибокі мережі у вигляді глибоких перцептронів були описані у роботах радянського вченого, українця О. Г. Івахненка [1]. Він розробив так званий метод групового обліку аргументів, суть якого виглядає приблизно так:

• спочатку ми вибираємо загальний вигляд, параметричну родину моделей, які навчатимемо;

• будуємо та навчаємо різні варіанти обраних моделей;

• вибираємо за допомогою метрики якості кілька найкращих моделей; якщо потрібну якість вже досягнуто, можна нічого далі не робити;

• але якщо ще не досягнуто – і це ключовий момент – ми починаємо будувати моделі наступного рівня, використовуючи виходи підібраних на попередньому кроці моделей як входи для наступних;

• цей процес можна рекурсивно повторювати доти, доки якість моделі або досягне потрібного рівня, або перестане поліпшуватися.

Метод групового обліку аргументів виглядає на подив сучасно. Якщо в ньому в якості базової моделі вибрати перцептрон, ми отримаємо типову нейронну мережу з кількома шарами, яка навчається шар за шаром: спочатку перший, потім він фіксується і починається навчання другого, і т.д. цілком успішно навчалися моделі до семи рівнів вглиб.

Сеппо Ліннемаа (1945 р. нар.), фінський математик

Фукусіма К. (1936 р. нар.) комп'ютерний спеціаліст з Японії

У серпні 1974 року тиражем 8 000 примірників вийшла книга Олександра Івановича Галушкіна, тоді — співробітника МФТІ, під назвою «Синтез многослойных систем распознавания образов» [2]. З погляду сучасної термінології назву книги можна розуміти як «Навчання багатошарових нейронних мереж». О. І. Галушкін розглядає завдання навчання нейронної мережі як завдання градієнтного спуску. Він застосовує ланцюгове правило для обчислення градієнта, докладно розглядає випадок навчання мереж з двома шарами, що навчаються, а також коротко показує, як слід вирішувати завдання у випадку багатошарових мереж і мереж з рекурентними зв'язками, але не дає методу якоїсь власної назви. При цьому питання конкретної топології мереж у книзі практично не торкається, зате приділено увагу мережам зі змінною кількістю шарів: О. І. Галушкін описує підхід, що нагадує мережу з пошаровим навчанням і шарами, що поступово нарощуються. Окрім математичних моделей, О. І. Галушкін спільно з В. Х. Нарімановим ще на початку 1970-х років сконструював власну версію апаратного персептрона[43].

Проте розвиток нейронних мереж у результаті пішов трохи іншим шляхом. Першою глибокою нейронною мережею вже можна було вважати Neocognitron Куніхіро Фукусіми, в якому з'явилися і згорткові мережі, і активації, дуже схожі на ReLU [3]. Однак ця модель не навчалася в сучасному значенні цього слова: ваги мережі встановлювалися з локальних правил навчання без вчителя. Приблизно водночас з'явилися й глибокі моделі з урахуванням зворотного поширення. Першим застосуванням зворотного поширення помилки до довільних архітектур можна вважати роботи фінського тоді ще студента Сеппо Лінненмаа[44]: в 1970 р. він побудував правила автоматичного диференціювання за графом обчислень, у тому числі і зворотне поширення [4].

[43] У ті роки і в західному, і в радянському світі нейрони не називали нейронами: у Розенблата в «Принципах нейродинаміки» вони називаються модулями (units), а те, що ми зараз знаємо як «нейросети», у радянській традиції мало кілька інших назв. Багатошаровий перцептрон (multilayer perceptron) у Івахненка називався багаторядним, замість нейронів учений використовував термін «змінні», замість мережі – «фільтр». У О.І. Галушкіна мережа називалася «системою розпізнавання», нейрон – «лінійно-пороговим елементом», а навчена мережа (тобто мережа, призначена тільки для виконання [inference]) – «мережею з розімкненим контуром». Чому вчені намагалися не використовувати слово нейрон? А тому, що це посилання на біологічний прототип — нейрон у корі головного мозку. Шари – це теж цілком нейробіологічний термін, успадкований від шарів клітин кори головного мозку, тому в О.Г. Івахненка перцептрон – багаторядний.

[44] С. Лінненмаа отримав перший ступінь Ph.D. з інформатики в історії університету Хельсинки. А роботу про зворотнє поширення він захистив у якості диплома магістра (M.Sc.).

А навіщо взагалі потрібні глибокі мережі? Класична теорема Курта Хорника, заснована на більш ранніх роботах А.Н. Колмогорова, стверджує, що будь-яку безперервну функцію можна як завгодно точно наблизити нейронною мережею з одним прихованим рівнем. Здавалося б, цього має бути достатньо. Навіщо плодити надмірну складність?

Справа в тому, що глибокі архітектури часто дозволяють висловити те саме, наблизити ті ж функції набагато ефективніше, ніж неглибокі: одну і ту функцію часто можна набагато краще наблизити глибшою мережею, ніж дрібною, навіть якщо загальну кількість нейронів у мережі залишити постійним.

А інший бік можливостей глибоких мереж — те, що глибока нейронна мережа створює не просто глибоке, а й розподілене уявлення. Тут йдеться про те, що кожен рівень глибокої мережі складається не з одного нейрона, а відразу з багатьох, і комбінації значень цих нейронів справляють справжнісінький експоненційний вибух у просторі входів!

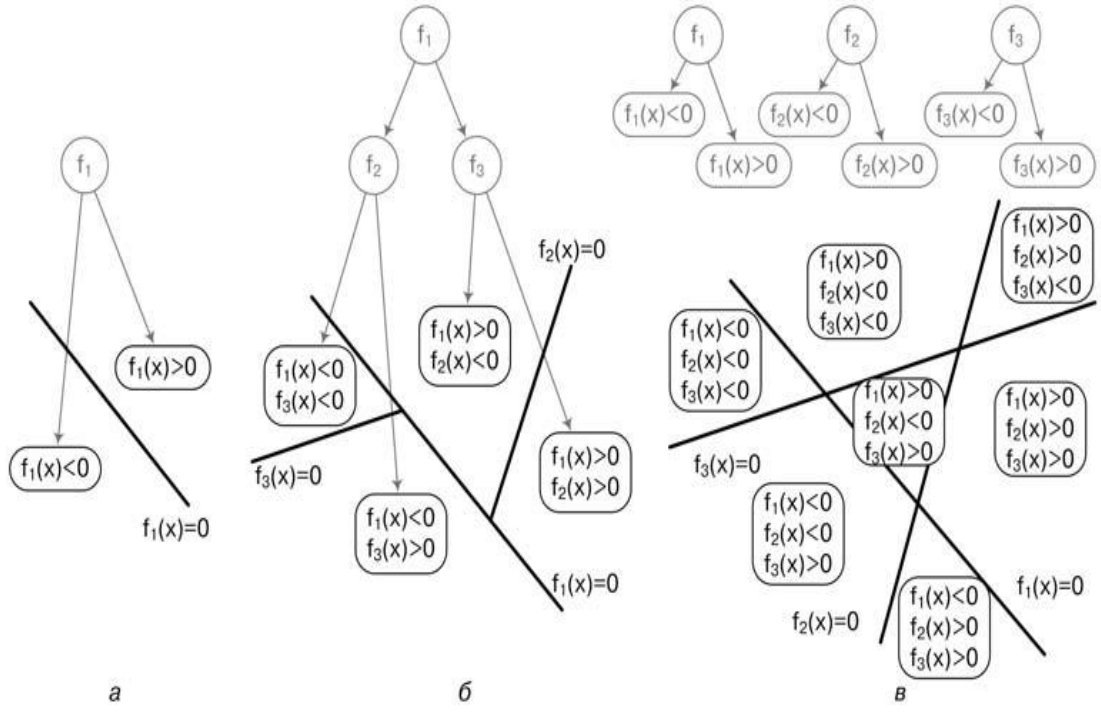


Рис. 18. Сила розподілених уявлень: а – розділяюча поверхня однієї ознаки; б – розділяюча поверхня дерева глибини 2 на трьох ознаках; в – розділяюча поверхня трьох ознак

Це можна проілюструвати прикладом на рис. 18, подібний до якого часто наводить у своїх статтях і книгах Йошуа Бенджі [5]. Уявімо, що ми можемо розділяти точки на площині за допомогою трьох можливих ознак, кожна з яких – це лінійна функція: f1, f2, f3. На рис. 18, а зображена

розділяюча поверхня по одному з них, тобто пряма на площині, і розмічені частини, на які ділить пряма пряма. Частин цих, зрозуміло, дві. Як слід поєднувати ці ознаки у нашому класифікаторі? По-перше, скільки в нього листя: щоб розібрати чотири різні можливі випадки, нам потрібно було побудувати дерево з чотирма листками. А на рис. 18, в зображено розподілене уявлення: уявімо, що наш «другий шар» може складатися з трьох нейронів відразу. Тоді всілякі комбінації цих трьох нейронів можуть розпізнати одночасно 2 = 8 різних випадків (на рисунку їх реалізується 7), і це число різних варіантів зростає експоненційно при зростанні числа нейронів.

Тепер виникає друге питання – чому глибокі мережі, в яких немає нічого теоретично складного і які з'явилися майже одночасно з алгоритмом зворотного розповсюдження помилки, так довго викликали такі складнощі? Чому революція глибокого навчання відбулася в середині 2000-х, а не 1980-х? На це питання є дві відповіді [6].

Перша – математична. Справа в тому, що глибокі нейронні мережі навчати, звичайно, можна тим же алгоритмом градієнтного спуску, але в базовому варіанті без додаткових хитрощів працювати це не буде. Уявіть, що ви почали навчати глибоку мережу алгоритмом зворотного поширення помилки. Останній, найближчий до виходів, рівень навчиться досить швидко і дуже добре. Але що буде далі? Далі виявиться, що більшість нейронів останнього рівня на всіх тестових прикладах вже «визначилися» зі своїм значенням, тобто їх вихід близький або до нуля, або до одиниці. Якщо в них класична сигмоїдальна функція активації, то це означає, що похідна у цієї функції активації близька до нуля з обох сторін, але на цю похідну ми повинні помножити всі градієнти в алгоритмі зворотного поширення!

Таким чином, виходить, що останній шар нейронів, що навчився, «блокує» поширення градієнтів далі назад за графом обчислень, і більш ранні рівні глибокої мережі в результаті навчаються дуже повільно, фактично стоять на місці. Цей ефект називається проблемою загасаючих градієнтів (vanishing gradients).

У випадку з рекурентними мережами, які фактично є дуже глибокими, виникає ще й зворотна проблема: іноді градієнти можуть почати «вибухати», експоненційно збільшуватися в міру «розгортання» нейронної мережі (exploding gradients). Обидві проблеми виникають часто, і досить довго дослідники не могли задовільно їх вирішити.

У середині 2000-х років з'явилися перші конструкції, що дійсно добре працюють і добре масштабуються, і вони дуже сильно нагадували вихідний механізм «глибоких моделей» О. Г. Івахненка! Рішення, яке запропонували в групі Джеффрі Хінтона, полягало в тому, щоб надавати

нейронні мережі рівень за рівнем за допомогою спеціального випадку ненаправленої графічної моделі, так званої обмеженої машини Больцмана (restricted Boltzmann machine, RBM). Справа в тому, що градієнтний спуск, звичайно, добре знаходить локальний мінімум функції, а різні варіанти градієнтного спуску здатні навіть вибиратися з невеликих локальних мінімумів і знаходити більш значний мінімум.

Але все одно градієнтний спуск по суті своїй локальний, він робить лише невеликі кроки в той чи інший бік від поточної точки, і результат залежить від ініціалізації, від того, з якої точки ми починатимемо. Тому навчання без вчителя може призвести до дуже гарного ефекту: за рахунок того, що модель вже починає трохи «розбиратися» в структурі даних, початкові значення ваг вже не випадкові, а досить розумні. Такі конструкції призвели до прориву спочатку у розпізнаванні мови, а потім і в обробці зображень.

Виходить, що просто так глибокі мережі навчати не вдається, і потрібно застосовувати різноманітні трюки, спочатку послідовно навчаючи окремі верстви мережі зовсім іншими алгоритмами, а потім локально «докручуючи» їх градієнтним спуском. Що сталося? Куди зникли всі колишні труднощі?

По-перше, в порівнянні з серединою 2000-х років, і тим більше в порівнянні з початком 1990-х, зараз ми все-таки краще вміємо навчати нейронні мережі. З'явився ряд важливих інструментів, які можна так чи інакше віднести або до регуляризації, або до різних модифікацій оптимізації в нейронних мережах.

В цілому, хоча основні конструкції нейронних мереж багато в чому прийшли до нас ще з 90-х, а то й 80-х років минулого століття, навчаємо ми їх зараз все-таки не найпростішим способом.

По-друге, на питання про те, чому глибокі нейронні мережі було складно навчати, є й інша відповідь: справа в тому, що раніше комп'ютери були повільнішими, а доступних для навчання даних було набагато менше, ніж зараз.

Для реалізації прикладів частини III, аналогічні мережі можна було навчити і на початку 1990-х років. Але, наприклад, навчання системи розпізнавання мови виглядало приблизно так: дослідники писали код і запускали навчання, яке тривало на тодішніх комп'ютерах тижнів зо два.

Причому відбувалося це двотижневе навчання на класичному датасеті TIMIT, який зараз вважається досить маленьким та використовується для швидких експериментів. Через два тижні дивилися на результат, розуміли, що, мабуть, потрібно було підкрутити той чи інший параметр (а їх у нейронних мережах дуже багато, як ми ще не раз побачимо), підкручували... і знову запускали навчання на два тижні. Це не сприяло ні тонкому

настроюванню мереж, ні своєчасним публікаціям до відповідних дедлайнів, ні технічній можливості успішно навчити нейронну мережу.

Успіхи глибокого навчання не були б настільки значними без використання матричної нотації – опису операцій з використанням матриць і тензорів, створення пакетів матричних операцій (таких як Numpy) та спеціальних мов (таких як Tensorfiow), автоматично адаптованих до застосування для розрахунків графічних процесорів (Graphics Processing Unit – GPU).

Пояснимо переваги матричної нотації. Звичайною практикою в сучасних мережах є введення відразу всіх навчальних прикладів у матричну обробку (необхідні зміни програмного забезпечення вже вбудовані, наприклад, у Numpy і Tensorflow). Пакет матричних операцій Numpy та мова Tensorfiow мають широкомовлення (broadcasting), тобто розширення всіх матриць до стандартного розміру за рахунок віртуальних рядків (зокрема це потрібно, коли вводять матрицю для векторів зсувів). Мова Tensorfiow має вбудоване програмне забезпечення, яке знаходить графічні процесори (Graphics Processing Unit – GPU), та використовує їх без зміни у початковому коді програми. Використання мови Python та пакета Numpy прискорює обчислення на порядок. Використання графічних процесорів (з урахуванням програмного забезпечення, що їх автоматично використовує) також часи обчислень зменшує на порядок. Навчання одразу на всіх прикладах за рахунок їх матричної обробки також скорочує час навчання, що не менш важливо. Наприклад, замість підстроювання кожної ваги методом градієнтного спуску $\Delta w_{i,j} = -\eta x_i \frac{\partial X(\Phi)}{\partial l_j}$, у матричній нотації останнє рівняння замінено на матричне рівняння $\nabla_l \vec{W} = -\eta \mathrm{X}^T \nabla_l X(\Phi)$. Тут вектор X має вже m рядків, що відповідають такій же кількості навчальних прикладів, причому у формулі X^T – вектор, який транспоновано. Варто звернути увагу на те, що створення архітектури нейронної мережі відбувається одночасно з конструюванням процедур її оптимізації. Тобто навчання мережі зараз нерозривно пов'язане із процесом її створення.

У середині 2000-х років все зійшлося воєдино: технічно комп'ютери стали досить потужними, щоб навчати великі нейронні мережі (більше того, обчислення в нейронних мережах незабаром навчилися делегувати відеокарт, що прискорило процес навчання ще на цілий порядок), набори даних стали досить великими, щоб навчання великих мереж мало сенс, а в математиці нейронних мереж відбулося чергове просування. І почалася та сама революція глибокого навчання, яка триває нині.

Література до розділу 8

1. Ивахненко А. Г., Лапа В. Г. Кибернетические предсказывающие устройства. Киев: Наук. думка, 1965.

2. Галушкин А. И. Синтез многослойных систем распознавания образов. М.: Энергия, 1974.

3. Fukushima K. Neocognitron: A Self-Organizing Neural Network for a Mechanism of Pattern Recognition Unaffected by Shift in Position // Biological Cybernetics, 1980, vol. 36, no. 4. P. 193–202.

4. Linnainmaa S. The Representation of the Cumulative Rounding Error of an Algorithm as A Taylor Expansion of the Local Rounding Errors, Master's thesis, Univ. Helsinki, 1970.

5. Бенджио И., Гудфеллоу Я., Курвилль А. Глубокое обучение. М.: ДМК-Пресс, 2018.

6. Кадурин А., Николенко С. И., Архангельская Е. В. Глубокое обучение. Погружение в мир нейронных сетей. СПБ.: Питер, 2018. 480 с.

Розділ. 9. НОВИЙ ЕТАП – ГЛИБОКЕ НАВЧАННЯ

Глибоке навчання – це продовження колишнього підходу в організації штучних нейронних мереж (ШНМ). Штучні нейронні мережі замислювалися як прилади, що створюються за зразком біологічного мозку. У нейронних мережах глибоке навчання теж орієнтоване на множинну аналогію з природним інтелектом. Тому аналізується робота мозку, вивчається його функціональність, і це стає відправною позицією для створення подібних систем у нейронній мережі. Крім того, глибоке навчання – це інструмент усвідомлення принципів організації природного розуму, а також розвитку принципів машинного навчання різних пристроїв, вже не обов'язково подібних нейронних мереж.

Зауважимо, що сучасні реалізації навчання нейронних мереж все більше засновані на застосуванні графічних процесорів (GPU), які розроблялися для графічних додатків, але пізніше використовувалися для організації паралельних обчислень складних завдань [1]. Для опису зображення обчислення у відеокарті досить прості і не містять такої великої кількості розгалужень, як у програмі, що виконується на CPU. Використання графічних карт для навчання нейронних мереж зазнало справжній вибух популярності з появою GPU загального призначення (GP–GPU), які вже здатні виконувати довільний код. Технологія програмування CUDA, розроблена за участі компанії NVIDIA, дала можливість написання довільного коду. Завдяки зручному програмуванню, паралелізму та пропускній здатності пам'яті GP–GPU стали ідеальною платформою для програмування нейронних мереж відразу після її появи (2010).

Алгоритми глибокого навчання варто застосовувати, коли мало відомо про предметну область, а збирання баз даних якісних ознак обтяжливо. Глибоке навчання застосовують, коли прості методи і системи не дають потрібного результату[45], коли потрібно шукати складні патерни в масивах, коли велика розмірність даних, коли треба враховувати часові виміри. Недостатнє знання предметної області (те, що машині пропонують допоміжні бази даних і знань) проте дозволяє технології глибокого навчання самостійно навчатися виділенню ознак, знаходити закономірності в масивах інформації, що збуджує нашу уяву і дозволяє приписувати

[45] Але люди почали усвідомлювати, що сучасне обладнання і зростання обчислювальних потужностей дозволяє зараз і звичайним повнозв'язним мережам домагатися потрібних результатів там, де раніше їх попередники, більш архаїчні мережі були безсилі. Можливо, колишні труднощі були подолані насамперед за рахунок збільшення обсягів, швидкодії і технічної досконалості мереж.

штучному інтелекту не властиві йому можливості. Однак людська свідомість все ще якісно відрізняється від можливостей штучного інтелекту[46], тобто існує гігантський розрив між уявленнями недосвідчених людей і реальністю[47].

Програмні методи опису та оптимізації мереж глибокого навчання [2]**.** Tensorflow – це біблотека та мова з відкритим вихідним кодом, розроблена Google для програмування у сфері глибокого навчання, його структурами даних є тензори (tensor). Функції TF визначають обчислення тільки за виклику команди run, але спочатку потрібний рядок import tensorflow. Зокрема функція TF Session створює сеанс, і з цим сеансом пов'язаний алгоритм обчислень. Раціонально користуватися інтерфейсом мови Python, хоча змінні з цієї мови, наприклад, x-константи TF, тут вже пов'язані з процедурами обчислень, лише дія функції sess. run(x) повертає значення константи TF. Змінні x і sess, функції import та print взяті з мови Python, a constant, Session та run – команди TF. Змінна Python може виконувати функцію заповнювача (placeholder) TF, але спосіб присвоєння їй значення відрізняється. Використовують тензори (їх видів близько півтора десятка, і вони названі). Заповнювач – тензор може мати тип, наприклад, float32. Поряд з типом тензор має форму. Наприклад, масив 3 x 17 x 6 має форму [3, 17, 6], скаляри вважаються теж тензорами, але вони мають нульову форму. Параметри TF створюються наступним чином: представляють тензор з початковими значеннями, перетворюють його на Variabe (TF так позначає параметри) та ініціалізують змінні. Наприклад, надамо програмне створення параметрів – зміщення b і ваги W:

```
bt = tf.random_normal([l0], stddev=.l)
b = tf.Variabe(bt)
W = tf.Variabe(tf.random_normal([784,10],stddev=.l))
sess=tf.Session()
sess.run(tf.global_variables_initializer())
print(sess.run(b))
```

Перший рядок програми додає інструкцію для створення тензора форми [10], десятьма значеннями якого є випадкові числа, створені з нормального розподілу (normal distribution) зі стандартним відхиленням 0,1;
Другий рядок отримує bt і додає фрагмент графа TF, який створює змінну з тими самими формою та значеннями;
Третій рядок створює параметри синапсів W, оскільки вихідний тензор після створення змінної не потрібен, у ньому можна поєднати дві події;

[46] Див. Russell S., Norvig P. 'Artificial intelligence: A modern apprroach' [3].
[47] Наприклад, машини поки не усвідомлюють себе, принаймні не проявляють свою примхливість.

