МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра системного аналізу та управління

К.С. Хабарлак, Т.А. Желдак

САМОНАВЧАННЯ СКЛАДНИХ СИСТЕМ

Конспект лекцій
для здобувачів ступеня магістра
освітньо-професійної програми «Системний аналіз»
зі спеціальності 124 Системний аналіз

Дніпро
НТУ «ДП»
2024

Хабарлак К.С.

Самонавчання складних систем [Електронний ресурс] : конспект лекцій для здобувачів ступеня магістра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / К.С. Хабарлак, Т.А. Желдак ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2024. – 112 с.

Автори:

Хабарлак К.С., доктор філософії;

Желдак Т.А., канд. техн. наук, доц.

Затверджено науково-методичною комісією зі спеціальності 124 Системний аналіз (протокол № 3 від 10.05.2024) за поданням кафедри системного аналізу та управління (протокол № 5 від 10.05.2024).

У конспекті лекцій з курсу «Самонавчання складних систем» подано теоретичні основи різних типів нейронних мереж, методи їх навчання, наведено способи використання в різних сферах. Здобувач ознайомиться не лише з методами, які вже широко використовуються на практиці, але й передовими науковими здобутками, зокрема генеративним штучним інтелектом.

Відповідальний за випуск завідувач кафедри системного аналізу та управління Т.А. Желдак, канд. техн. наук, доц.

Зміст

Вступ

Складні системи присутні на багатьох підприємствах у сферах медицини, фінансів, біології, техніки, програмування. Ким би ви не працювали, вас будуть оточувати системи штучного інтелекту, що самонавчаються: відшукують невідомі значення, надзвичайно швидко обробляють вхідний сигнал різної природи, роблять прогнози розвитку системи. Сучасний світ вже неможливо уявити без самонавчання складних систем.

У даному конспекті лекції з курсу «Самонавчання складних систем» подано теоретичні основи різних типів нейронних мереж, методи їх навчання, наведено способи використання в різних сферах. У межах курсу будуть розглянуті наступні теми:

1. Штучний інтелект: огляд досягнень. Перцептрон Розенблата.
2. Багатошаровий перцептрон. Метод зворотного поширення помилки.
3. Радіально-базисні мережі. Мережі зустрічного поширення.
4. Метод групового врахування аргументів.
5. Згорткові нейронні мережі.
6. Рекурентні нейронні мережі.
7. Нейронна мережа із довгою короткочасною пам'яттю.
8. Відновлення сигналів за допомогою штучних нейронних мереж.
9. Прикладні задачі машинного навчання.
10. Генеративний штучний інтелект.

Під час вивчення цих тем здобувач ознайомиться не лише з методами, що вже широко використовуються на практиці, але й передовими науковими здобутками, зокрема генеративним штучним інтелектом.

Мета дисципліни – сформувати у здобувачів вищої освіти: 1) практичні навички обробки, аналізу, генерації даних провідними методами на основі нейронних мереж; 2) вміння будувати нейронні мережі, що відповідають задачі, та навчати їх; 3) здобути навички роботи із бібліотеками машинного навчання, зокрема TensorFlow та Keras, та мовою програмування Python для побудови нейронних мереж. Знання та навички, отримані в курсі, будуть корисними для подальшого працевлаштування здобувача.

Завдання курсу:

- навчитися проводити попередню обробку даних різної природи, здійснювати інтерполяцію невідомих значень;
- розуміти та навчитися застосовувати належні архітектури нейронних мереж, налаштовувати їх параметри, навчати їх та оцінювати якість роботи у відповідності до поставленої задачі;
- опанувати згорткові нейронні мережі, рекурентні нейронні мережі, підходи для відновлення та генерації сигналів, RBF-мережі та метод групового врахування аргументів;

– отримати практичні навички роботи з бібліотеками TensorFlow і Keras, мовою програмування Python для задач штучного інтелекту та побудови нейронних мереж різної архітектури.