Рядок 4 створює сеанс;
Рядок 5 ініціалізує b W;
Рядок 6 здійснює висновок.

Прямий прохід. Для одного шару лінійних блоків можна, слідуючи [2], уявити програму прямого проходу. Розглянемо для прикладу її першу частину-налаштування TF обчислень. Спочатку імпортуємо *Tensorflow* та код для читання даних *Mnist* (тут цілі числа від нуля до дев'яти).

0 import tensorflow as tf
1 from tensorflow.examples.tutorials.mnist import input_data
2 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
Визначимо, як це зроблено вище, набори параметрів
4 W tf.VariaЬle(tf.random_normal([784, 10],stddev=.l))
5 Ь = tf.VariaЬle(tf.random_normal([lOJ,stddev=.l))

Нижче в рядку 6 наведено заповнювач даних зображення. Це тензор форми [*batchSz,* 784]. Цей заповнювач *batchSz* включає 100 рядків по 784 пікселі. Рядок 7 – це відповідь – вектор завдовжки 10, заповнений нулями, крім обраного числа від нуля до дев'яти – відповідний компонент вектора дорівнює одиниці. Це унітарний вектор (*one-hot vector*), він вибирає лише одне значення набору компонентів. У цьому ж рядку параметри та вихідні значення програми

6 img=tf .placeholder (tf. float32, [batchSz, 784])
7 ans = tf.placeholder(tf.float32, [batchSz, 10])

Наступний рядок 8 передає (розмір партії) зображення лінійні блоки (як визначено *W* і *Ь*), а потім застосовує *softmax*, щоб отримати вектор ймовірностей.

8 prbs=tf.nn.softmax(tf.matmul(img, W) + Ь)

Тут матриці matmul вхідних зображень множаться: [100, 784) на W [784, 10], щоб отримати матрицю [100, 10], до якої слід додати зсуви, що завершуються матрицею форми [100, 10] - 10 логітів для 100 зображень партії. Діючи функцією *softmax,* отримуємо матрицю [100, 10] ймовірностей міток для зображень цифр.

*9 xEnt = tf.reduce_mean(-tf.reduce sum(ans * tf.log(prbs),*
reduction_indices=[l]))

У цьому 9 рядку обчислюється середня крос-ентропійна втрата за 100 прикладів, які обробляються паралельно (тут функція *tf. log(x)* повертає тензор, де кожний елемент х замінюється його натуральним логарифмом,

а функція ans * ts. множення двох тензорів). Частина рядка *tf.reduce sum(A,reduction_indices=[1])* підсумовує рядки *A*. Критично важлива частина тут *reduction_indices=[1].* Функція *reduce sum* може підсумовувати стовпці за допомогою *reduction_indices= [1*]. В результаті виходить масив [100,1] з логарифмом правильної ймовірності як єдиний запис у кожному рядку. Наприкінці обчислення крос-ентропії функція reduce mean підсумовує всі стовпці та повертає середнє (1,1 або близьке до цього).

10 train = tf.train.GradientDescentOptimizer(0.5) .minimize(xEnt)

Цей рядок включає зворотний прохід – змінює ваги з використанням градієнтного спуску при швидкості навчання 0,5. Тут компілятор TF обчислює похідні та інше (він містить безліч вбудованих функцій, якими він користується). До речі, обернений прохід обчислюється, якщо прямий прохід використовував лише функції TF.

11 numCorrect= tf.equal(tf.argmax(prbs,l), tf.argmax(ans,l))
12 accuracy = tf.reduce_mean(tf.cast(numCorrect, tf.float32))

Рядки 11 та 12 обчислюють точність моделі *(accuracy*), тобто підраховують кількість правильних відповідей і поділяють кількість оброблених зображень. Тут функція *at-gmax*, як і arg maxxf (x), повертає значення x, яке максимізує *ƒ (x).* Причому *tf. argmax (prbs, 1)* отримує два аргументи. Перший – це тензор, яким визначаємо argmax. Друга – це вісь (*axis*) тензора для використання функції *argmax.* Це працює як іменований аргумент, який раніше використовували для reduce surn, – він дозволяє підсумовувати різні осі тензора. Наприклад, якщо тензор дорівнює ((0,2,4), (4,0,3)), використовуємо вісь 0 (стандартно), повертаємо (1,0,0). Тобто спочатку порівняємо 0 із 4 і повернемо 1, тому що 4 було більше. Потім порівняємо 2 з 0 і повернемо 0. Якби вісь 1 використовувалася, повернули (2,0). У рядку 15 є масив розміру партії логітів. Функція *argmax* повертає масив розміру партії логітів. Потім ми використовуємо функцію tf. equal, щоб порівняти максимальні логіти з правильною відповіддю. Він повертає вектор розміру партії логічних значень (True, якщо вони є рівними), який перетворює *tf. cast (tensor, tf. float32)* у числа з плаваючою комою, так що *tf. reduce mean* може скласти їх та отримати правильний відсоток.

13 sess = tf.Session()
14 sess.run(tf.global_variaьles_initializer())

Далі, як тільки визначено сеанс (рядок 13) та ініціалізовані значення параметрів (рядок 14), можна навчати модель. Для збереження інтриги, залишимо це для частини 3 цієї книги.

Автокодувальники. Складні мережі часто формуються з набору менших мереж, або блоків, або спеціалізованих шарів. Наприклад, базовими є багатошарові мережі прямого поширення (з одним зміщенням[48] зі зворотним поширенням помилки), але вони перемежовуються такими конструкціями, як ***обмежені машини Больцмана***[49] (ОМБ – RBM) і автокодувальниками.

Робота автокодувальника полягає в тому, що вхід формує уявлення-прикмета-вектор у прихованому шарі, а цей прихований шар повертає на вихід той самий результат, який він отримав на вході. Якщо прихований шар відрізняється (наприклад, числом нейронів) від вхідного та вихідного, подання на ньому може мати меншу або більшу розмірність (останні називаються overcomplete autoencoders, а реально знижують розмірність undercomplete). Важливо, що в першому випадку можна очікувати, що (1) з вхідної інформації будуть виділені важливі дані і вони не будуть громіздкими, як зовнішні, (2) уявлення-вектор у прихованому шарі може мати потрібну розмірність для використання в наступних шарах. Але виявилося, що автокодувальники також здатні зменшити рівні шуму, відновити втрачені дані, виділяючи необхідний сигнал, вони здатні краще архівувати дані, які потім вони ж розшифрують, і т.д. Тут також використовується дропауд[50] посилення процедури виділення потрібних ознак. Отримані ознаки – уявлення вектори дозволяють краще навчати мережу прямого поширення з наявністю автокодувальників.

ОМБ, окремий випадок автокодувальника[51], – теж мережа прямого поширення (з двома зсувами). Її можна навчати без вчителя[52], що дозволяє виділити ознаки вхідного вектора (а також знижувати розмірність вихідного вектора), що надходить на вхід ОМБ і краще оптимізує наступні шари персептрону.

---

[48] Нейрони зміщення (bias нейрони) потрібні тільки для зсуву функції активації, між собою вони не пов'язані. Найчастіше нейрони зміщення на схемах не показані. Наявність нейронів зміщення – один з гіперпараметрів.

[49] Обмеженість тут в тому, що зв'язки між блоками, нейронами кожного шару відсутні.

[50] Дропаут (вибивання – від англ. dropout) – форма регуляризації, для зменшення перенавчання. Тут можливо виключення третини нейронів у шарах, а потім їх підключення для посилення вагі, що залишилися, що може збільшити стійкість роботи.

[51] За словами Д. Хінтона, «після того як ми з Террі Сейновскі придумали алгоритм навчання машини Больцмана, мені знадобилося 17 років, щоб знайти такий його варіант, який працював ефективно».

[52] Тут можливо пошарове навчання: на перший видимий шар ОМБ подається вектор даних, а другий – прихований шар ОМБ виділяє необхідні ознаки. Взагалі кажучи, пошарове навчання поступово втрачало значення для сучасних мереж, де почали застосовувати якісно нові методи оптимізації (тобто навчання), де ретельно підбирали функції активації і знайшли ефективні способи ініціалізації ваг.

Ваги зв'язків з прихованими блоками налаштовують так, щоб після отримання вхідного вектора і його реконструкції сигнал мережі нагадував вхідний[53]. Навчання ОМБ передбачає, що вона здатна по обмеженій вибірці реконструювати повний набір вхідних даних. Кілька таких подвійних шарів ОМБ дає кращі результати з виділення ознак. Потім цей сформований сигнал відправляється на вхід мережі прямого поширення.

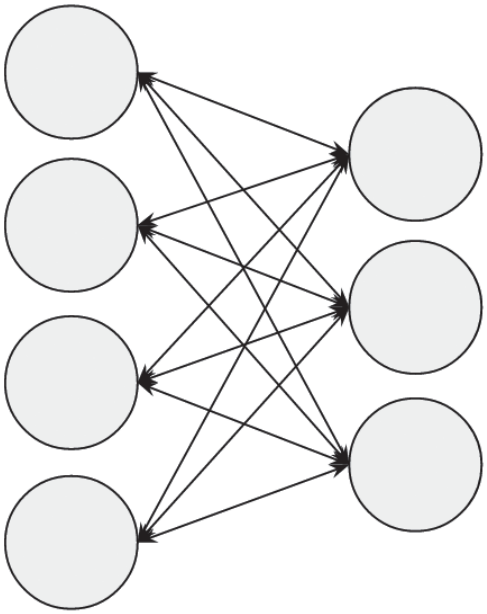


Рис. 19. Мережа ОМБ складається з видимого шару і прихованого шару з графом ваг [4]

Кілька більш загальних класів мереж подібних ОМБ є більш складні автокодувальники, вхідний і вихідний шари яких однієї розмірності, а внутрішні шари можуть мати помітно меншу розмірність. На шарах меншої розмірності можна знімати вектори меншої розмірності (знижують розмірність сигналу, наприклад, для архівації), крім того, автокодувальники, виявляється, допомагають знижувати рівень шуму, очищаючи сигнал. Навчання відбувається на нерозмічених (непомічених) даних, на відміну від ОМБ, яка навчається на вхідних помічених навчальних даних. Зміщення допомагають обчислювати помилку реконструкції вхідних даних.

Крім автокодувальників до мереж, попередньо навчених без вчителя (Unsuperviserd Pretrained Networks – UPN), відносяться глибокі мережі довіри (ГМД) і породжуючі змагальні мережі.

Мережі довіри складаються з ОМБ і мережі прямого поширення, потрібної для точного налаштування (за допомогою зворотного поширення помилки).

[53] Етап реконструкції за словами Д. Хінтона, нагадує «дані, які машини бачать уві сні».

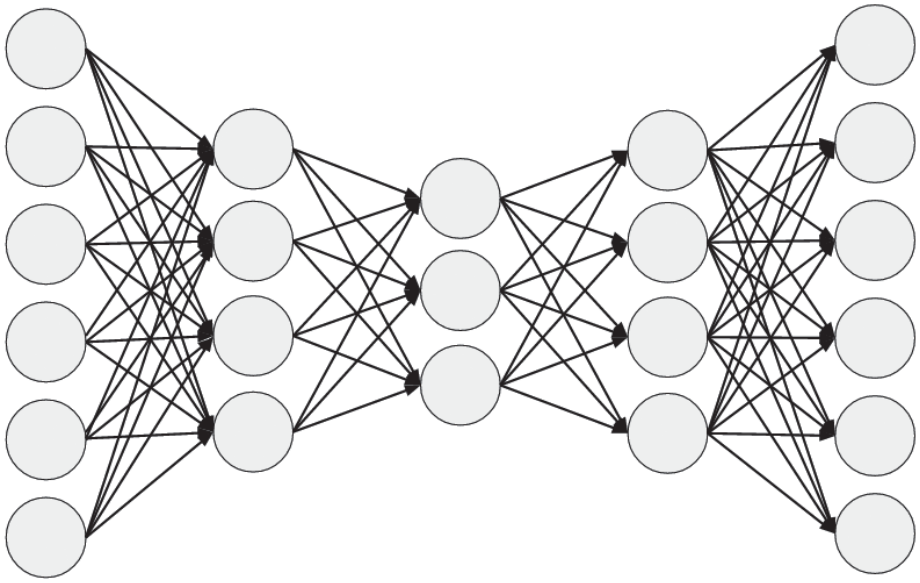


Рис. 20. Мережа автокодувальника [4]

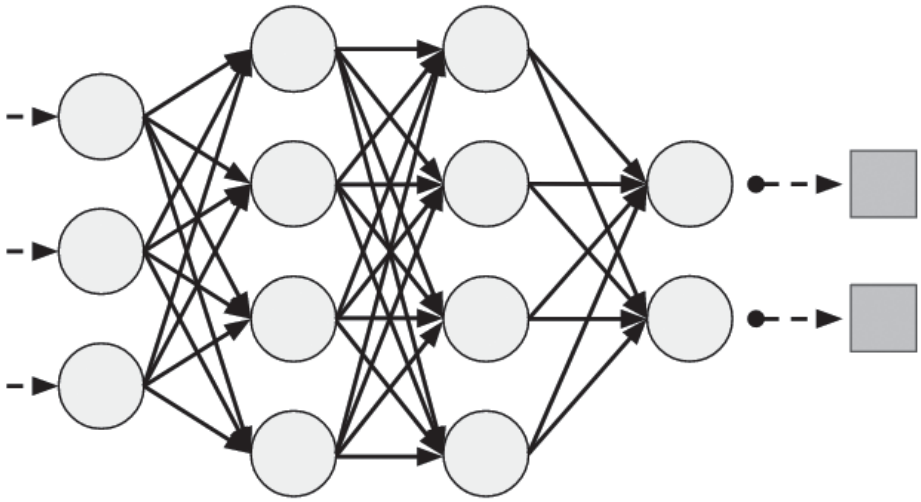


Рис. 21. Архітектура мережі ГМД.
Справа вказана мережа прямого поширення [4]

Виділені ознаки шарами ОМБ після попереднього навчання без учителя подаються на вхід підмережі прямого поширення в складі ГСД (DBN), де вже здійснюється точне налаштування за допомогою зворотного поширення помилки. В даний час ГМД, яким відданий безумовний пріоритет в історії відродження глибокого навчання, все більше витісняються згортковими нейронними мережами.

Породжуючі змагальні мережі (ПЗМ) на основі навчання на ряді зображень здатні самостійно синтезувати нові зображення, мелодії, відео, навіть будувати зображення на основі текстового їх опису[54].

[54] Породжуюча мережа перетворює дані, використовуючи і так званий антизгортковий (decjnvolutional) шар.

Маючи великий набір навчальних даних (в цій мережі їх багато більше числа її параметрів), завжди можна побудувати породжуючу нейронну мережу, яка відразу представляє зображення, які б сприймалися як вибірки з набору навчальних даних. Цій мережі не було б необхідності займатися класифікацією. Але сумніви залишаються, тому цю підмережу об'єднують з дискримінаційною підмережею, що перевіряє зображення. Тому ці мережі називають змагальними.

Так звана дискримінаційна модель (підмережа) в складі цієї інтегрованої ПЗМ ці зображення класифікує (наприклад, чи є вони справжніми або синтетичними).

У процесі навчання набір прикладів дуже великий (наприклад, для даних ImageNet прикладів 100 млн. Обсяг їх 200 ГБ і перевищує кількість параметрів в 100 МБ). Дискримінаційною підмережею часто є згорткова мережа CHC (див. нижче), яка бере зображення і надає їх класифікацію. Градієнти виходів дискримінаційної підмережі по її вхідних даних дозволяють поправити синтетичні дані.

Згорткові нейронні шари і мережі (convolutional neural network, CNN) і рекурентні (і їх аналог – рекурсивні) нейронні шари і мережі (recurrent neural network, RNN) – це також нові мережеві архітектури.

Згорткові мережі[55] для роботи з зображеннями з'явилися в результаті зусиль співробітників AT & Bell Labs на рубежі століть. Успіхи в цьому напрямку були пов'язані з розробкою нейронних мереж з тимчасовою затримкою (Time-delay neural network – TDNN), яка являла собою одновимірну згорткову мережу для часових рядів. При цьому Д. Хінтон з колегами показав (1988), як навчати такі мережі за допомогою зворотного поширення. До речі, в цей же час був створений тестовий набір MNIST рукописних цифр (цей набір даних, як і ImageNet, CIFAR, дозволили далі розвивати технології оптимізації).

CNN архітектура. Основою цих структур є наявність фільтрів – ядер. Сенс цих фільтрів виділяти необхідні параметри зображення або деяких масивів даних. Нейрони розташовуються в шарі (в якому багато підшарів) в трьох просторових вимірах.

Нейрони кожного підшару пов'язані з невеликим числом нейронів сегменту попереднього шара.

При цьому вхідна матриця даних обробляється ядром. Ця процедура є в процедурному, математичному сенсі згорткою двох матриць. При цьому матриці можуть бути різної розмірності.

Ядро (фільтр згорткового шару – матриця, яка має менший розмір, ніж у вхідного сигналу) згортається з вхідними даними, породжуючи одне

[55]Див. https://programforyou.ru/poleznoe/convolutional-network-from-scratch-part-zero-introduction

значення в карті ознак[56]. Карти активації складаються в стопку (тому потрібен третій вимір). Нейрони, пов'язані з однією ділянкою вхідних даних, називають «стовпчиком» (depth column). «Крок» зрушує фільтр і формує інший стовпчик.

Отримані значення застосування фільтрів нормуються (пакетне нормування), це спрощує аналіз. Між згортковими шарами вставляють пулінгові, розмірність яких менше, вважається, що це запобігає перенавчанню.

Операція згортки. Для матриць розмірності n_A для $\widehat{A}=[a_{ij}]$ і n_B для $\widehat{B}=[b_{ij}]$ при $n_A=n_B$ отримаємо $\widehat{A}*\widehat{B}=\sum_{i=1}^{n}\sum_{j=1}^{n}a_{ij}b_{ij}=\sum_{i'}^{n^2}a_{i'}b_{i'}$, причому в першій подвійний сумі рядки і стовпці визначені окремо, а в другому одинарному підсумовуванні $i'\subset 1,...n^2$ – підсумовування проводиться по всій площі матриць.

Якщо одна квадратна матриця більша за іншу, то процедура згортки наступна. Вибираємо в більшій квадратній матриці квадратну область, рівну менший квадратній матриці, і згортаємо їх. Послідовно зрушуючи квадратну область розміру меншої матриці, отримуємо після аналогічних згорток результуючу матрицю. Якщо зрушення вправо і вниз квадратної області дорівнюють одному стовпчику і одному рядку, то параметр зсуву s дорівнює одиниці. Розмірність результату, тобто нової матриці, дорівнює $n_C=[\frac{n_A-n_B}{s}+1]$. Тут квадратні дужки визначають цілу частину їх вмісту (операція floor).

До речі, людина добре бачить, і у неї висока роздільна здатність зображення завбільшки з ніготь на відстані витягнутої руки (звідси можна визначити тілесний кут з високою роздільною здатністю цієї «центральної ямки»). Мимовільні рухи очей (саккад) дозволяють переглянути інші ділянки зображення, які потім «зшиваються» вже в результаті їх інтегрування при роботі мозку.

Операція зведення (pooling). На матрицю входу, тобто більшу матрицю, накладається вікно завбільшки з матрицю ядра і знаходиться максимальне значення з елементів в цьому вікні. Але перетину колишнього вікна з новим положенням вікна не допускається. Результат – виходить нова матриця максимальних значень з елементів кожного вікна.

---

[56] Нейрофізіологи Девід Хубель і Торстен Візель з'ясували (Нобелівська премія), що окремі нейрони первинних зорових центрів (первинної зорової кори V1) сильніше реагують на специфічні зорові образи в певній ділянці зображення. Реакція простих клітин цієї кори стала основою для створення фільтрів згорткової мережі, а складні клітини кори V1, що інтегрують і згладжують відхилення елементів зображення, стали прообразом пулінгових блоків.

Операція заповнення (padding). Якщо в результаті операції згортки виходить матриця, розмірність якої відрізняється від розмірності вихідного зображення, то матриця згортки доповнюється рядками і стовпцями так, щоб розмірності збіглися.

Згортковий і зводящий шари. Тут кількість ядер (channels) n_C. Активаційні функції повинні бути нелінійні. Вхідний сигнал діє на кожен фільтр незалежно, результати складають в стек. Матриці результатів $n_B \times n_B$ складаються в матрицю $n_B \times n_B \times n_C$. Оскільки ядра мають розмірність $n_K \times n_K$, то параметрів (ваг) в шарі $n_K^2 \cdot n_C$ і це не залежить від розміру вхідного сигналу (зображення).

Такий згортковий шар позначають Conv з номером, де вхідне зображення $n_A \times n_A$ після згортки з ядрами переходить в тензор з розмірністю $n_A \times n_A \times n_C$. Тут також слід враховувати 3 колірні канали RGB (червоний, зелений, синій). Тобто розмірність вхідного зображення може дорівнювати $n_A \times n_A \times 3$. Згортковий шар вимагає визначити гіперпараметри (тут це розмір ядра, зрушення, заповнення).

Зводящий шар позначають як Pool з номером. Цей шар не має навчених параметрів, але використовує гіперпараметри (n_K, s), а зведення використовується для скорочення розмірності тензорів. Цей шар завжди слідує за згортковим шаром.

Фахівцям не зовсім зрозуміло досі, чому згорткові мережі виявилися настільки ефективними. Порівняння зображень, які мережа запам'ятала при навчанні, та отриманих на її вході відбувається за ознаками, яких чим більше, тим краще. Саме згорткові мережі дозволяють виділити більш ніж значне число ознак по кожному зображенню. Не дивно, що при збігу значної кількості ознак цілком можливо ефективно виявити ідентичність зображень.

Вважають також, що допоміг поділ обчислювальних процедур при застосуванні ядер, що розбило процедуру обчислення, характерну для повнозв'язних мереж, на множину більш простих процедур. Крім того, таке розбиття створило нові можливості для оптимізації.

Згорткові нейронні мережі в задачах навчання з підкріпленням. Згорткові шари успішно застосовуються в задачах навчання агентів, що входять, даними для яких виступають зображення без попередньої обробки або підготовки, наприклад відеопотік від камери на автомобілях, оснащених автопілотом. У процесі навчання агент на базі згорткової нейронної мережі повинен класифікувати кожен типовий блок зображень як якийсь набір типових станів, для кожного з яких має знайтися оптимальна дія. Таким чином, моделюючи безліч разів різні комбінації станів і реакцій середовища на дії агента, можна через деякий час

отримати алгоритм, який працює правильно для кожної ситуації, яка можлива в межах моделі.

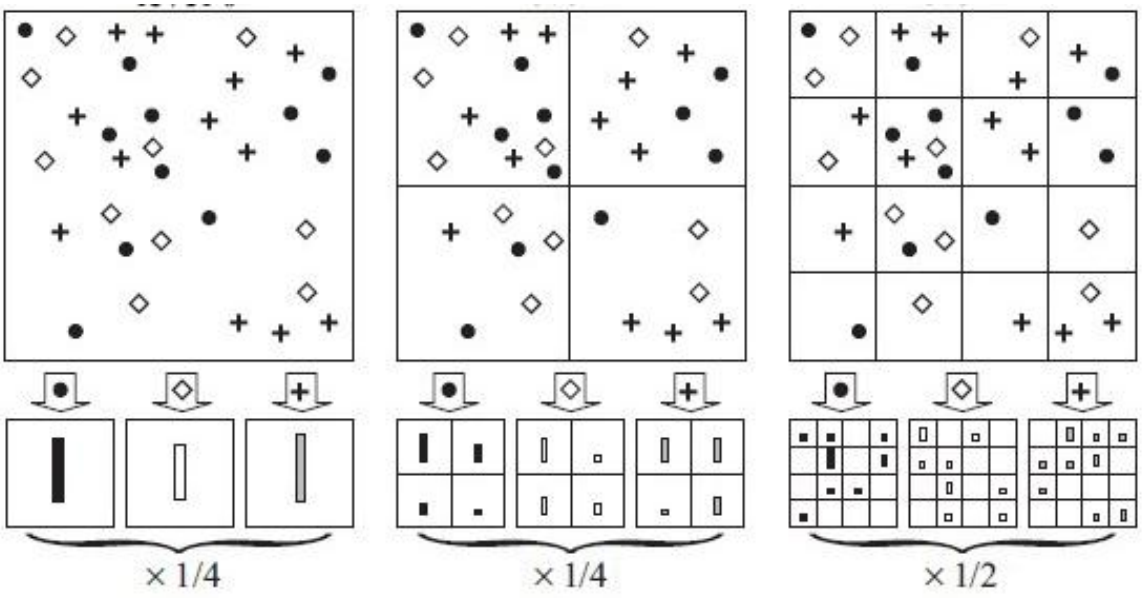


Рис. 22. Послідовна (пошарова) обробка зображення мережею CNN

Рекурентні мережі РНМ (RNN) – архітектура. Ці шари і мережі призначені для обробки інформації, для якої порядок дотримання важливий. Активне використання подібних архітектур схильні приписувати С. Хохрайтеру і його колегам на початку 90-х років минулого століття, які навчилися моделювати і обробляти послідовні дані.

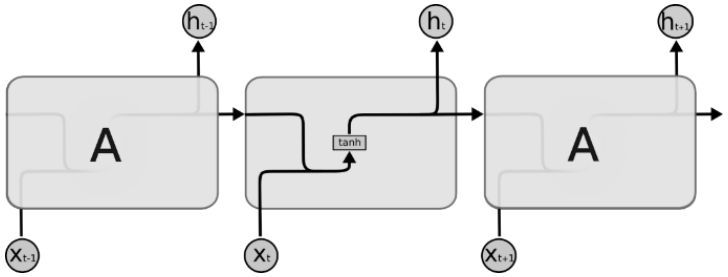


Рис. 23. Модуль рекурентної мережі з одним шаром, що включає функцію активації

Пізніші значення даних в цих моделях залежать від більш ранніх. Породжуються послідовності, в яких вихід на кроці залежить від поточного входу і входів на попередніх кроках. Градієнти обчислюються за допомогою ***зворотного поширення в часі*** (backporpagation through time – BPTT).

Перевага конекційних РНМ в тому, що міститься інформація з як завгодно довгого контекстного вікна (в колишніх марковських моделях існує на цей розмір обмеження). Рекурентна нейронна мережа являє собою ланцюжок повторюваних модулів[57].

Саме рекурентні зв'язки створюють цикли. На кожному часовому кроці обчислюється вихід h_t, який залежить від входів x_t, x_{t-1} x_{t-2}, … x_{t-m} на попередніх часових кроках. Вихід одного шару з'єднується зі входом подальшого як у мережах прямого поширення.

Ці рекурентні шари[58] застосовують одну і ту ж операцію до всіх членів даної послідовності і запам'ятовують попередні значення цієї послідовності для подання таких значень цієї послідовності[59]. Кожен елемент обробляється окремо і послідовно.

Входи x_t, внутрішня пам'ять s_t, один набір ваг (для станів внутрішньої пам'яті) W, інший (для елементів послідовності) – U. Перший нейрон, отримавши x_t, зробив обчислення в момент t, справив на виході h_t, створив стан внутрішньої пам'яті s_t. Другий обчислює в період $t+1$, отримує на вході x_{t+1} і попередній стан пам'яті s_t, генерує h_{t+1} і стан пам'яті s_{t+1} і т. д. Новий стан внутрішньої пам'яті можна записати як $s_t = f(Ux_t + W \cdot s_{t-1})$, де f – активаційна функція. Рекурентність мережі[60] в тому, що стан пам'яті в даний момент визначається через стан пам'яті в момент попередній.

Навчання рекурентної мережі. На вхід подається по одному речовому вектору послідовності. Це послідовність активації вхідних блоків. Решта обчислює поточний стан активації на кожному кроці. Помилка кожного вектора вхідної послідовності – сума відхилень активацій від міток (відомих даних навчання). Використовується алгоритм зворотного розповсюдження часу. Він подібний до алгоритму зворотного

---

[57] Див. розробку «The unreasonable effectiveness of recurrent neural networks». https://habr.com/ru/company/wunderfund/blog/331310/

[58] Такі шари створюють на базі LSTM мереж, використовується осередок LSTM пам'яті, вентильні рекурентні блоки (Gated Recurrent Units – GPU). У варіанті вентильного рекурентного блоку (Gated Recurrent Unit – GRU) теж є вентилі забування і поновлення, але пам'ять повністю відкрита.

[59] Ці шари здатні: 1). визначати ймовірності слів з урахуванням попереднього тексту; 2) забезпечувати машинний переклад; 3). перетворювати мову в текст; 4). створювати мітки, анотації на зображеннях; 5). з урахуванням слів на вході (питання, обговорення) генерувати з них відповіді.

[60] Рекурентний (recurrent – англ., франц.) – повторюється; рекурсивний (recursive – англ., recursif – франц.) – подібний (recursion – лат. круговорот, повернення).

поширення, тут градієнти (помилки) від майбутнього до минулого. На практиці застосовують, як правило, усічений алгоритм, розбиваючи весь інтервал на невеликі ділянки. Методика навчання називається зворотним поширенням у часі k (back propagation through time). Кількість ітерацій назад у часі цих мереж називають розміром вікна (window size), зазвичай це близько 20.

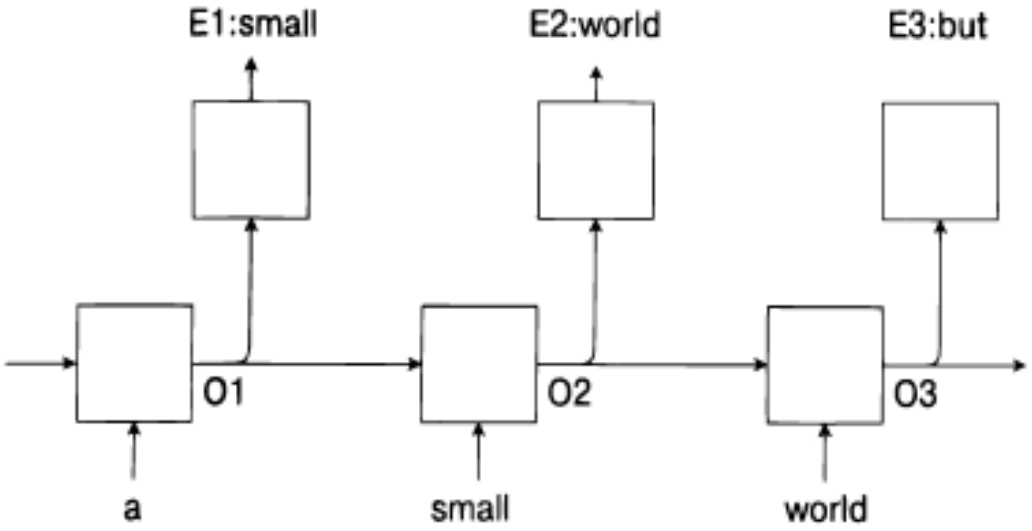


Рис. 24. Робота рекурентної мережі

Нехай розмір вікна 3. Розглянемо процес, дотримуючись роботи [4]. На входи надходять слова "a small world", і потрібно спрогнозувати "small world but". Зліва, згідно з рис. 24, надходить (00), з'єднується з векторним поданням "a" і передається через мережу RNN, де стає 01, передаючи втрати на E1. Однак, крім переходу до El, 01 поєднується з другим словом "small". Обчислюємо помилку. Тут ми обчислюємо вплив як W, і b на помилку у прогнозуванні слова "small". Але W і b викликають помилку двома різними способами – через помилку, яка веде від них до E2, а також через те, як вони подіяли на 01. При обчисленні E3, W і b впливають на помилку трьома способами: безпосередньо з O3, O1 і O2. Таким чином, параметри в цих змінних змінюються шість разів (програма зберігає проміжний підсумок і змінює їх один раз).

Найпоширеніший варіант РНС – мережі LSTM[61], в яких осередок пам'яті містить вентиль забування, вхідний і вихідний вентилі.

[61] Google Translate використовує з 2017 р. 8-шарові LSTM як кодера і декодера сенсу пропозицій.

Як покращити пам'ять RNN про минулі події, навчивши її запам'ятовувати потрібні дані та забувати непотрібні? Розглянемо спочатку якісно цей процес.

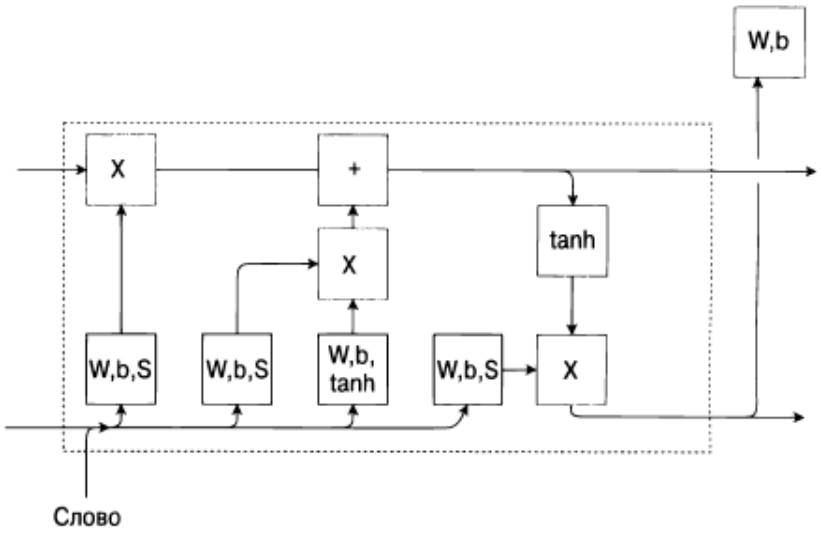


Рис. 25. Структура LSTM [4]

Верхня лінія пам'яті з (t-1) – стан попередньої комірки (cell state). Нижня лінія пам'яті називається h(t-1). Пам'ять спочатку видаляється (скоріше слабшає) у блоці часу X зліва вгорі (time unit) і додається у блоці "плюс (+)" (plus unit). Детальніше, додавання відбувається в такий спосіб. Внизу зліва формується нове значення h', де h множиться на вектор слова e, що знизу. Потім формується функція забування f як комбінація значення сигмоїдної функції (h'e). Ця функція забування множиться на c(t-1) із верхньої лінії пам'яті. Виправлена таким чином верхня пам'ять c'(t-1) поелементно множиться зі скалярним добутком застосування сигмоїдної функції до h'(t-1) (враховує вектор слова) та функції tanh до h(t-1) (враховує попередні значення локальної пам'яті), остаточно формуючи стан наступного осередку c(t). До речі, функція tanh відрізнятиметься від сигмоїдної функції тим, що може виводити як позитивні, так і негативні значення, тому вона змінює не лише масштаб, а й приносить щось нове. Крім того, слід мати на увазі, що добуток цих функцій, нормованих на одиницю, дає завжди меншу абсолютну величину, тобто визначає рівень «забуття». Тобто вся ця конструкція дозволяє, трохи послаблюючи пам'ять від минулих етапів застосування блоків, вносити поправки і формувати новий стан пам'яті.

Закриті вентилі зберігають інформацію. У цих мережах використовують навчання з учителем. Зазвичай РНС обчислює градієнт за допомогою алгоритму зворотного поширення в часі (backpropagation through time – BPTT).

Тепер можна розглянути детальніше роботу LSTM. Важливим для LSTM є стан осередку (cell state) – горизонтальна лінія, що проходить зверху схеми. Стан осередка-комплексу шарів подібний конвеєру. Інформація легко просувається, практично не змінюючись. Однак існуючі фільтри (їх три) можуть інформацію прибирати.

Відзначимо, що мережі Long short–term memory (LSTM) – здатні до навчання довготривалим залежностям. Нагадаємо, що вони були представлені З. Хохрайтер і Ю. Шмідхубер (1997).

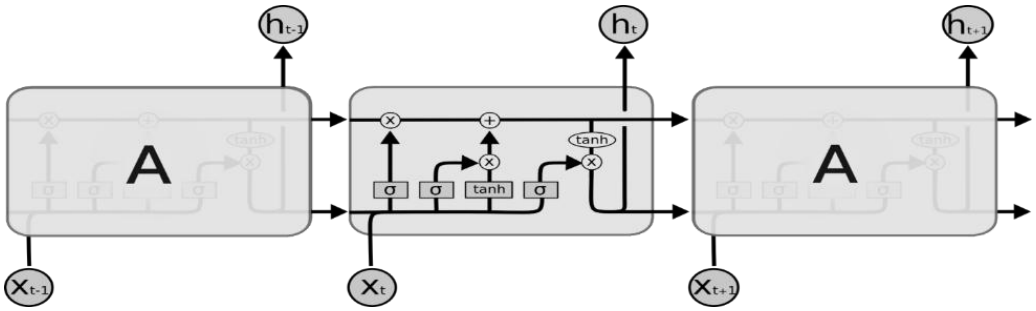


Рис. 26. Кожен нейрон одного осередку має якісь фільтр: вхідний (input gate), вихідний (output gate) і забування (forget gate)

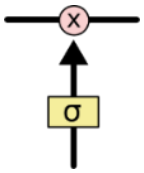
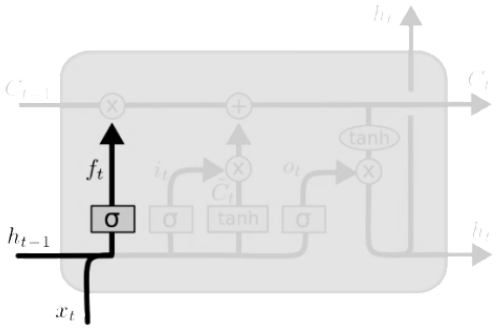


Рис. 27. Представлення фільтру

Розглянемо докладніше роботу блоку. Спочатку треба вияснити, яку інформацію з попереднього осередку треба відкинути. Сигмоїдний оператор-шар (forget gate layer) отримує h_{t-1}, а також x_t та повертає значення від 0 до 1 для кожного символу – вектора стану осередку C_{t-1}.



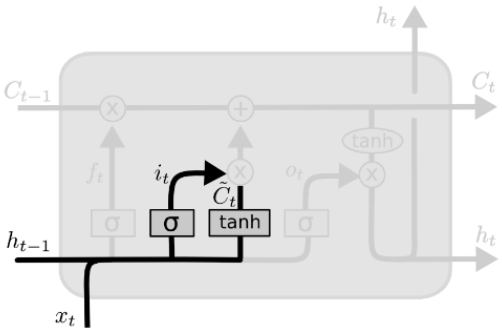
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Рис. 28. Фільтр забування (forget gate layer)

Дані попереднього шару (зокрема пам'ять) з сигналом надходять на вхід нейрона шару, і він повертає число від 0 до 1. Коли нуль, фільтр інформацію не пропускає.

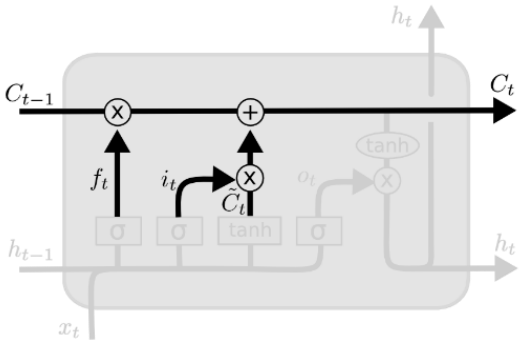
Потім ініціативу перехоплює «шар вхідного фільтра» (input layer gate), який визначає, які значення слід оновити.

Потім tanh–шар будує вектори нових значень $\tilde{C}_t$, що представлено першим і другим рядками символів на рис. 29.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

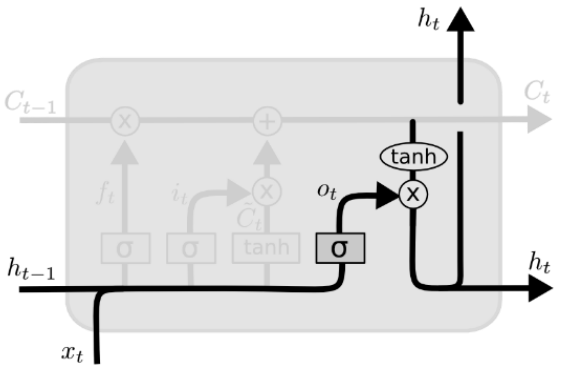
Рис. 29. Шар вхідного фільтра (input layer gate) і tanh-шар. Далі дані попередньої комірки – осередку C_{t-1} змінюються на новий стан наступної комірки – осередку C_t



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Рис. 30. Формування реакції осередку C_t

Тут слід уточнити застосування фільтрів. Цей сигмоїдальний шар визначає, що буде виведено в наступну комірку. Шар tanh-шар потрібен, щоб отримати на виході значення з діапазону від –1 до 1. Потім це значення перемножується з вихідними значеннями сигмоїдального шару, що дозволяє виводити тільки потрібну інформацію.



$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Рис. 31 Робота фільтрів при поданні інформації

В даний час активно використовують керовані рекурентні нейрони (Gated

recurrent units, GRU). Фільтри «забування» і входу об'єднані в один фільтр «оновлення» (update gate). Стан осередку інтегрують з прихованим станом і т. п. Ця модель виявилася навіть простішою, ніж стандартна LSTM

Як працює мережа. Проілюструємо роботу рекурентної мережі на мовній моделі, яка знадобиться нам нижче. Мовна модель – це розподіл ймовірностей наступного слова в реченні, якщо задано його початок. Рекурентна нейронна мережа «читає» пропозицію зліва направо. На кожному кроці вона формує вектор («приховане уявлення»), в якому закодована інформація про вже пройдений відрізок тексту. На основі цього прихованого уявлення робиться прогноз наступного слова. На наступному кроці модель приймає на вхід поточне слово і попереднє все приховане уявлення, обчислює наступне приховане уявлення і знаходить подальше слово[62].

Найбільш успішною архітектурою «осередку» мережі – функції, що обчислює наступне приховане подання на кожному кроці – ось тут і виявилися корисні LSTM (long short-term memory) і GRU (gated recurrent unit). Обидві ці функції реалізують перетворення вхідних векторів, що дозволяють не втрачати інформацію, яка з'явилася в тексті кілька десятків кроків назад, що забезпечує облік контексту пропозиції[63].

Навчання рекурентної мережі. На вхід подається по одному матеріальному вектору послідовності. Це послідовності активації вхідних блоків. Решта обчислюють поточний стан активації на кожному кроці. Помилка для кожного вектора вхідної послідовності – сума відхилень активацій від міток (відомих даних навчання). Використовується алгоритм зворотного поширення в часі. Він подібний до алгоритму зворотного поширення, тут градієнти (помилки) від майбутнього до минулого. На практиці застосовують же, як правило, усічений алгоритм, розбиваючи весь інтервал на невеликі ділянки.

Рекурсивні нейронні мережі. У рекурентних мережах в 90-х роках минулого століття було виявлено явище зникнення або навпаки вибухового зростання градієнтів при навчанні особливо стохастичним градієнтним спуском. Послабити ці явища вдалося, наприклад, при заміні ланцюгової структури на структури типу дерев. Крім того, рекурсивна

---

[62] Для формування акцентів, важливості слів запропонований механізм «уваги». Формується векторне представлення всього речення як сума векторів окремих слів (з прихованих уявлень рекурентної мережі на кожному кроці), при цьому множачи кожен вектор на його вагу, який також виявляється в процесі навчання нейронної мережі.

[63] Мовна модель може навчатися на будь-якій безлічі текстів без спеціальної розмітки. Для багатьох інших завдань потрібні розмічені навчальні дані, які, як можна побачити нижче, досить складно створювати.

мережа може виділяти ієрархічні структури в даних, що важливо в багатьох додатках.

Тут послідовність операцій являє собою дерево, а не ланцюгову структуру. Вигляд дерева залежить від завдань. Прямий прохід до вершини дерева – вгору. Входи – внизу.

Навчання рекурсивної мережі. Використовувалися алгоритми зворотного поширення крізь структуру (backpropagation through structure – BPNS). Найчастіше зустрічаються рекурсивні автокодувальники (здатні відтворювати вхід), з частковим залученням вчителя. Рекурсивна тензорна мережа, яка навчається з учителем, визначає цілі в кожному вузлу дерева. Тут градієнт в зворотному поширенні замінюють тензором, тобто це більш складна функція. Також були запропоновані методи навчання тільки вихідних ваг зворотних блоків (відлуння мережі і машини нестійких станів, тобто мережі так званих резервуарних обчислень – вони запам'ятовують історії входів).

Нейромережеві стимулятори [5, 6]. Програмний пакет NEST використовують для моделювання біологічно реалістичних мереж: 100 тис. нейронів і мільярд синапсів, архітектура large-scale modeling у формі дерев. Використовують моделі синапсів з механізмами синаптичної пластичності. Програмний пакет NEURON дозволяє моделювати окремі нейрони і мережі.

Критерії оцінки мереж[64]**.** Оцінки формуються на основі матриці неточностей (confusion matrix), де передбачені (класифікатором) результати порівнюються з отриманими на виході[65] (з мітками). Чутливість (повнота) TP / (TP + FN) – це уникнення хибно негативних результатів, специфічність TN / (TN + FP) – це уникнення хибно позитивних результатів. Іноді використовують так звану правильність (TP + TN) / (TP + TN + FP + FN), точність (precision або positive prediction value) TP / (TP + FP), повнота як частота істинно позитивних результатів, F–міра (або F1) – 2TP / (2TP + FP + FN). Оцінка якості може бути такою min (Точність, Повнота).

Література до розділу 9

1. Гущин І. В., Куклін В. М., Мішин О. В., Приймак О. В.. Моделювання фізичних процесів із використанням технології CUDA. – Х. : ХНУ імені В. Н. Каразіна, 2017. – 116 с.

[64] Тут T – truth, L – false, P – positively, N – negatively.

[65] Помилка тіпу I – псевдопозитивний (прогноз позитивний, мітка негативна) результат, помилка типу II – помилково негативні результати (прогноз негативний, мітка позитивна). Істинно позитивний результат (прогноз і мітка позитивні) і істинно негативний результат (прогноз і мітка негативні).

2. Черняк Е. Введение в глубокое обучение / пер. с англ. – СПб. : Диалектика, 2020. –192 с. : ил.

3. Рассел С., Норвиг П. Искусственный интеллект. Современный подход. М.: Вильямс, 2017.

4. Паттерсон Дж., Гибсон А. П20 Глубокое обучение с точки зрения практика / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2018. – 418 с.: ил.

5. Eugene M. Izhikevich. Dynamical Systems in Neuroscience: The Geometry of Excitability and BurstingMassachusetts Institute of Technology – 2007.

6. Прокин И. С., Симонов А. Ю., Казанцев В. Б. Математическое моделирование нейродинамических систем: Электронное учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2012. – 41 с.

Розділ 10. ПРО РОЗВИТОК НЕЙРОННИХ СИСТЕМ

Для пояснення змін в динаміці розвитку нейронних мереж корисно повернутися до природного інтелекту. Мозок – це нейронна мережа, яку можна з певними умовностями представляти як багатошарову структуру з нейронів (хоча в дійсності уявлення у вигляді шарів навряд чи відповідає реальності). На що слід звернути увагу, так це на те, що ці шари (або локальні масиви нейронів, що одне й те саме) окремо, вибірково не піддаються навчанню. Навчання проводиться відразу по всьому масиву нейронів або в значній його частині. Хоча відомо, що різні ділянки мозку відповідають за вироблення різних типів рішень[66].

Окремі органи чуття підключені до різних ділянок мозку і зокрема його кори, і процес навчання локалізовано саме там. Точно так же деякі ділянки мозку пов'язані з руховим апаратом, вегетативною нервовою системою і т. п. Але якщо виключити ці мотиви зокрема, то навряд чи в корі мозку наявна чітко організована пошарова переробка інформації, швидше за все формування рішень відбувається одночасно у великих масивах нейронів. А ось той факт, що навчальна вибірка у людини надзвичайно велика, є визначальним. Оскільки навчається людина довго.

Про необхідність збільшення обсягів нейронних систем. Можливості людини оцінювати і прогнозувати істотно перевершують аналогічні можливості у вищих тварин – ссавців. Обсяг кори головного мозку у людини перевищує обсяги кори головного мозку у інших тварин. Процес формування другої сигнальної системи стався тільки в умовах такого зростання обсягу кори головного мозку – природної нейронної системи. Природа підказує нам, що потрібно різко збільшувати обсяги штучних нейронних мереж, якщо ми хочемо створити інтелектуальні системи, які можна порівняти з можливостями людини. Нейрони мозку і нейрони штучних мереж подібні. Зв'язки між ними також нагадують зв'язки в корі головного мозку.

Людський інтелект вирішує безліч важливих завдань, часто не оперуючи теоріями і не виробляючи розрахунки. Ці рішення приймаються на основі порівняння виниклої проблеми з безліччю життєвих ситуацій, які запам'ятала людина раніше. Зрозуміло, що для цього природі знадобилося помітно збільшити обсяг і функціональні можливості природної нейронної мережі людини.

---

[66] Також відомо, що при необхідності, наприклад, при травмах, мозок цілком здатний передати функції ураженої ділянки іншим, уцілілим при травмах областям кори. У нормальних умовах вибір функцій окремих локальних центрів прийняття рішень, мабуть, більше пов'язаний зі шляхами надання інформації від органів чуття. І визначається зв'язками з виконавчими структурами, наприклад, руховим апаратом організму.

Поява великих нейронних мереж з мільярдами нейронів, безпрецедентне збільшення швидкодії і неймовірне розширення обсягів пам'яті дозволило повернутися до напівзабутих мереж прямого поширення, використовувати майже забуту систему навчання з учителем на базі зворотного поширення помилки.

Тобто діяти досить прямолінійно. Адже з такими можливостями інтелектуальних обчислювальних систем можна було не зважати на багато колишніх хитрощів, які були вимушеними в умовах слабкої технічної бази. Ці нові технічні та обчислювальні можливості дозволили збільшити навчальні вибірки до багатьох тисяч прикладів. Багато експертів відзначають, якщо таких прикладів буде кілька мільйонів, то здібності до навчання великих нейронних мереж швидше за все перевершать можливості людини.

Про теорії глибокого навчання. Успіхи в області створення і навчання великих нейронних систем породили нові мотиви в розвитку наукових напрямків – теорії так званого глибокого навчання для систем в основному прямого поширення (в нейронному виконанні – персептронів).

Оскільки людина освоює світ поступово від простих речей до більш складних, не вимагаючи спочатку формального опису усвідомленого, то цей принцип вирішили застосувати і для навчання систем штучного інтелекту. Важливо, що, усвідомивши прості речі, людина використовує це знання для створення уявлень про речі більш складні. Формуючи при цьому мимоволі ієрархічну систему усвідомлення дійсності.

Цей ієрархічний, поетапний підхід навчання спочатку називався глибоким навчанням. Потім глибоке навчання почали пов'язувати з іншими принципами, зокрема з більшою свободою експериментування для нейронної мережі.

Практика створення штучного інтелекту зіткнулася з проблемою внесення знань у прилади. Логічні системи отримання рішень зажадали величезних обсягів роботи з формування баз знань і даних, що здалося людям, охочим створити інтелект, який можна порівняти з природним, тупиковим шляхом. Хоча для багатьох обчислюваних задач цей шлях знайшов своє застосування.

Труднощі освоєння машиною знань про світ (рішення так званих необчислюваних завдань) було ускладнено впливом безлічі зовнішніх факторів, впливів на характер одержуваної інформації. Це явище мінливості однотипної інформації назвали її варіативністю. Перші нейронні мережі та обчислювальне середовище їхнього розміщення, володіючи порівняно низькими технічними можливостями, не дозволяли творцям цих приладів раніше навіть підступитися до подібних завдань.

Після першої хвилі нейронних систем на основі ідей нейрофізіологів[67], коли навчали мало не кожен нейрон простих мережевих конструкцій типу одношарового персептрону і настала пора першої зими розчарування, сформувалася друга (конекціоністська[68]) хвиля активності з технологіями навчання мереж. Але завищені очікування інвесторів середини 90-х років минулого століття до кінця першого десятиліття наступного століття знову заморозили інтерес до нейронних мереж[69]. Оживив інтерес до них канадський Інститут перспективних досліджень (Canadian Institute for Advanced Research – CIFAR), профінансувавши програму нейронних обчислень і адаптивного сприйняття (Neural Computation and Adaptive Perception – NCAP). Алгоритми навчання вимагали великої обчислювальної потужності, що вже до середини першого десятиліття було досягнуто. Дж. Хінтон зі співавторами продемонстрував, як в теорії можна було б ефективно навчати мережу за допомогою стратегії пошарового попереднього навчання (Hinton et al., 2006).

Таким чином Дж. Хілтон з колегами спровокував сплеск інтересу до інтелектуальних мереж, і почався третій етап розвитку цього науково-технологічного напрямку – глибоке навчання. Спираючись традиційно на досвід вивчення світу людиною, потенційні винахідники вважали, що можна створити багатошарову мережу прямого поширення, кожен наступний шар якої, спираючись на здобуті знання попереднім шаром, виділяв би нові риси усвідомлюваного процесу[70]. Аналітична теорія і програмні форми такого навчання засновані були на представленні, що мережі, що складаються з безлічі шарів, можуть навчатися послідовно. Кожен шар спирається на дані попереднього і розширює уявлення про об'єкти опису. Тобто один шар описує просту схему – каркас об'єкта, наступний шар її доповнює і ускладнює і т. д. Фактично це і було реалізовано в згорткових мережах.

---

[67] Уже на цьому етапі нейрофізіологи, мабуть, вичерпали запас ідей, який допомагав розвивати мережеві варіанти штучного інтелекту. Вважали, що це було пов'язано з недостатнім знанням роботи мозку. Хоча їхні експерименти дозволили усвідомити, що рішення різних завдань нейронами є однотипним у межах всього мозку (Von Melchner, 2000).

[68] Цей підхід спирався на уявлення, що процес роздумів може бути модельований мережею з простих елементів зі зв'язками (нейрони і синапси; слова і семантичні ознаки, а також їх зв'язки). Взагалі кажучи, конекціонізм – це породження когнітивістики – вивчення процесу пізнання, що об'єднує різні дисципліни.

[69] На тлі успіхів ядерних і графічних методів опису.

[70] Процес такого формального послідовного усвідомлення об'єкта представлений в [1]. За цими уявленнями, перший (видимий) шар формує на основі яскравих пікселів зображення його межі. Другий (прихований) шар, спираючись на ці дані, визначає кути і контури. Третій може на цій основі визначати мікроструктуру зображення і т. д.

Важливо відразу зазначити, що ці моделі відповідали так званим мережам прямого поширення без застосування зворотних зв'язків (тому базовою моделлю став персептрон). Це уявлення було реалізовано в формалізованих моделях опису і мало слугувати моделлю реальних систем отримання рішень в багатошарових і багатоелементних системах. Такі уявлення інтуїтивно засновані були також на відомому факті кращого засвоєння нового матеріалу людиною на основі її попереднього знання (досвіду). Хоча частина цього попереднього знання була освоєна в інших умовах і в інший час, але на це вже не звертали уваги.

Комплекснозначні нейронні мережі (Complex-Valued Neural Networks, CVNN). Перспективним напрямом розвитку нейронних мереж є мережі, в яких вхідні та вихідні дані, а також вагові коефіцієнти є комплексними числами. Такі мережі можна використовувати у додатках, для яких застосування комплексних чисел є природним – цифрова обробка сигналів, обробка даних радарів та сонарів, у будь-яких системах, де продуктивним є перехід до частотного чи фазового уявлення. CVNN мають ряд переваг у порівнянні з традиційними нейромережами, що працюють у просторі дійсних чисел. З точки зору нейробіології нейрони можуть взаємодіяти синхронно, коли фази нервових імпульсів збігаються і сигнали взаємно посилюються, і асинхронно – коли імпульси надходять на входи нейрона, що їх приймає, у протифазі і гасять активність один одного. Всі ці механізми краще описуються саме в комплексному поданні. З обчислювальної точки зору операції множення у просторі комплексних чисел зводяться до модуляції амплітуди і повороту фази, що у ряді випадків ефективніше навчання нейромережі. Слід зазначити, що CVNN не можна просто звести до звичайних нейромереж, збільшивши вдвічі кількість нейронів (окремо для реальної та уявної частини кожного параметра). Математика комплексних чисел дає нові можливості створення алгоритмів навчання, наприклад, модернізації методу зворотного поширення помилки з допомогою похідних Віртингера (Wirtinger Calculus), навчання без застосування градієнтного спуску, дозволяє використовувати функції активації, позбавлені традиційних недоліків.

Ілюстрацією нових можливостей, які відкривають комплекснозначні нейромережі, є проблема XOR (що виключає АБО), яка протягом багатьох років була, з одного боку, каменем спотикання в теорії нейронів (див. критику Мінського). Однак ця проблема насправді може бути легко вирішена за допомогою одного нейрона з комплексними вагами. У роботі [2] ілюструється вплив фаз наборів комплексних чисел представлення зображень. Для цього автори цієї роботи взяли два зображення, отримані комплекснозначною нейромережею, зробили фур'є-перетворення масиву чисел і, зберігши абсолютні значення членів низки, поміняли місцями лише фази подібних членів низки у цих двох зображеннях. Таким чином,

було продемонстровано, що, по-перше, комплексні числа в структурі синапсів і сигналів цілком здатні якісно описувати зображення і, по-друге, роль фаз цих чисел для якісного опису зображень визначальна (див. рис.).

Рис. 32 a. Орігінали зображеннь [2]

Рис. 32 b. Модифіковані зображення
(амплітуди членів ряду не змінювалися,
їх фази замінені фазами з іншого зображення) [2]

Докладно з особливостями комплекснозначних нейромереж можна ознайомитися з допомогою монографій [3, 4] і з поточним станом справ у цій галузі за допомогою огляду [5].

Далі уявлення про глибоке навчання розширилося на будь-які багаторівневі системи, кожен рівень яких навчається з використанням отриманих попередніми рівнями знань. Хоча досягнення перших хвиль розвитку і результати освоєння нейронних мереж вплинули на

формальні алгоритми цього глибокого навчання[71]. Однак зараз теорія глибокого навчання використовує і інші досягнення науки: лінійну алгебру, теорію ймовірностей, теорію інформації, чисельну оптимізацію і багато іншого.

Теорії глибокого навчання використовують також нові ідеї – так зване розподілене представлення – групи нейронів навчаються своїми функціями, використовуючи інші дані, що містять потрібні для цієї категорії якості[72]. Спочатку вважалися перспективними методи навчання без вчителя. Тепер повернулися до навчання з учителем. Повернулися до напівзабутого зворотного поширення помилки для навчання, розвинули теорію моделювання нейронними мережами послідовностей[73], з'явилися мережі з довгою короткостроковою пам'яттю (long short-term memory – LSTM).

Але поки, за словами Д. Белбріджа[74], «Глибоке навчання виникло з вивчення того, як функціонують нейрони людини, але, наскільки мені відомо, штучні нейронні мережі до сих пір не мають нічого спільного з архітектурою розуму істот з плоті і крові».

Не сподіваючись на здатності мереж до узагальнення, раніше використовували спеціально підготовлені і розмічені набори даних при навчанні глибоких моделей. Кількість нейронів у мережах подвоювалася кожні два з половиною роки, відбувалося також прискорене зростання числа зв'язків між ними – синапсів. Незабаром чисельність нейронів і їх зв'язків у нейронних мережах і в природному інтелекті зрівняються.

Однак неважко побачити слабкість цього підходу – пошарового навчання в разі нейронних систем із зворотними зв'язками, тобто з елементами двох напрямлених згортовних, рекурентних і рекурсивних (тим більше, що за загальним визнанням саме такі мережі показують кращі результати і навчання, і застосування) блоків. Це впадає в очі і при аналізі природного інтелекту, де висновок формується вже в загальному масиві нейронів і частково відображений у вихідному (або призначеному, або обраному вихідним) шарі. Мабуть, тому апологети формальних теорій глибокого навчання намагаються відсторонитися від проблем опису нейронних систем, що імітують природний інтелект, захоплюючись розвитком нових підходів, навіяних успіхами навчання великих нейронних систем прямого поширення.

---

[71] Так, модель ADALINE (Widrow and Hoff, 1960) могла передбачати числа, виходячи з вхідних даних. Модифікований її варіант використовується в підходах глибокого навчання.

[72] Навчання визначенню кольору може бути виконано на основі розгляду безлічі автомобілів.

[73] Робота зі створення зв'язків послідовностей з іншими послідовностями дозволила в тілі нейронних мереж прискорити прогрес машинного перекладу.

[74] https://www.peoplepattern.com/post.html#!/ai-not-yet-but-machine-learning-is-here-and-now.

Література до розділу 10

1. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с англ. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.

2. Oppenheim, A.V., Lim, J.S. The importance of phase in signals // IEEE Proceedings 69, 529–541 (1981).

3. Aizenberg I. Complex-valued neural networks with multi-valued neurons. – Heidelberg : Springer, 2011. – Vol. 353.

4. Hirose A. Complex-valued neural networks. – Springer Science & Business Media, 2012. – Vol. 400.

5. Bassey J., Qian L., Li X. A survey of complex-valued neural networks //arXiv preprint arXiv:2101.12249. – 2021.

Розділ 11. ВВЕДЕННЯ І ВЕКТОРИЗАЦІЯ ДАНИХ

Представлення даних у векторній формі полягає в їх векторизації і нормуванні. Це відноситься зокрема до послідовних даних, зображень, текстів і мови. При цьому можна менше приділяти часу виділенню ознак[75] (залучаючи фахівців з предметної області), оскільки практика показала, що мережа це зробить не гірше, а, можливо, навіть краще самостійно. Прикладами алгоритмічної векторизації можуть бути ядерне хешування (наприклад, векторизація за один прохід), частота терма (зворотна частота документа TF–IDF – векторизація за рахунок багатьох проходів) і Word–2Vec.

Перетворення вхідного вектора. Вхідний сигнал у вигляді вектора значень може бути перетворений таким чином, щоб кожне значення x-компоненти вектора перетворювалося за допомогою логістичної функції $f(x)=1/(1+\exp\{-\theta\cdot x\})$ або її узагальнення – функції softmax, інтервал зміни яких від нуля до одиниці.

З практики чисельного рішення задач відомо, що найбільш раціональним є вибір параметрів і змінних, які змінюються в межах одиниць і не дуже малі. При цьому відносні помилки і похибки слабо впливають на точність обчислень.

Крім того, ці зміни величин від нуля до одиниці аналогічні змінам ймовірності, що призвело до використання цієї аналогії, можливо не завжди цілком коректно.

Хоча логістична функція і функція softmax можуть використовуватися в якості функції активації у вихідному шарі нейронних мереж, коли мета мережі – рішення задачі класифікації. Результати вихідного шару при цьому можна розглядати як ймовірності.

Робота з перекладом. Робота з мовою весь час вдосконалюється. Наприклад, раніше так зване мовне програмування в режимі перекладу передбачало перехід від одного слова однієї мови до іншого слова, викладеного іншою мовою.

Поступово цей підхід замінюється так званим машинним перекладом (Machine Translation – MT) [1]. Технологія навчання мережі тут наступна. Даються дві пропозиції з двох різних мов і їх взаємні переклади з однієї мови на іншу (вирівняний корпус – aligned corpus), і без вчителя очікуємо, щоб машина сама знайшла відповідності. Тобто тут вже перехід від слова до слова в різних мовах замінюється переходом від послідовності до послідовності (sequence–to–sequence lerning – seq2seq).

---

[75] У машинному навчанні ознака часто представляється вектором або стовпцем матриці, який може сприйматися як незалежна змінна. Ознака - це, як правило, інформація, перероблена в форму, зручну для сприйняття її машиною.

Тобто це метод глибокого навчання для відображення однієї послідовності символів на іншу послідовність символів.

Замість мережі LSTM використовується керований рекурентний блок (Gated Recurrent Unit – GRU). Переводимо французьку фразу "hansard revise numero 1" в англійську "edited hansard number 1". Перший прохід (кодування – encoding pass) завершується, коли останній французький токен (завжди слово "STOP") обробляється GRU. Потім стан GRU передається в другу половину процесу. Мета цього проходу – створити вектор, який «підсумовує» пропозицію. Це іноді називають векторним поданням речення (sentence embedding) за аналогією з векторним поданням слів [1].

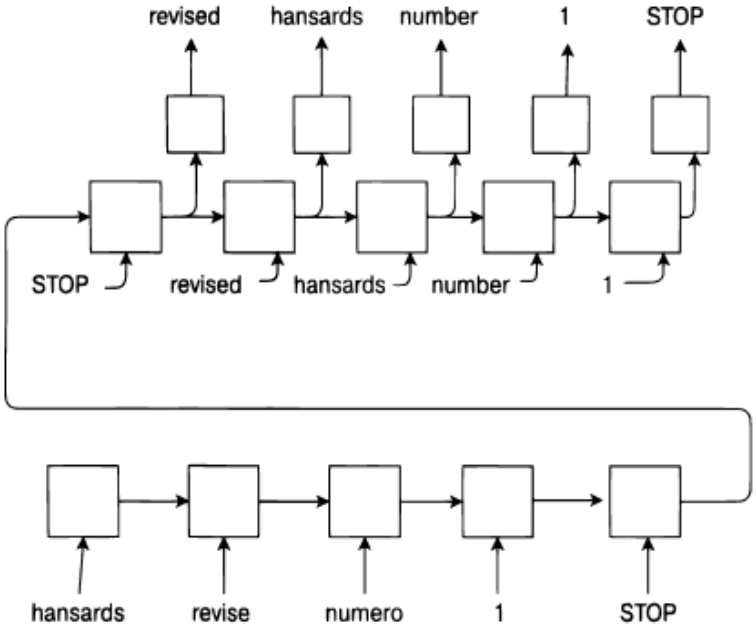


Рис. 33. Навчання від послідовності до послідовності [1]

На рис. 32 представлено схему зворотного поширення в часі, тут всі блоки RNN в нижньому верхньому рядах ряду фактично є однаковими блоками, що повторюються, але в послідовні моменти часу.

Принципи введення та обробки текстів природної мови

Обговоримо тепер методи аналізу текстів і побудови нових фраз однієї мови. Машина сприймає речові вектори даних, які зазвичай подаються на зовнішній вхідний шар і обробляються прихованими шарами нейронної мережі. У мережах, які розпізнають мову або тексти, початкові дані ще потрібно перетворити у векторну форму, зручну для роботи мережі. Текстові дані і звуки мови спочатку класифікуються за заздалегідь добре сформульо-

ваними ознаками (за рахунок лінійних класифікаторів), тобто вони повинні бути представлені в просторі ознак. Отримані таким чином (за допомогою ***унітарного кодування*** – т. зв. розріджене кодування) початкові вектори тексту, що враховують такі ознаки (в структурі вектора ознак наявність ознаки в певній позиції відзначається одиницею, а її відсутність – нулем), мають розмірність вельми значну – сотні тисяч і більше, а у відповідних їм векторів занурення, створених вже машиною (***щільне кодування***), розмірність виявляється на багато порядків менше. Тобто зазвичай вхідний шар (***шар занурення***, пошуку відповідності) таким чином виконує функцію виділення цих ознак. Але це не означає, що приховані шари в режимі глибокого навчання не можуть зайнятися класифікацією (самостійно або за попередньо отриманими правилами) даних від попередніх шарів. Якщо класифікаторів недостатньо, то використовують так звану попередню обробку документа, замінюючи слова ***лемами.***

Обговоримо нижче ***механізми класифікації текстів*** за різними системами ознак. Спочатку відбувається розбиття тексту на лексеми (лексемізація = tokenization), зазвичай це слова або групи слів. Для цього поділу користуються пробілами та розділовими знаками. Для кожного слова розроблені зазвичай вручну ***леми – словникові статті***, що пояснюють особливості застосування слова.

Ці леми об'єднані в словники, лексичні фонди – ресурси, морфологічні аналізатори, що формують ознаки слів, до яких є доступ машині. Причому слова змінюють структуру і видозмінюються (***флексії***), що також в цих базах даних враховується одночасно. Національний інститут стандартів (National Institute of Standards) США розробив набір даних ***Mnist*** (60 тисяч зображень і їх міток, набір для розробки і тестування містить 10 тисяч зображень і міток), що описує цілі числа – цифри. «Вихідні чорно-білі (дворівневі) зображення з NIST були нормалізовані за розміром і поміщені в прямокутник розміром 20 x 20 пікселів зі збереженням пропорцій.

Результуючі зображення містять рівні сірого, отримані в результаті застосування методу згладжування алгоритмом нормалізації. Зображення були центровані в поле 28 x 28 шляхом обчислення центра мас пікселів і транслювання зображення таким чином, щоб розташувати цю точку в центрі поля 28 x 28».

Існують лексичні ресурси, зазвичай створені вручну, це надійніше і допомагає машині[76]. Наприклад, ***WordNet*** – лексичний ресурс, зібраний

[76] Однак слід мати на увазі закон Ціпфа: якщо відранжувати слова за частотою їх появи в текстах, то їх частота зменшується експоненціально. Тому в перших роботах з векторизації слів Міколова (творця Word2Vec) розглядалися 50-мірні або 100-мірні вектори.

вручну, пов'язує слова з їх сінсетом – семантичним знанням – поняттям. ***FrameNet*** і **VerbNet** – лексичні ресурси, зібрані вручну для дієслів.

Існує **Paraphrase (PPDB)** – база даних парафраз. Тексти можна порівнювати на основі лічильників слів і порядку букв і слів (для англійської мови) в тексті. Тексти формують ***Bag–of–words – BOW*** – мішки слів, і відбувається їх оцінка (зважування), наприклад, програмою ***TF–IDF*** за ознаками, за частотою появи (Term Frequency), з літерними біграмами (або за n-грамами). Використовуються і ***Skip–грами*** – це n-грами з проміжками, внаслідок пропуску проміжних токенів.

Це називають тематичною класифікацією – за мішком слів і мішком словесних біграм. Для виключення неоднозначності передбачена оцінка за контекстом, в мінімальній частині це ***вікно*** – найближче оточення слова.

Це наслідок з дистрибутивної гіпотези мови – сенс слова завжди можна вивести із контексту. Розроблено ***biRNN*** – архітектуру для узагальнення віконних ознак. Також існують проекти типу ***Universal Treebank Project*** – визначення частин мови за мітками ***POS*** (Part–of–speech tag), за положенням слова в реченні. Якщо слова представлені у векторній формі вже після процедури занурення, то відстань між словами визначається з використанням однієї з відомих метрик. При навчанні з учителем, що характерно для глибоких систем, використовують оптимізацію, яка складається в процедурах регуляризації – спробах уникнути багатозначності і заснована на перевазі помилок, які змінюються раз від разу рішенням (перенавчання).

Також намагаються забезпечити мінімізацію помилок (втрат) при зворотному поширенні помилки.

Мови розмітки. Велику роль у навчанні мережі відіграють також мови розмітки – логічні і візуальні (не повні за Тьюрингом) – набір символів в тексті для передачі інформації про структуру. Вважаються класом комп'ютерних мов, але не мовами програмування: SGML, TeX, PostScript, RTF для друкування, Microsoft Word, OpenOffice для інтерфейсів, а також досить поширені розмітки
HTML, XHTML, XML, WML, VML, PGML, SVG, XBRL для Інтернету.

Перейдемо до більш конкретного опису основних операцій.

Стадії вилучення, перетворення і завантаження

Обробки (Extract – Transform – Load, ELT) даних використовуються зокрема в бізнес-аналітиці (business intelligence, BI) і Zalando SE[77],

[77] Компания Zalando підготувала MNIST-подібний набір зображень їх одягу Zalando SE, для якого вони надали 60 тис. тренувальних зображень і 10 тис. тестових зображень. Як і в наборі даних MNIST, кожне зображення мало 28x28-піксельний формат у відтінках сірого. Розробники

в системі Apache Hadoop – ELT конвеєр (об'єднання наборів даних, фільтрація, перетворення і завантаження).

Вихідні табличні дані можуть відрізнятися (номінальні – кінцеве число категорій; порядкові – розташовані в певному незмінному порядку; інтервальні – впорядковані і виміряні у фіксованих одиницях; відносні – числові, описують відхилення щодо базового значення). Створюється вектор (n–1 ознак і одна мітка, всього n).

Стандартизація передбачає формування масиву з нульовим середнім значенням і дисперсією, рівною одиниці (в первісному масиві віднімається середнє і результат ділять на дисперсію). Масиви можуть бути розрідженими (відсутність даних – нуль) або щільними. Мінімаксне масштабування застосовують до зображень (формат RGB має діапазон [0, 256], а приводиться до діапазону [0,0, 1.0]). Процедура відбілювання полягає в приведенні вхідних даних у форму вектора білого шуму і т. д. Зниження розмірності можна робити вручну, але цим непогано займаються і автокодувальники.

Занурення слів (word embendding) – вхідні векторні уявлення слів (з гігантською розмірністю словника), вхідні шари мережі переводять або ***занурюють їх у простір ознак***, багато меншої розмірності. І далі використовуються вектори тільки такої розмірності в просторі ознак.

Векторизація тексту. Розглянемо елементи Обробки Природної Мови (ОПМ). Тексти бувають різного розміру. Тут проблема в перетворенні змінного числа атрибутів у постійне число ознак. Розглянемо найпростіший метод векторизації – векторна модель тексту (Vector Space Model – VSM).

Формується масив, кожен елемент якого містить кількість входжень даного слова з номером, рівним індексу елемента. Кожне слово має свій номер.

1. Сегментація пропозицій.
2. Розбиття на лексеми. Розглядаються окремі слова.
3. Стемінг. Визначення основи слів.
4. Лематизації[78].
5. Видалення стоп-слів.
6. Векторизація.

Ще більш проста векторна модель – це мішок слів (вектор частот термів), який складається зі списку слів і лічильників їх входження (їх

компанії Zalando згрупували всі зображення в десять класів і надали мітки для кожного зображення. Набір даних містить 785 стовпців. Перший стовпець – це мітка класу (ціле число від 0 до 9), і 784, що залишилися містять значення пікселя зображення з відтінком сірого.

[78] Стемінг – це процес, який залишає лише корінь слів. Лематизація – більш коректний процес, на основі словника і морфологічного аналізу, призводить слово до його канонічної форми – леми.

зазвичай нормують – визначають ймовірності їх входження). Порядок слів і граматика не враховуються.

Метод векторизації TF–IDF вносить ваговий коефіцієнт значущості слова (вага зростає пропорційно частоті слова в документі і падає при високій частоті входжень в корпус (тобто набір документів)). Значення – величина TF–IDF – це множення частоти терма (term frequency – TF) на зворотну частоту терма в документі (inverse document frequency – IDF). Тут TF – продукт поділу частоти терма на загальну кількість термів у документі. Точніше, IDF – логарифм відношення загального числа документів у корпусі на число документів, що містять слово. Ще більш привабливим є застосування N-грам. Це послідовності N сусідніх елементів тексту. N-грами (зазвичай не більше 5 слів) використовують, коли потрібно передбачити наступне слово при моделюванні контексту. Слова моделюються за допомогою послідовностей N-грам[79]. При обчисленні TF–IDF видаляються ***стоп–слова,*** що часто зустрічаються, ***наприклад, артиклі та сполучники*** (такі як a, an, who, the, what ...), адже вони можуть перешкодити визначити вагу корисних слів. Тобто ця модель векторного простору будується на статистиці спільної зустрічальності слів. Спрощеним видом векторизації є ядерне хешування – прохід по тексту в цьому випадку одинарний для економії ресурсів, на відміну від багаторазового при TF–IDF.

Метод векторизації Word2Vec[80]. Більш ґрунтовним є застосування Word2Vec, який оцінює схожість слів, вивчаючи і навчаючись контексту, де вони зустрічаються. Модель формує список занурень слів (тобто їх уявлень у векторній формі). В цьому випадку полегшується визначення близькості слів (відстаней між векторами) і взагалі з'являється можливість здійснювати операції над ними, подібні арифметичним.

На вхід алгоритму Word2Vec подається великий корпус текстів з мінімальною попередньою обробкою. Система сама знаходить семантичну близькість і складає списки синонімів.

Використовуючи ***дистрибутивну гіпотезу – сенс слова визначається контекстом***, в якому воно часто зустрічається, – побудували модель, яка передбачає слово за його сусідами; або пророкує сусідів цього слова, причому результати в значній мірі збігаються. Це варіант моделі переносу знань (transfer learning): навчаємо на великій вибірці модель і застосовуємо результати для невеликих масивів.

Word2Vec – це апаратне подання до форми штучної нейронної мережі, програмне – з використанням відкритого коду. Завантажується

---

[79] N-грам з букв (графем) використовуються для ідентифікації мов.

[80] Див. [3]. Вихідну 300-мірну модель Word2vec Google можна знайти на Google Drive за адресою https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM

великий текст (корпус), створюється словник, і кожному слову протиставляється вектор.

Підрахунки виявлення й частоти токенів були отримані, наприклад, для еталонної реалізації Word2Vec на 100 млрд слів з корпусу Google News. До речі, в предметних областях зі своєю термінологією, потрібно працювати з предметно-орієнтованою моделлю векторизації слів.

Ідея word2vec [2] полягає в тому, щоб можна було кожному слову і зі словника V ($i \in V$) поставити у відповідність вектор ознак w_i (вектор уявлення слова – word embedding) розмірності d, $w_i \in R$, d (тут d кілька сотень). Висловлюючи тепер ймовірність того, що слово і з'являється в контексті ($c_1,...c_n$), як функцію векторів ознак, отримаємо; $p(i \mid c_1,...c_n) = f(w_{c1},...w_{cn};\theta)$ тут $w_{c1},...w_{cn}$ – це вектори слів з контексту, а f - деяка функція з параметрами θ, на вхід якої подаються вектори слів. Можна навчати параметри функції f і вектори уявлень w_i, оптимізуючи логарифм загальної правдоподібності корпусу

$$L(W,\theta) = \frac{1}{T}\sum_t \log f(w_{c1},...w_{cn};\theta) + R(W,\theta),$$

де t пробігає всі доступні в корпусі вікна слів від 1 до T, а $R(W,\theta)$ тут регуляризатор поправки.

Зрозуміло, що близькі слова мають компоненти векторів, що слабо відрізняються і це дозволяє використовувати ці явища вже для визначення семантичної близькості слів. Передбачено дві опції. Алгоритм CBoW (Continuous Bag of Words) пророкує поточнє слово, виходячи з контексту. Алгоритм Skip–gram діє навпаки: використовує дане слово, щоб передбачати можливих сусідів – оточуючі слова.

Бібліотека для обробки тексту на природній мові Apache OpenNLP виконує для вкладених в бібліотеку мов:

1.Токенізацію – розбір на лексеми для отримання ідентифікованих послідовностей. Токен подібний лексемі – слову, але може бути трохи складніше – наприклад, містить свій ідентифікатор і послідовність символів лексеми (подібно угрупованню букв у слові).

2. Сегментацію пропозицій, розбиття на фрагменти тексту, розбиття по ключових словах.

3. Тегування (ідентифікація) частин мови – розмітка (POS tagging, part–of–speech tagging) – визначення частини мови і граматичних характеристик слів у тексті (корпусі) з приписуванням їм відповідних тегів. POS tagging – це перший з етапів комп'ютерного аналізу тексту.

4. Виділення названих об'єктів.

5. Синтаксичний аналіз.

Дерева категорій слів. При великому обсязі словника нейронні мовні моделі передбачають створення ієрархії категорій слів, потім дерева – категорії категорій слів і т. д.

Навчання. Нейронній мережі кожне зі слів представляється у вигляді унітарного вектора. Вихідний вектор виконує зіставлення – занурення (embedding) нейронної мережі також подібне до унітарного.

Логістична функція активації вузлів вихідного шару обчислює ймовірність виявлення вихідного слова серед тих, що оточують вхідне слово. До речі, вихідний вектор ймовірностей слів потім можна перетворити в унітарний вектор, в якому слову з максимальною ймовірністю відповідає 1, а всім іншим словами – 0. Це спрощує обчислення функції втрат. А можна цього не робити, залишивши оточення даного слова зі своїми можливостями, що навіть корисніше. Після завершення навчання нейронної мережі часто можна побачити, що ваги відображають семантичний сенс. Після навчання вектори семантично подібних слів будуть також подібні.

Важливою підмогою в навчанні є використання запропонованих IT–гігантами ембедінгів – попередньо навчених на великих корпусах моделей (наприклад, BERT – модель, створена і навчена в Google) для отримання векторних уявлень слів і пропозицій. Використовуючи разом з цими моделями свій мережевий класифікатор, можна досягти більшого успіху[81]. Головна перешкода сьогодні полягає в тому, що модель генерує слово за словом і генерує зв'язний текст, «дивлячись далеко назад, вона не тримає думки, дивлячись далеко вперед», немає плану донести до співрозмовника якусь ідею, просто розвиває певну тему, не знаючи, куди її занесе. Тобто модель не усвідомлює себе[82], «не будує плани і не має власних цілей». Відповідно, наступне покоління моделей мови повинно мати таку можливість – «триматися задуму і йти до кінцевої мети[83]». Вважають, що без цього неможливо буде створити штучний розум, який би зацікавлено,

[81] Однак побудовані на таких моделях (нехай навіть навчених на величезних корпусах) висловлювання поки не дозволяють досить прийнятно забезпечувати адекватний зміст згенерованого тексту. Отримуючи інформацію від різних джерел, алгоритми не забезпечують необхідний контроль і узгодженість окремих фрагментів згенерованого тексту. Хоча людина і сама в разі довгих монологів не раз виявляється суперечливою.

[82] Так і хочеться сказати – не усвідомлює себе самодостатньою особистістю. Хоча, про що це ми?

[83] Цікаво, що така ж поведінка властива і абсолютній більшості людей, які, володіючи різноманітною інформацією, не намагаються зробити потрібні висновки (тобто не можуть міркувати) і використовувати цю поінформованість для вирішення особистих і корпоративних амбітних завдань.

аргументовано міг відстоювати і доводити свою точку зору[84]. Поки навіть фахівцям не зрозуміло, як строго визначити поняття свідомості[85]. Як і раніше перехід від найдосконаліших нейромереж до людського інтелекту представляється неможливим (leap of faith).

У 2017 р. Google Brain розробила альтернативну не рекурентну архітектуру Transformer, за рахунок схеми кодування положення слів з використанням перетворення Фур'є. Багатошаровий кодер знаходить абстрактний сенс всіх слів на вході, їх взаємний вплив, використовуючи механізм уваги. Використовується архітектура seq2seq. У 2018 році команда з Open AI показала модель GPT (Generative Pretraining Transformer). Використовують попередньо навчені семантичні вектори слів. Це вже навчається ціла модель мови.

Пізніше з'явився варіант подібної моделі GPT2, навченої на 40 ГБ-му масиві текстів з інтернету, з результатами, які можна порівняти з людськими. Число параметрів моделі ще набагато менше кількості прикладів, що дозволяє сподіватися на більше. Якість генерування GPT2 текстів вражає [3]. Компанії вже намагаються не афішувати подібні технології, все більше побоюючись вибухового зростання фейкового контенту в Інтернеті.

Література до розділу 11

1. Черняк Е. Введение в глубокое обучение / пер. с англ. – СПб. : Диалектика, 2020. – 192 с. : ил.

2. Bengio Y., Ducharme R., Vincent P. A Neural Probabilistic Language Model // Journal of Machine Learning Research, 2003, vol. 3. — P. 1137–1155.

3. Mikolov T., Corrado D. Efficient Estimation of Word Representations in Vector Space, Sep. 2013 (https://arxiv.org/pdf/1301.3781.pdf).

4. Larsen ‘The Myth of Artificial Intelligence: Why Computers Can’t Think the Way We Do./ Kindle Edition.

5. Искусственный интеллект. Обработка естественного языка, распознавание и синтез речи. Альманах. Moscow Institute of Physics and Technology, Ceo/Cdo Pro-library. 179 с.

[84] Див. [4].

[85] Коли пробують це зробити, далі «I know it, when I see» часто діло не йде. Мабуть, мозок вирішує багато завдань: функціонування організму, спостереження і аналізу навколишнього середовища, а у сучасної людини аналізу і синтезу гри образів і уявлень, які відносяться до її внутрішнього світу – своєрідної віртуальної реальності. Цей аналіз і синтез, гра уявлень і образів відбувається поза увагою людини. Тільки ті розумові процеси з численного набору перманентних розумових дій нашого мозку, на яких загострюється наша увага і до яких природа дала нам доступ, перекладаються (можливо, з труднощами) на зрозумілу для нас мову, і ми називаємо це свідомістю.

Розділ 12. СПРОБИ АВТОМАТИЧНОГО ВИЯВЛЕННЯ ЗАТВЕРДЖЕНОЇ ТЕМИ В ТЕКСТОВИХ МАСИВАХ

Навчивши мережу мови, на базі розпізнавання букв і символів, а також простих слів, слід розділити ці слова за групами (наприклад, група – дієслова) і за смисловими пріоритетами (наприклад, використовуючи порядок слів в англійській мові) і т. д. Розпізнавання елементів букв, символів, слів за групами має бути визначено на рівні, наприклад, 60%. Виділяючи з тексту (внесеного вислову, пропозиції-речення) елементи, які мають вже семантичне (смислове) значення, з них конструюється простий предикат (твердження ***Claim***), що містить головний сенс внесеної кожної пропозиції-висловлення.

Далі можна розглянути спробу авторів [1] знайти твердження з текстів.

Може додаватися опція формування множини нових предикатів ***Predicate Recycling*** (як в теорії предикатів) з набору сформованих з тексту предикатів-речень. Це здатне створювати хмару прив'язаних до первісного набору предикатів смислів для майбутніх доказів і висловлювань. З головного предиката може формуватися предикат–заперечення (наприклад, ***Automatic Claim Negation***). Далі все йде в наборі – ***твердження–заперечення***. Тепер в новому створюваному тексті виділяються свої прості предикати і відшукується відповідність з колишніми простими предикатами (твердженням або запереченням). Формуються посилання на підтвердження твердження або заперечення. Розглянемо, як в текстах мережа знаходить твердження.

> Використовується механізм **Wikification** з **Вікіпедії**, який полегшує звуження вибору пропозицій по темі. Виділення пропозицій-речень з текстових масивів за допомогою текстів та за допомогою **OpenNlp** і усунення неоднозначності TagMe[86] на Wikify. Застосовується розроблений і верифікований в роботі [2] алгоритм: створюється запит ***Claim Sentence Query (CSQ),*** що дозволяє виділяти набір пропозицій-речень, ***Claim Lexicon (CL),*** що відповідають темі.

Твердження можуть бути аргументовані або не аргументовані.

Приклад аргументованого твердження: Ясно, що кваліфікація повинна бути єдиним визначальним фактором при пошуку роботи

[86] Система вікіфікації TagMe – це система усунення неоднозначності, вбудована в Вікіпедію, заснована на картуванні (створенні карти) набору слів (Приклад: Система семантичних тегів на основі Вікіпедії). POS-теги – завдання маркування кожного слова в реченні відповідної частини мови (POS).

(Clearly, qualifications should be the only determining factor when competing for a job).

Приклад неаргументованого твердження: У 1961 році Джон Ф. Кеннеді став першим, хто використав термін «позитивна дія» в його сучасному розумінні (In 1961 John F. Kennedy became the first to utilize the term «affirmative action» in its contemporary sense).

Евристика: твердження (***Claim*)** зазвичай (і) семантично пов'язані з темою; і (іі) існують в рамках уявлення з ідентифікованими структурними властивостями. Тобто щось повинно на них вказувати.

Кожне твердження містить основну концепцію (***main concept – MC***) теми або згадку про неї.

На щастя, існує вже чинний засіб виявлення і виділення згадок – тверджень (***TagMe:*** анотація коротких фрагментів тексту «швидко» – по об'єктах Вікіпедії), який відображає поверхневі форми в заголовках Вікіпедії (також відомі як вікіфікації = a. k. a ***Wikification***[87]), щоб упорядкувати процес майнінгу – тут цей термін означає тільки пошук пропозицій-речень, в яких MC виявлений. Вікіфікації можуть допомогти запобігти дрейф у значеннях по темі, тобто уникнути відхилень по подібним, але все ж зовсім іншим темам.

Наприклад, розглянемо тему «Шлюб застарів», для якого MC – Шлюб. З величезної кількості пропозицій Вікіфікація виявить споріднену тему «Одностатевий шлюб».

Частина мови (токен[88]) 'that' часто використовується в якості попередника до твердження. Це створює лексикон тверджень – (***Claim Lexicon – CL***), з якого ми отримуємо запит **– (*Claim Sentence Query – CSQ*)**, що складається з упорядкованого триплета: ***that – MC – CL.***

> Під вікіфікацією зазвичай розуміють дві речі: процес оформлення вікі-розмітки; процес проставляння внутрішніх посилань. Пошук взаємозв'язків між словами в тексті і статтями «Вікіпедії» – надзвичайно популярне завдання, відоме як вікіфікація. Система вікіфікації TagMe – це система усунення неоднозначності, вбудована в Вікіпедію, заснована на картуванні (створення карти) набору слів (Приклад: Система семантичних тегів на основі

[87] Вікіфікація включає проставлення внутрішніх посилань на ключові слова статті там, де вони доречні. Система вікіфікації TAGME – це система усунення неоднозначності, вбудована в Вікіпедію, заснована на картуванні (створення карти) набору слів (Приклад: Система семантичних тегів на основі Вікіпедії). POS-теги – завдання маркування кожного слова в реченні відповідної частини мови (POS).

[88] Токенізувати текст (word_tokenize) – привласнити деякий певний токен (частина мови) кожному слову. Приклади з набору скорочень – VBZ – дієслово, нині з 3-ї особи; FW – іноземне слово і т. д.

Вікіпедії). POS-теги – завдання маркування кожного слова в реченні відповідної частини мови (POS).

Було оцінено продуктивність пропонованої системи. Застосовано крауд-мітки[89] до прогнозованих заявками по всіх 150 темах у **dev-** і **test-**наборах. Для кожної теми ми відзначили 50 кращих прогнозів, або всі прогнози, якщо їх було менше. Прогноз вважався правильним, якщо більшість анотаторів позначило це як розшукане твердження[90]. Результати виявились краще, ніж у розробників вручну за рахунок того, що система здатна узагальнювати (охоплювати) різні теми. Чим більше пропозицій з **CSQ**, тим більше точність. Перевагою також є можливість роботи у великих текстах, в яких співвідношення позитивних випадків низька (2,4% пропозицій-речень, що містять **MC**). Розглянуто тільки теми єдиної концепції, якій відповідає сторінка Вікіпедії. Розширення пропонованого підходу основи на більш складні запити, що охоплюють більше, ніж окрема концепція, заслуговує подальшого вивчення. Але навіть без такого розширення відзначимо, що спірні теми часто характеризуються відповідною сторінкою у Вікіпедії. Підхід авторів націлений на твердження, в яких **MC** ідентифіковано інструментом вікіфікації. Хоча це дозволяє знайти твердження, в яких **MC** виражається через різні форми поверхневого уявлення, але помилки Вікіфікації також поширюються на продуктивність системи[91].

Література до розділу 12

1. Unsupervised corpus–wide claim detection. R. Levy et.al. Proceedings of the 4th Workshop on Argument Mining, pages 79–84/ Copenhagen, Denmark, September 8, 2017.

2. Paolo Ferragina and Ugo Scaiella. 2010. Tagme: on–the–fly annotation of short text fragments (by wikipedia entities). In Proceedings of the 19th ACM international conference on Information and knowledge management, pages 1625–1628. ACM.

3. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

4. Ehud Aharoni, Anatoly Polnarov, Tamar Lavee, Daniel Hershcovich, Ran Levy, Ruty Rinott, Dan Gutfreund, and Noam Slonim. 2014. A benchmark dataset for automatic detection of claims and evidence in the context of controversial topics. In Proceedings of the First Workshop on Argumentation Mining, pages 64–68.

[89] Через платформу CrowdFlower: www.crowdflower.com/, подробиці в додаткових матеріалах, зазначених у статті авторів.

[90] На кожного кандидата потрібно не менше 10 анотаторів. Після 10 анотацій подальші інструкції збираються до тих пір, поки або досягнуто 90% згоди, або 15 анотацій.

[91] Детальніше див. [4].

ЧАСТИНА ІІІ

ПРАКТИКУМ ЗІ СТВОРЕННЯ НЕЙРОННИХ МЕРЕЖ

Розвиток програмних та обчислювальних засобів дозволяє на сучасній стадії розвитку технології конструювати та навчати нейронні мережі помірної глибини буквально на персональному комп'ютері з наявністю відеокарти з технологією обчислення на графічних процесорах CUDA.

Compute unified device architecture (CUDA) – архітектура паралельних обчислень, розроблена компанією Nvidia. CUDA забезпечує можливість обчислень загального призначення на GPU від Nvidia [1]. Програмісти використовують мову програмування «C для CUDA» (мова програмування C з розширеннями і обмеженнями) для написання алгоритмів, що виконуються на GPU. Також CUDA надає обчислювальні інтерфейси для Python, Perl, Fortran, Java, Mathematica, MATLAB та інших мов програмування і прикладних програм [2].

Переваги CUDA:

– висока швидкість обчислень (на GPU порівняно з CPU від декількох до кількох десятків разів);

– наявність інтерфейсів для різних мов програмування;

– безкоштовність;

– постійний розвиток;

– поширеність порівняно з іншими технологіями GPGPU.

Недоліки CUDA:

– додаткова складність написання програм для GPU порівняно з CPU;

– не підтримуються алгоритми, які не піддаються розпаралелюванню;

– архітектуру CUDA підтримує тільки виробник Nvidia;

– GPU, що мають старі версії архітектури CUDA, не підтримують можливості нових версій;

– швидкість обчислень у подвійній точності менше, ніж в одинарній точності, на відміну від CPU, які оптимізовані під обчислення в подвійній точності (на CPU швидкість обчислень у подвійній точності і в одинарній збігаються);

– висока вартість т. зв. «професійних GPU» – сімейств GPU Quadro і Tesla, оптимізованих для обчислень загального призначення, тобто які мають високу швидкість обчислень у подвійній точності.

Де-факто основною мовою програмування для проектування цих архітектур став Python за рахунок доступності синтаксису, великої відкритої бази всіляких оптимізованих бібліотек і фреймворків, що дозволяє швидше порівняно з багатьма іншими мовами створювати робочі прототипи, які дозволяють проводити попередню оцінку обраного підходу, пов'язаних із застосуванням нейронних мереж.

До переваг мови належать гнучкість, розширюваність, простота синтаксису, інтерпретованість, PEP (єдиний стандарт для написання коду), Open Source мови та наявність дружнього ком'юніті. Незважаючи на всі явні переваги, Python також має ряд недоліків, які все ж таки варто враховувати при проектуванні додатка виходячи з цілей і вимог до проектів. Отже, до недоліків відносяться низька продуктивність (це обумовлено тим, що Python інтерпретується), незвичний синтаксис при переході з іншої мови, динамічна типізація (хоча в актуальній версії мови вже є сувора типізація) [3].

Оскільки основою нейронних мереж є матриці і увесь процес навчання та видачі відповідей складається з різних операцій лінійної алгебри, то створити найпростішу мережу можна і без використання масштабних фреймворків. Для першого практичного прикладу в поточному розділі достатньо мати бібліотеку для операцій з матрицями NumPy, що містить безліч оптимізованих функцій, що працюють набагато швидше за вбудовані операції з даними мови Python.

Розділ 13. ПРОЕКТУВАННЯ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ З ЗАСТОСУВАННЯМ БІБЛІОТЕКИ NUMPY

Під класифікацією зображення зазвичай розуміють присвоєння якомусь зображенню мітки, згідно з отриманим уявленням про наявні зразки. Для цього типу завдань існує безліч відкритих наборів даних (датасетів), нерідко попередньо віднормованих за розміром. Так, датасет Fruit360 [7] зберігає зображення фруктів та містить 60 класів: яблуко, гуава, банан, фінік, ківі та інші категорії фруктів. У цьому прикладі буде розібрано роботу з чотирма класами: яблуко Braeburn, лимон Meyer, манго, малина. Кожен клас має близько 491 зображень для навчання та 162 зображення для тестування. Розмір зображення 100x100 пікселів.

Підготовка даних. На етапі попередньої обробки даних важливо враховувати факт, що простір кольорів RGB не ізолює інформацію про колір від іншої інформації, наприклад, освітлення. Якщо використовувати RGB для представлення зображення, в розрахунках потрібно враховувати всі три канали. Тому набагато зручніше працювати з колірним простором HSV, який ізолює інформацію про колір у єдиний канал. У такому разі за колір відповідає канал hue(H), відтінок.

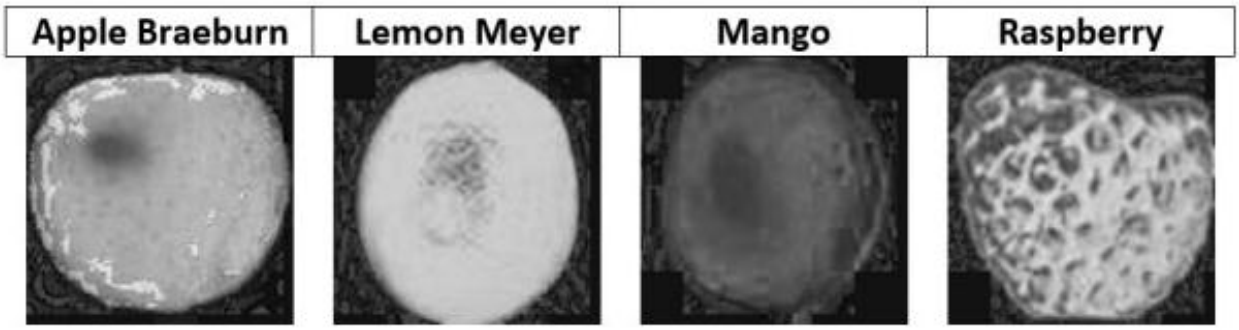


Рис. 34. Вигляд hue-каналу для вибраного набору фруктів

Канал hue також має розмір 100x100 пікселів. Якщо ANN використовувати канал цілком, тоді вхідний шар повинен мати 10000 нейронів, це надмірної інформації. Щоб зменшити кількість використовуваних даних, можна використовувати гістограму для представлення відтінку каналу. Така гістограма матиме істотно меншу кількість використовуваної інформації (bin), показуючи при цьому відносно однозначні відмінності між фруктами.

Зображення чотирьох класів можна обійти у циклі та витягти ознаки з усіх зображень. Відповідно до сукупного числа зображень у 4 класах (X) та довжини вектора ознак, витягнутих з кожного зображення (Y),

створюється масив NumPy з нулів розмірності (X, Y) і зберігається в змінній dataset_features [4].

Для зберігання міток класів для кожного зображення можна використовувати ще один NumPy масив у змінній outputs. Яблуку відповідає мітка 0, лимону – 1, манго – 2, малині – 3. Найзручніше дані для завдань класифікації зображень розташовувати у вигляді директорій з іменами, що відповідають назвам фруктів, записаним у списку fruits.

Алгоритм, представлений у програмному коді нижче, містить такі пункти:

• з кожного зображення отримується діаграма каналу hue;

• кожному зображенню приписується позначка класу;

• вилучені ознаки та мітки класів зберігаються за допомогою бібліотеки pickle.

Псевдокод цього алгоритму представлений нижче:

• Формування списку об'єктів та ініціалізація заповненого нулями numpy масиву, що за розміром збігається з розміром навчальної вибірки. Формування векторів відповідей.

• Прохід у циклі по всьому зображенню та отримання HSV характеристики для кожного з них з подальшим збереженням гістограми зображення, що є способом помітно зменшити кількість навчальних параметрів.

Код також містить використання бібліотек skimage для різних операцій із зображеннями, os для можливості обходу в циклі шуканих зображень, pickle для серіалізації ознак і міток класів.

```
import numpy
import skimage.io, skimage.color, skimage.feature
import os
import pickle

fruits = ["apple", "raspberry", "mango", "lemon"]
#492+490+490+490=1,962

dataset_features = numpy.zeros(shape=(1962, 360))
outputs = numpy.zeros(shape=(1962))
idx = 0

class_label = 0

for fruit_dir in fruits:
  curr_dir = os.path.join(os.path.sep, fruit_dir)
  all_imgs = os.listdir(os.getcwd()+curr_dir)
  for img_file in all_imgs:
```

```
    fruit_data = skimage.io.imread(fname=os.getcwd()+curr_dir+img_file, as_grey=False)
    fruit_data_hsv = skimage.color.rgb2hsv(rgb=fruit_data)
    hist = numpy.histogram(a=fruit_data_hsv[:, :, 0], bins=360)
    dataset_features[idx, :] = hist[0]
    outputs[idx] = class_label
    idx = idx + 1
  class_label = class_label + 1

with open("dataset_features.pkl", "wb") as f:
  pickle.dump(dataset_features, f)

with open("outputs.pkl", "wb") as f:
  pickle.dump(outputs, f)
```

Тепер кожне зображення представлено вектором ознак, що складається із 360 елементів. Вектор фільтрується таким чином, щоб зберігати найрелевантніші елементи для поділу 4 класів. Новий укорочений вектор ознак має довжину 102 замість 360. Використання меншої кількості елементів допомагає прискорити роботу алгоритму. Змінна dataset_features матиме розмір 1962x102.

Реалізація нейронної мережі. На рис. 35 представлена структура нейронної мережі. Вхідний шар має 102 вхідні нейрони, які відповідають за однозначну передачу на вхід вектора ознак; два приховані шари складаються з 150 і 60 нейронів, відповідно, вихідний шар має 4 виходи (по одному на кожен клас). Між кожними шарами існує матриця вагових коефіцієнтів, при проходженні сигналу від шару до шару відбувається матричне перемноження, яке в результаті надасть на виході останнього шару вектор, розмірність якого буде відповідати кількості класів. Цей процес зазвичай називають прямим проходженням сигналу, і він є першим етапом навчання нейронної мережі.

Після зчитування раніше збережених ознак і міток фільтрації ознак визначаються вагові матриці в кожному шарі. Їм випадково приписуються значення в діапазоні від -0.1 до 0.1. Наприклад, змінна *input_HL1_weights* зберігає матрицю ваги при переході від вхідного до першого прихованого шару. Розмір цієї матриці визначається відповідно до кількості елементів у векторі ознак та кількості нейронів у прихованому шарі.

Після створення вагових матриць наступним кроком є матричне множення. Змінна *H1_outputs* зберігає виходи від перемноження вектора ознак обраного об'єкта та матрицею ваг між вхідним та першим прихованим шарами.

Зазвичай, щоб виявити нелінійні залежності між входами та виходами, функція активації застосовується до виходів кожного прихованого шару.

Результати матричного множення проходять через функцію сигмоїдної активації.

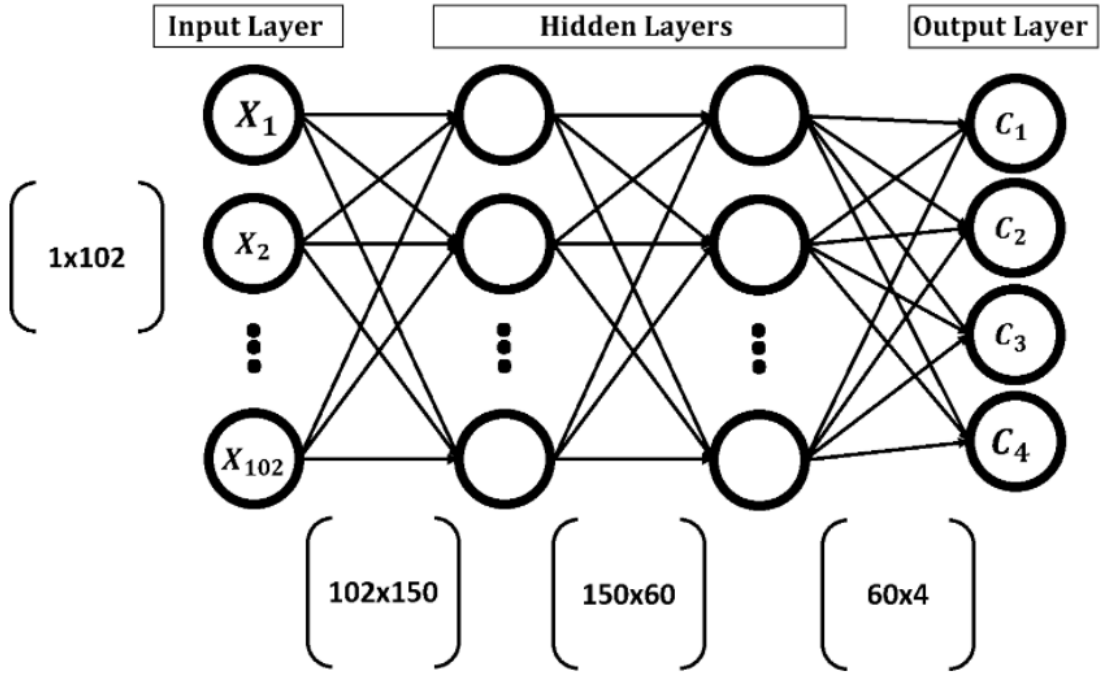


Рис. 35. Архітектура ANN мережі з двома прихованими шарами

Після генерації виходів на фінальному шарі робиться прогноз. Передбачувана позначка класу зберігається в змінній *predicted_label* [5].

Псевдокод цього алгоритму представлений нижче:

1. Формування списку функцій, таких як функції сигмоїди, ReLU, функції оновлення ваг та основної функції навчання мережі, в якій вхідними параметрами буде виступати кількість епох навчання, матриця ваг, вхідні та вихідні дані, швидкість навчання та функція активації.
2. Формування архітектури, що містить 2 приховані шари на 150 і 60 нейронів відповідно і вихідний шар з 4 нейронів за кількістю міток результуючих класів.
3. Генерація рівномірно розподілених випадкових ваг у діапазоні від -0.1 до 0.1 для кожного з шарів.
4. Призначення для кожного із шарів функції активації – сигмоїда або ReLU.
5. Для N ітерацій (у навчальних елементів) до кожного елементу навчальної вибірки послідовно застосувати матричний твір з матрицею терезів.
6. До кожного з матричних творів застосувати функцію активації, яка вказана як застосовна у конкретному шарі.
7. Отримання мітки класу після проходження шарів і, якщо передбачення мітки не збігається з реальним класом, застосувати оновлення ваги.

```
 import numpy
import pickle

def sigmoid(inpt):
 return 1.0 / (1 + numpy.exp(-1 * inpt))

def relu(inpt):
 result = inpt
 result[inpt < 0] = 0
 return result

def update_weights(weights, learning_rate):
 new_weights = weights - learning_rate * weights
 return new_weights

def train_network(num_iterations, weights, data_inputs, data_outputs, learning_rate,
activation="relu"):
 for iteration in range(num_iterations):
  print("Itreation ", iteration)
  for sample_idx in range(data_inputs.shape[0]):
   r1 = data_inputs[sample_idx, :]
   for idx in range(len(weights) - 1):
    curr_weights = weights[idx]
    r1 = numpy.matmul(a=r1, b=curr_weights)
    if activation == "relu":
     r1 = relu(r1)
    elif activation == "sigmoid":
     r1 = sigmoid(r1)
  curr_weights = weights[-1]
  r1 = numpy.matmul(a=r1, b=curr_weights)
  predicted_label = numpy.where(r1 == numpy.max(r1))[0][0]
  desired_label = data_outputs[sample_idx]
  if predicted_label != desired_label:
   weights = update_weights(weights,
    learning_rate=0.001)
 return weights

def predict_outputs(weights, data_inputs, activation="relu"):
 predictions = numpy.zeros(shape=(data_inputs.shape[0]))
 for sample_idx in range(data_inputs.shape[0]):
  r1 = data_inputs[sample_idx, :]
   for curr_weights in weights:
    r1 = numpy.matmul(a=r1, b=curr_weights)
   if activation == "relu":
    r1 = relu(r1)
```

```
    elif activation == "sigmoid":
      r1 = sigmoid(r1)
   predicted_label = numpy.where(r1 == numpy.max(r1))[0][0]
   predictions[sample_idx] = predicted_label
  return predictions

f = open("dataset_features.pkl", "rb")
data_inputs2 = pickle.load(f)
f.close()

features_STDs = numpy.std(a=data_inputs2, axis=0)
data_inputs = data_inputs2[:, features_STDs > 50]

f = open("outputs.pkl", "rb")
data_outputs = pickle.load(f)
f.close()

HL1_neurons = 150
input_HL1_weights = numpy.random.uniform(low=-0.1, high=0.1,
size=(data_inputs.shape[1], HL1_neurons))

HL2_neurons = 60
HL1_HL2_weights = numpy.random.uniform(low=-0.1, high=0.1,
size=(HL1_neurons, HL2_neurons))

output_neurons = 4
HL2_output_weights = numpy.random.uniform(low=-0.1, high=0.1,
size=(HL2_neurons, output_neurons))

weights = numpy.array([input_HL1_weights,
  HL1_HL2_weights,
  HL2_output_weights])

weights = train_network(num_iterations=10,
  weights=weights,
  data_inputs=data_inputs,
  data_outputs=data_outputs,
  learning_rate=0.01,
  activation="relu")

predictions = predict_outputs(weights, data_inputs)
num_flase = numpy.where(predictions != data_outputs)[0]
print("num_flase ", num_flase.size)
```

Головною функцією у наведеному вище коді є *train_network,* оскільки навчання відбуватиметься саме в ній через проходження в циклі по всіх об'єктах датасету з повторенням усіх кроків для кожного елемента. Кількість ітерацій, ознаки, позначки класів, ваги, швидкість навчання та функція активації залишаються постійними. За функцію активації береться ReLU або сигмоїда. ReLU – функція з порогом значення, вона повертає вхідне значення, коли воно більше за нуль. Інакше ReLU повертає 0.

У разі помилкового передбачення для певного об'єкта ваги оновлюються за допомогою функції *update_weights.* У цьому прикладі для оновлення ваг не використовується алгоритм оптимізації. Вага просто оновлюється відповідно до параметра learning rate (швидкість навчання). Сукупна точність отриманої моделі вкрай не більше 50%, що свідчить про непридатність моделі для прикладних цілей. Модель вимагає досягнення кращої точності вже використання алгоритму оптимізації. Наприклад, для реалізації ANN у бібліотеці *scikit-lear*n використовується градієнтний спуск [6].

Самостійна робота. Для поточного завдання застосувати будь-який алгоритм оптимізації та довести точність класифікації хоча б до 60%.

Література до розділу 13

1. Гущин І.В. Моделювання фізичних процесів із використанням технології CUDA: монографія / І. В. Гущин, В. М. Куклін, О. В. Мішин, О. В. Приймак. – Харків: ХНУ імені В. Н. Каразіна, 2017. – 116 с.

2. Что такое CUDA? – Nvidia Corporation [Электронный ресурс]. – Режим доступа: http://www.nvidia.ru/object/what_is_cuda_new_ru.html. – Электрон. версія, 2014. – HTML формат.

3. Python official site https://www.python.org/

4. NumPy manual https://numpy.org/doc/stable/

5. Practical computer vision applications з використанням глибокого навчання з CNNs. Ahmed Fawzy Gad, Apress, ISBN: 978-1484241660, Березень, 2019

6. Scikit manual https://scikit-learn.org/stable/user_guide.html

7. Fruit 360 dataset https://www.kaggle.com/moltean/fruits

Розділ 14. ВИКОРИСТАННЯ ФРЕЙМВОРКУ TENSORFLOW У ЗАВДАННІ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

Для вирішення прикладних завдань застосовують комплексні фреймворки, що містять безліч оптимізованих функцій та інструментів моніторингу.

Одним із подібних фреймворків є TensorFlow, що дозволяє ефективно оперувати з багатовимірними матрицями – тензорами, що особливо вигідно у світлі обсягів інформації, з якими мають справу сучасні нейронні мережі в контексті поставлених перед ними завдань.

Далі розглядається приклад [1] застосування TensorFlow для створення простої тришарової нейронної мережі. Загальновідомий набір рукописних символів MNIST міститься в TensorFlow (доступний за допомогою завантажувача) та являє собою набір зображень цифр у відтінках сірого розміром 28 × 28 пікселів. Він має 55 000 навчальних рядів, 10 000 рядків тестування та 5000 рядків перевірки.

Початок роботи з даними – завантаження бібліотек та датасету. В об'єкті Mnist буде міститися набір зображень з поданням міток у вигляді one hot міток. One_hot=True – аргумент вказує, що мітки, пов'язані з кожним проектованим зображенням, замінюються на вектор з «одним гарячим» вузлом, а решта вузлів дорівнює нулю, тобто [0, 0, 0, 0, 1, 0, 0 0, 0, 0]. Це дозволяє легко подавати його у вихідний шар нейронної мережі. Навчальний коефіцієнт призначається рівним 0.5 і розмір пакета (batch) рівним 100 (тобто одноразово мережа буде навчатися на 100 зразках з усього датасету).

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

learning_rate = 0.5
epochs = 10
batch_size = 100
x = tf.placeholder(tf.float32, [None, 784]) # input x - for 28 x 28 pixels = 784
y = tf.placeholder(tf.float32, [None, 10]) # 10 digits
```

Слід звернути увагу, що вхідний рівень містить 784 вузли, що відповідають 28 x 28 = 784 пікселів, а вихідний рівень – 10 вузлів, що відповідають 10 можливим цифрам (класам). Далі необхідно налаштувати змінні ваги та усунення для кожного з шарів нейронної мережі. Завжди

має бути L-1 кількість тензорів ваги/зміщення, де L – кількість шарів. Тому в цьому випадку потрібно налаштувати два тензори для кожного, оскільки мережа передбачається тришаровою, архітектурно схожою на представлену на рис. 35:

```
W1 = tf.Variable(tf.random_normal([784, 300], stddev=0.03), name='W1')
b1 = tf.Variable(tf.random_normal([300]), name='b1')
W2 = tf.Variable(tf.random_normal([300, 10], stddev=0.03), name='W2')
b2 = tf.Variable(tf.random_normal([10]), name='b2')
```

Значення ваг ініціалізуються за випадковим нормальним розподілом із середнім значенням, рівним нулю, і стандартним відхиленням 0,03. TensorFlow має репліковану версію випадкової нормальної функції бібліотеки NumPy, яка дозволяє створити матрицю заданого розміру, заповнену випадковими вибірками, отриманими з даного розподілу. Створення змінних W1, W2, b1 та b2 відбувається досить типово, тільки необхідно враховувати розміри кожного наступного шару, які мають зменшуватися. Потім потрібно налаштувати входи вузлів та функції активації вузлів прихованого шару:

```
hidden_out = tf.add(tf.matmul(x, W1), b1)
hidden_out = tf.nn.relu(hidden_out)
```

У першому рядку виконується стандартне матричне множення ваги W1 на вхідний вектор X з додаванням зміщення b1. Матричне множення виконується з допомогою операції tf.matmul. Застосовуючи функцію активації relu до утворення матриці ваг W1 і входу X плюс усунення, можна отримати підсумковий вид змінної hidden_out.

Налаштування вихідного рівня y_:

```
y_ = tf.nn.softmax(tf.add(tf.matmul(hidden_out, W2), b2))
```

У цьому випадку використовується активація softmax для вихідного рівня. Використовуватиметься функція витрат крос-ентропії.

```
y_clipped = tf.clip_by_value(y_, 1e-10, 0.9999999)
cross_entropy = -tf.reduce_mean(tf.reduce_sum(y * tf.log(y_clipped) + (1 -
y) *
tf.log(1 - y_clipped), axis=1))
```

Перший рядок – операція, що перетворює вихід y_clipped версію, обмежену між 1-10 і 0.999999. Другий рядок – розрахунок крос-ентропії за

допомогою функції tf.reduce_sum, що бере суму заданої осі тензора. Налаштування оптимізатора в TensorFlow:

```
optimizer= tf.train.GradientDescentOptimizer(
        learning_rate=learning_rate).minimize(cross_entropy)
```

Тут використовується оптимізатор градієнтного спуску, який надається з TensorFlow. Ініціалізується він за допомогою швидкості навчання, яка вказана на самому початку. Після цього слід зазначити, що безпосередньо потрібно від оптимізатора – звести до мінімуму транзакційну операцію крос-ентропії [2]. Потім ця функція виконає градієнтний спуск.

Налаштування операції ініціалізації змінних та операції для вимірювання точності наших прогнозів:

```
inotrope = tf. global_variables_initializer()
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
tf.summary.scalar('accuracy', accuracy)
```

Операція пророкування correct_prediction використовує функцію tf.equal TensorFlow, яка повертає True або False в залежності від того, чи рівні його аргументи. Функція tf.argmax ідентична функції NumPy argmax, яка повертає індекс максимального значення у векторі чи тензорі. Тому операція correct_prediction повертає тензор розміру (m x 1) True та False значення, що визначають, чи правильно передбачила цифру нейронна мережа.

Псевдокод цього алгоритму:

1. Визначення параметрів збереження сесії
2. Обчислення кількості навчальних пакетів (batch)
3. Навчання мережі на наборі пакетів, застосування оптимізуючою функцією отримання усередненої точності класифікації.

Запуск процесу навчання нейронної мережі.

```
merged = tf.summary.merge_all()
writer = tf.summary.FileWriter('C:\\D')  # path to local file
with tf.Session() as sess:
  sess.run(init_op)
  total_batch = int(len(mnist.train.labels) / batch_size)
  for epoch in range(epochs):
    avg_cost = 0
    for i in range(total_batch):
```

```
    batch_x, batch_y = mnist.train.next_batch(batch_size=batch_size)
    _, c = sess.run([optimiser, cross_entropy], feed_dict={x: batch_x,
y: batch_y})
    avg_cost += c / total_batch
  print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost))
  summary = sess.run(merged, feed_dict={x: mnist.test.images, y:
mnist.test.labels})
  writer.add_summary(summary, epoch)
print("\nTraining complete!")
writer.add_graph(sess.graph)
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y:
mnist.test.labels}))
```

Запуск цієї програми має дати результат на тренувальній вибірці близько 98%. Тобто очевидно, що використання алгоритмів оптимізації навчання призводить до більш відповідних результатів.

Самостійна робота. На тестовій вибірці отримати певну точність передбачення, пояснити, чому точність вийшла більшою (меншою), ніж точність на тренувальній вибірці. Спробувати різні значення гіперпараметрів (наприклад, швидкість навчання).

Література до розділу 14

1. Гафаров Ф.М. Штучні нейронні мережі та додатки: навч. посібник / Ф. М. Гафаров, А.Ф. Галимянів. – Казань: Вид-во Казан. ун-ту, 2018. – 121 с.

2. TensorFlow manual https://www.tensorflow.org/versions

Розділ 15. ВИКОРИСТАННЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ У ЗАВДАННІ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

Згорткові нейронні мережі є однією з форм багатошарових нейронних мереж. Основною відмінністю є використання операції математичної згортки матриць, яка дає назву відповідним шарам і самій архітектурі.

Типова схема CNN складається з двох частин (рис. 34). Перша частина складається з шарів згортки (convolutional layer) і максимального пулу (max pooling), які виступають як екстрактор ознак. Друга частина складається з повнозв'язкового шару, який виконує нелінійні перетворення вилучених ознак та діє як класифікатор.

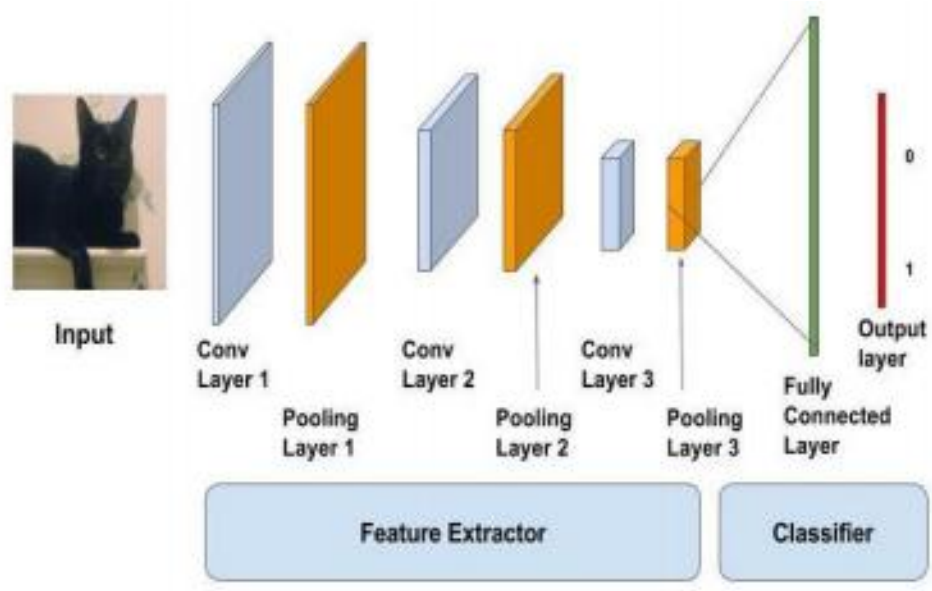


Рис. 36. Загальна схема згорткової нейроної мережі CNN

На наведеній вище діаграмі на вхід подається до мережі послідовних шарів Conv, Pool та Dense [3]. Вихідний сигнал може бути шаром softmax, що вказує, чи є кішка чи щось інше. Також як вихідний може бути використаний сигмоїдний шар, на виході якого буде ймовірність того, що зображення буде кішкою.

Згортку матриць можна розглядати як зважену суму між двома сигналами чи функціями. Приклад операції згортки на матриці розміром 5 × 5 з ядром розміром 3 × 3 показаний на рисунку 30. Ядро згортки ковзає по всій матриці для отримання карти активації.

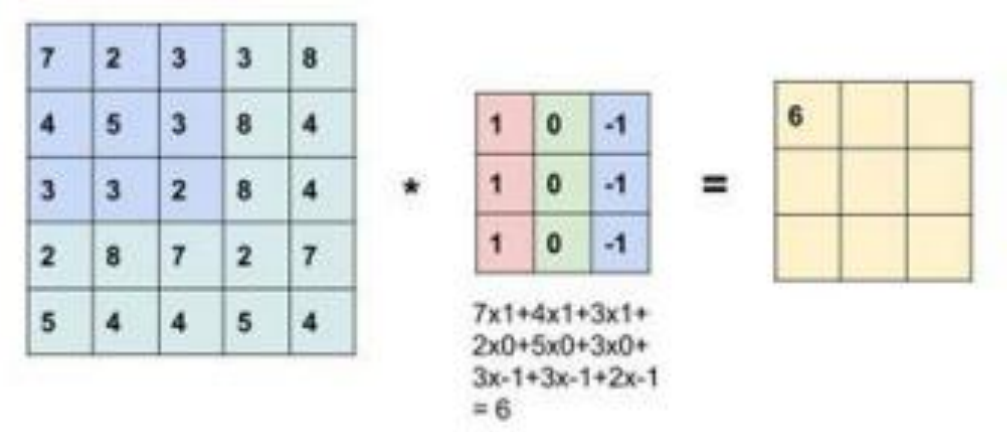


Рис. 37. Згортка матриць

Операції згортки можливі для повноцінного кольорового зображення, що робить необов'язковим попередню підготовку вихідних даних та приведення їх у монохромний вигляд.

Припустимо, що зображення має розмір 32x32x3, тобто це тривимірний масив глибини 3. Будь-який фільтр згортки на цьому шарі повинен мати глибину, що дорівнює глибині введення (наприклад, 3x3x3 або 5x5x3, або 7x7x3). Для кольорового зображення при вибраному фільтрі 3x3x3 ядро згортки буде кубом замість квадрата.

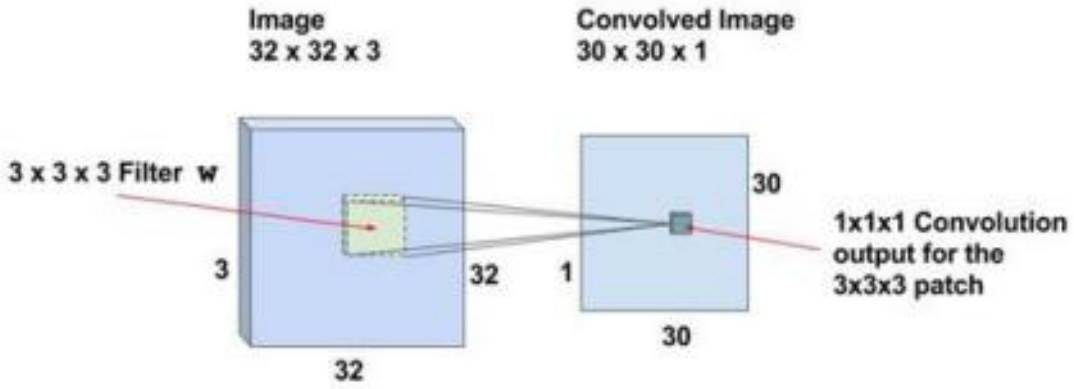


Рис. 38. 3D згортка матриць

Якщо можна виконати операцію згортки, зсунувши фільтр 3x3x3 на все зображення розміром 32x32x3, ми отримаємо зображення з роздільною здатністю 30x30x1. Для вхідного зображення 32x32x3 та розміру фільтра 3x3x3 є 30x30x1 розташування, і для кожного розташування існує нейрон. Тоді виходи 30x30x1 чи активації всіх нейронів називаються картами активації. Карта активації одного рівня є входом для наступного шару.

На відміну від традиційних нейронних мереж, де ваги та зміщення нейронів незалежні один від одного, у разі згорткових нейронних мереж нейрони, що відповідають одному фільтру в шарі, мають однакові ваги та зміщення.

Як правило, використовують більше одного фільтра в одному шарі згортки. Якщо ми використовуємо 32 фільтри, у нас буде карта активації розміром 30x30x32.

Зверніть увагу, що всі нейрони, пов'язані з тим самим фільтром, мають однакові ваги і зміщення. Таким чином, кількість терезів при використанні 32 фільтрів – це 3x3x3x32 = 288, а кількість зсувів – 32.

Очевидно, після кожної згортки результат зменшується за розміром. Для зручності стандартна практика полягає в тому, щоб накладати нулі на межу вхідного шару таким чином, щоб вихід був такого розміру, як і вхідний. Якщо додати доповнення розміром 1 з обох боків від вхідного шару, розмір вихідного рівня буде 32x32x32, що спростить реалізацію.

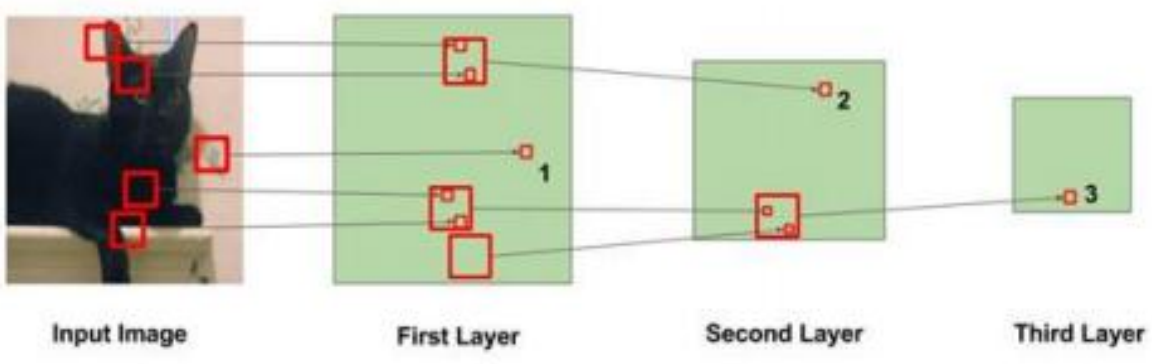


Рис. 39. Сегментація вихідного зображения за допомогою згортки

На рис. 38 великі квадрати вказують область, де виконується операція згортки, а малі квадрати вказують вихід операції, яка є просто числом. З приводу загального алгоритму роботи згорткових мереж можна сказати, що початкові шари аналізують дрібніші області зображення і, отже, можуть виявляти лише прості ознаки, такі як краї, кути тощо. У міру додавання шарів, нейрони отримують інформацію з більших частин зображення та різних інших нейронів. Таким чином, нейрони на пізніших шарах можуть вивчити більш складні функції, такі як очі, ноги і т.д.

З метою зменшення часу навчання за рахунок зменшення кількості параметрів застосовують шар пулінга, який в основному використовується відразу після згорткового шару для зменшення просторового розміру (тільки за шириною та висотою, а не за глибиною). Також використання

меншої кількості параметрів дозволяє уникнути перенавчання. Перенавчання – це умова, коли навчена модель відмінно працює з даними навчання, але не дуже добре працює у тестових даних.

Найбільш поширеною формою пулінгу є максимальний пулінг, в якому ми беремо фільтр розміру та застосовуємо максимальну операцію max з певною частиною зображення.

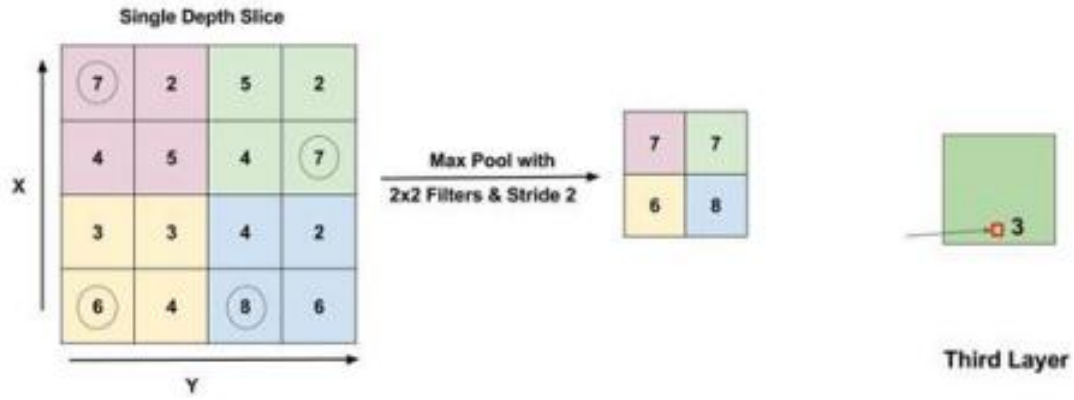


Рис. 40. Максимальний пул з розміром фільтру 2×2

Вихід являє собою максимальне значення області 2×2, показаної з використанням оточених цифр. Найбільш поширена операція пулінгу виконується з фільтром розміром 2×2 з кроком 2. Це суттєво зменшує розмір введення на половину.

Підготовка даних. У поточному розділі розгляд мереж згортання відбувається за допомогою бібліотеки Keras, яка є високорівневою надбудовою над фреймворком TensorFlow і прискорює процес проектування за рахунок спрощення або приховування від розробника низки операцій. Набір даних CIFAR10 постачається разом із Keras.

Він має 50 000 навчальних образів та 10000 тестових зображень 10 класів, таких як літаки, автомобілі, птахи, кішки, олені, собаки, жаби, коні, кораблі та вантажівки. За допомогою наступного програмного коду можна здійснити завантаження та підготовку даних CIFAR10 для подальшої обробки за допомогою нейронних мереж.

Псевдокод цього алгоритму представлений нижче:

1. Підключення бібліотек, що реалізують можливість проектувати архітектуру нейронної мережі.

2. Визначення параметрів мережі, таких як розмір пакету, кількість класів для навчання, тривалість досвіду в епохах, а також визначення файлу для зберігання ваги мережі.

3. Завантаження навчального набору даних із супутнім розбиттям на навчальну та тестову вибірки. Перетворення на категоріальний тип величин, що є міткою класів.

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os
batch_size = 32
num_classes = 10
epochs = 100
num_predictions = 20
save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'keras_cifar10_trained_model.h5'
# Разделяем данные на обучающий и тестовый наборы:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
# Преобразование векторов классов в двоичные матрицы
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Згорткова нейронна мережа складатиметься зі згорткових шарів і шарів MaxPooling. Також буде включений Dropout шар для уникнення перенавчання. На виході мережі буде повнозв'язковий шар (Dense), за яким слідує шар softmax. Тут наведено програмний код створення структури моделі. У наведеному вище коді використовується 6 згорткових шарів та 1 повнозв'язковий шар. Спочатку в модель додаємо згорткові шари з 32 фільтрами з розміром вікна 3×3, далі слід додати згортковий шар з 64 фільтрами. За кожним шаром доданий шар максимального пулінгу з вікном розміром 2×2. Також додані шари Dropout з коефіцієнтами 0,25 та 0.5 для того, щоб не відбулося перенавчання мережі. Наприкінці шар Dense, що виконує класифікацію серед 10 класів з використанням функції активації softmax.

```
model = Sequential()
model.add(Conv2D(32,(3,3),padding='same', input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
```

```
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()
```

Оскільки це проблема класифікації за 10 класами, то процес навчання полягатиме у використанні оптимізатора RMSProp, який оптимізуватиме категоричну втрату ентропії.

```
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accura
cy'])

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

model.fit(x_train,  y_train,  batch_size=batch_size,  epochs=epochs,
validation_data=(x_test, y_test), shuffle=True)

if not os.path.isdir(save_dir):
  os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)

scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

Самостійна робота. На тестовій вибірці отримати певну точність передбачення, пояснити, чому точність вийшла більшою (меншою), ніж точність на тренувальній вибірці. Спробувати різні значення гіперпараметрів (наприклад, швидкість навчання).

Література до розділу 15

1. Гафаров Ф. М. Штучні нейронні мережі та додатки: навч. посібник / Ф. М. Гафаров, А. Ф. Галимянів. – Казань: Вид-во Казан. ун-ту, 2018. – 121 с.

2. Keras manual https://keras.io/api/

3. CNN and Keras for image classification https://learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/

ВИСНОВКИ ДО ЧАСТИНИ III

У практикумі показані підходи до проектування нейронних мереж різної складності та різної топології, починаючи від найпростіших із застосуванням бібліотек для операцій над матрицями, закінчуючи комплексними рішеннями на базі обчислення тензорів. Показано, що стандартні алгоритми навчання на базі алгоритму градієнтного спуску можна і потрібно оптимізувати із застосуванням різних модифікацій шуканого алгоритму.

Сучасний стан технології дозволяє знайти для будь-якого поставленого завдання відповідну архітектуру нейронної мережі, яка в досить великій кількості випадків буде в навченому вигляді приносити результат і користь. Застосування спеціалізованих і оптимізованих фреймворків дає очевидне прискорення у створенні робочого прототипу нейронної мережі на відміну простого моделювання операцій над матрицями, оскільки більшість внутрішніх операцій заховано від розробника і дозволить йому зосередитися на архітектурних завданнях.

Показано важливість коректно обробити дані, очистити від неліквідних елементів, додатково перетворити на більш зручний формат у разі потреби та визначитися зі способами вирішення, зокрема з архітектурою нейронної мережі та її параметрами.

ПРИКЛАДИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ ДЛЯ САМОСТІЙНОЇ РОБОТИ

1. Штучна нейронна мережа з використанням нечіткої логіки та зв'язок зі штучним інтелектом.

Визначте математичний опис нечітких нейронів. Покажіть, за яким принципом відбувається формування шарів нейронної мережі на основі нечіткої логіки. Чому говорять про єднання експертних систем і нейронних мереж? Який прогрес у створенні нейронних мереж та експертних систем на основі нечіткої логіки спостерігається зараз?

2. Механізми упізнавання. Принцип роботи персептрону. Історія та сучасне використання.

Показати, як нейронна мережа-персептрон формує поняття області зосередження даних про однотипні предмети та явища. Що ви знаєте про системи упізнавання в аеропортах у великих скупченнях людей? Основний зміст цього приладу – впізнавання та ідентифікація. Що для цього потрібно?

3. В чому складність налагодження великих нейронних мереж? Чому нема застережень при піднятті складності нейронних мереж? Чи можна це коректно робити?

Чому ще не зрозуміло, як краще це робити? Як в сучасності налагоджують нейронні мережі, що спілкуються з людьми в мережі Інтернет? Математики кажуть, що підвищення їх складності приведе до генерації одразу багатьох можливих рішень, до втрати однозначності. Чомусь така практика не лякає програмістів, в чому тут справа?

4. Інструменти навчання гігантських нейронних мереж.

Поясніть, в чому сенс системи пілотних дебатних програм Фонду «Відкрите суспільство». Як використовуються технології ШІ і різні лінгвістичні функції, як створюються правила для визначення ефективного заперечення, як робиться пошук фрагментів тексту з документів, які підтверджують запит тощо? Що таке iDebate2? Кажуть, це база даних спірних тем. Як це?

5. Поясніть, що це за програми – Machine reading comprehension (MRC), Machine listening comprehension (MLC), Automatic speech recognition (ASR), Question answering (QA).

Може, це розумне машинне читання, машинне розуміння, авторозпізнавання мови тощо. В якому стані розробки цих програм та підходів?

6. Механізм машинного перекладу із застосуванням рекурентних мереж. Показати алгоритм розв'язання задачі машинного перекладу з однієї мови іншою. Відмінності у застосуванні архітектур LSTM і GRU для цієї задачі.

7. Поясніть, чому нейронні мережі з урахуванням алгоритмів навчання з підкріпленням показують видатні результати у реальних задачах.
У чому схожість цього підходу з реальним процесом мислення? Чому AlphaGo та подібні алгоритми у протиборстві з людиною перемагають із тотальною перевагою?

8. Поясніть, які проблеми з алгоритмом градієнтного спуску в архітектурах глибоких нейронних мереж.
Пояснити суть проблеми згасання градієнта. Чому певні функції активації в шарі допомагають впоратися із загасанням краще за інших? Як впливає кількість шарів у мережі на ймовірність виникнення такої проблеми? Наведіть приклад інших потенційних проблем та способів їх вирішення.

9. Поясніть, що таке semi-supervised навчання.
Навести приклади завдань, які можуть призвести до такого типу навчання. Показати, чому у деяких випадках використання такого підходу може дати більшу точність.

10. Механізм повторного використання навчених мереж нових даних (transfer learning).
Чому навчена на одних даних мережа може в принципі успішно працювати на інших даних при цьому підході? Перерахувати причини успішності даного підходу та показати критерій вибору попередньої архітектури для поточних даних.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

Частина І

1. Скільки входів та виходів у штучного нейрона?
2. Що таке активаційна функція? Що вона повинна робити?
3. Чому навчання мережі називають параметричною оптимізацією?
4. Що таке мережа прямого розповсюдження? Як її ще називають?
5. Навчання з вчителем, що це означає, хто є вчителем?
6. Як можна навчати без вчителя? Чому це називають класифікацією?
7. Поясніть, що таке градієнтний спуск? Про який градієнт йдеться?
8. Що стохастичного в стохастичному градієнтному спуску?
9. Формування кластерів, або нейрона, що виграв, у вихідному шарі мережі. Про що йде мова?
10. Що таке недостатнє вчення?
11. В чому бачать недоліки дуже точного навчання?
12. Поясніть, що таке регуляризація.
13. Що можна сказати про трансфертне навчання?
14. В чому полягає акредитація Кохонена?
15. Які допоміжні мережеві системи ви знаєте?
16. Асоціативна пам’ять. Які прості мережі з таким поняттям вам відомі?

Частина ІІ

1. Що можна сказати про архітектуру рекурентної мережі?
2. Поясніть процедуру звертки у відповідній мережі.
3. Рекурсивні та рекурентні мережі, що їх відрізняє, а що об’єднує?
4. Критерії оцінки мережі.
5. Машинне навчання, глибоке навчання, що їх об’єднує?
6. Для чого вхідні дані переводять у векторну форму?
7. Що за шар, що погружає дані кодування, який стискує дані в мережах, що працюють з природною мовою?
8. Яка роль лексичних фондів та ресурсів? Які лексичні ресурси вам знайомі?
9. Процедури векторизації для набору (корпусу) слів. Про що йдеться?
10. Ембендінгі – навчальні моделі на великих корпусах. Чому вони важливі?
11. Вікіфікація – як вона працює у Вікіпедії?

12. Дистрибутивна гіпотеза мови та аналіз вікна – у контексті, як це використовується у семантиці?
13. Навчання мереж за допомогою графічних процесорів. Поясніть, про що йде мова?
14. Що таке лематизація і стемінг?
15. Поясніть популярний метод векторизації Word2Vec.
16. Які можливості популярної бібліотеки для обробки тексту природною мовою Apache OpenNLP?

Частина III

1. Наскільки реально навчити нейронну мережу середнього масштабу на персональному комп'ютері? Що потрібно мати на рівні ресурсів комп'ютера?
2. Чому Python повсюдно застосовується у завданнях, пов'язаних з навчанням нейронних мереж, хоча він не має високої продуктивності?
3. Які можливі етапи підготовки даних існують під час роботи з кольоровими зображеннями? Чому вони важливі?
4. У чому переваги використання серіалізації даних (наприклад, pickle)?
5. Які існують загальні рекомендації щодо проектування простої багатошарової мережі для завдання класифікації даних?
6. Чому застосування сигмоїдальної функції активації краще за застосування порогової функції?
7. Чому застосування ReLU функції активації краще за застосування сигмоїдальної функції?
8. Що таке тензор та в чому переваги використання фреймворків для тензорних обчислень, таких як Tensorflow чи PyTorch?
9. Що таке one hot encoding? Яку основну проблему вирішує?
10. Що таке згорткові нейронні мережі?
11. Що таке пулінг-шари? Перерахуйте види пулінг-шарів.
12. Що таке фільтри згортки? Який їхній фізичний зміст?
13. Як впливають розміри навчальної вибірки на роботу згорткових мереж?
14. На що впливає параметр швидкості навчання в задачах класифікації за допомогою згорткових нейронних мереж?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна література

1. Петров А. П. О возможностях перцептрона // Известия АН СССР, Техническая кибернетика. – 1964. – №6.
2. Бонгард М. М. Проблемы узнавания. – М.: Физматгиз, 1967.
3. Минский М., Пейперт С. Перцептроны. – М.: Мир, 1971 (https://ru.wikipedia.org/wiki/ Перцептроны).
4. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. – 2-е изд. – М.: Горячая линия - Телеком, 2002. – 382 с.
5. Гольдберг Й. Нейросетевые методы в обработке естественного языка / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2019. – 282 с.: ил.
6. Паттерсон Дж., Гибсон А. Глубокое обучение с точки зрения практика / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2018. – 418 с.: ил
7. Траск Э. Грокаем глубокое обучение. — СПб.: Питер, 2019. — 352 с.: ил. — (Серия ≪Библиотека программиста≫).
8. Аггарвал Чару. Нейронные сети и глубокое обучение: учебный курс / пер. с англ. – СПб.: Диалектика, 2020. – 752 с.
9. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с англ. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.: цв. ил.
10. Черняк Е. Введение в глубокое обучение / пер. с англ. – СПб. : Диалектика, 2020. –192 с.: ил.
11. Саттон Р.С., Барто Э.Г. Обучение с подкреплением. – М. Бином Лаборатория знаний, 2014. – 402 с.
12. Хобсон Лейн, Ханнес Хапке, Коул Ховард. Обработка естественного языка в действии. – СПб.: Питер, 2020. – 576 с.: ил. – (Серия «Для профессионалов»).
13. Микелуччи У. Прикладное глубокое обучение. Подход к пониманию глубоких нейронных сетей на основе метода кейсов / пер. с англ. – СПб.: БХВ-Петербург, 2020. – 368 с.: ил.
14. Искусственный интеллект. Обработка естественного языка, распознавание и синтез речи. Альманах. Moscow Institute of Physics and Technology, Ceo/Cdo Pro-library 179с.
15. Куклин В. М. Особенности развития искусственного интеллекта на современном этапе // Вісник Харківського національного університету імені В.Н. Каразіна, серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». – 2018. – С. 34–40.
16. Куклін В. М. Подання знань і операції над ними: навчальний посібник / В. М. Куклін. – Х.: ХНУ імені В. Н. Каразіна, 2019. – 164 с.
17. Куклин В. М. Представление знаний и операции над ними: учебное пособие / В. М. Куклин. – Х.: ХНУ имени В. Н. Каразина, 2019. – 180 с. (http://dspace.univer.kharkov.ua/handle/123456789/15167)

Допоміжна література

1. Мак-Каллок У. С., Питтс В. Логическое исчисление идей, относящихся к нервной активности. Архивная копия от 27 ноября 2007 на Wayback Machine // Автоматы / Под ред. К. Э. Шеннона и Дж. Маккарти. – М.: Изд-во иностр. лит., 1956. – С. 363–384. (Перевод английской статьи 1943 г.).

2. Hecht-Nielsen R. Kolmogorov's Mapping Neural Network Existence Theorem // IEEE First Annual Int. Conf. on Neural Networks, San Diego, 1987, Vol. 3, pp. 11–13.

3. Колмогоров А. Н. О представлении непрерывных функций нескольких переменных суперпозициями непрерывных функций меньшего числа переменных // Известия АН СССР, 108 (1956), с. 179–182.

4. Lorentz George. Metric entropy, widths, and superpositions of functions // American Mathematical Monthly. 1962, vol. 69, p.469–485.

5. Sprecher D. A. On the structure of continuous functions of several variables //Transactions of the American Mathematical Society1965, vol. 115, p. 340–355.

6. Яновский В. В. Коллективный интеллект. НТК «Институт монокристаллов» НАН Украины – Киев: Наукова думка, 2020.

7. Гущин І.В. Моделювання фізичних процесів із використанням технології CUDA : монографія / І. В. Гущин, В. М. Куклін, О. В. Мішин, О. В. Приймак. – Харків: ХНУ імені В. Н. Каразіна, 2017. – 116 с.

8. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с англ. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.

9. Лейн Х., Хапке Х., Ховард К. Обработка естественного языка в действии. – СПб.: Питер, 2020. — 576 с.: ил. — (Серия «Для профессионалов»).

10. Введение в методы программных решений: учебное пособие / Е. В. Белкин, А. В. Гахов, А. М. Горбань, В. М. Куклин. В. М. Лазурик, А. С. Петренко, М. Ю. Силкин, В. В. Яновский; под ред. проф. В. М. Куклина. – Х.: ХНУ имени В. Н. Каразина, 2011. – 308 с.

11. Гущин І. В, Куклін В. М., О. В. Мішин О. В. Нейроні мережі в сучасну епоху / Вісник Харківського національного університету імені В. Н. Каразіна серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління», вип. 50, 2021, с. 49–58.

Посилання на інформаційні ресурси в мережі Інтернет, відео-лекції, інше методичне забезпечення

На сайті «Відеолекції Харківського університету імені В. Н. Каразіна, є лекція Кукліна В.М.

1. Neural network Как развивался искусственный интеллект (https://www.youtube.com/channel/UCzFWNOQBRA1SysklOhQrPHA)

2. На сайті факультету http://www-csd.univer.kharkov.ua/ є презентація В. М. Кукліна теми «Перспективи використання нейронних мереж». (http://www-csd.univer.kharkov.ua/wp-content/uploads/2017/09/Perspektivi-vikoristannya-nejronnih-merezh.pdf)