Дисциплінарні результати навчання:

1. Знати сучасні наукові здобутки, вміти будувати та досліджу-вати архітектури глибоких нейронних мереж для обробки та розпізнавання даних різної природи. Вміти застосовувати різні підходи щодо оцінки якості роботи мережі на даних.
2. Знати сучасні наукові здобутки, вміти будувати та досліджувати нейронні мережі спрямовані на відновлення або генерацію сигналу.
3. Розуміти теоретичні основи та вміти застосовувати алгоритми машинного навчання для обробки неповних, частково визначених або пошкоджених сигналів.
4. Розуміти архітектуру повнозв’язної нейронної мережі, сферу її застосування, метод зворотного поширення помилки для навчання мережі.
5. Знати основні структурні блоки нейронних мереж спрямованих на обробку, розпізнавання та сегментацію зображень. Вміти будувати, навчати та застосовувати такі нейронні мережі.
6. Знати структурні блоки нейронних мереж спрямованих на обробку, класифікацію та генерацію текстів. Вміти оброблювати текст, будувати та навчати такі нейронні мережі.
7. Вміти проводити попередню обробку даних різної природи та інтерполювати невідомі значення.
8. Вміти будувати та навчати модель багатопараметричних даних, досліджувати вплив її параметрів на якість передбачення.

Лекція 1 – Штучний інтелект: огляд досягнень. Перцептрон Розенблата

1.1. Що таке нейронна мережа?

Робота над штучними нейронними мережами, які зазвичай називають просто «нейронними мережами», з самого початку була мотивована визнанням того, що людський мозок обчислює зовсім інакше, ніж звичайний цифровий комп’ютер. Мозок – це дуже складний, нелінійний і паралельний комп’ютер (система обробки інформації). Він має здатність організовувати свої структурні складові, відомі як нейрони, для виконання певних обчислень (наприклад, розпізнавання образів, сприйняття та керування моторикою) у багато разів швидше, ніж найшвидший цифровий комп’ютер, який існує сьогодні. Розглянемо, наприклад, людський зір, який вирішує задачу обробки інформації. Функція зорової системи полягає в тому, щоб забезпечити уявлення про навколишнє середовище і, що більш важливо, надати інформацію, необхідну для взаємодії з навколишнім середовищем. Точніше кажучи, мозок зазвичай виконує задачу перцептивного розпізнавання (наприклад, розпізнавання знайомого обличчя в незнайомій сцені) приблизно за 100–200 мс, тоді як задачу набагато меншої складності займають набагато більше часу на потужному комп’ютері.

Для іншого прикладу розглянемо ехолокатор кажана. Окрім надання інформації про те, наскільки далеко знаходиться ціль (наприклад, літаюча комаха), ехолокатор кажанів передає інформацію про відносну швидкість цілі, розмір цілі, розмір різних елементів цілі та нахил цілі. Складні нейронні обчислення, необхідні для отримання всієї цієї інформації з відлуння цілі, відбуваються в мозку розміром зі сливу. Дійсно, ехолокаційний кажан може переслідувати та захоплювати свою ціль із такою простотою та успіхом, яким позаздрив би інженер радара чи гідролокатора.

Як же тоді мозок людини чи мозок кажана це робить? При народженні мозок уже має значну структуру та здатність вибудовувати власні правила поведінки через те, що ми зазвичай називаємо «досвідом». Дійсно, досвід накопичується з часом, причому більша частина розвитку (тобто жорсткого зв’язку) людського мозку відбувається протягом перших двох років від народження, але розвиток продовжується і після цього етапу.

Нервова система, що «розвивається», є синонімом пластичного мозку: пластичність дозволяє нервовій системі, що розвивається, адаптуватися до навколишнього середовища. Подібно до того, як пластичність є важливою для функціонування нейронів як одиниць обробки інформації в мозку людини, так само це стосується нейронних мереж, що складаються зі штучних нейронів. У найзагальнішому вигляді нейронна мережа – це машина, яка розроблена для моделювання того, як мозок виконує конкретну задачу чи функцію, що цікавить; мережа зазвичай реалізується за допомогою електронних компонентів або імітується програмним забезпеченням на цифровому комп'ютері. Щоб досягти

високої продуктивності, нейронні мережі використовують масивний взаємозв'язок простих обчислювальних комірок, які називаються «нейронами» або «блоками обробки». Таким чином, можемо запропонувати наступне визначення нейронної мережі, яка розглядається як адаптивна машина:

Нейронна мережа – це масивний паралельний розподілений процесор, що складається з простих процесорів, який має природну схильність зберігати знання, отримані з досвіду, та робити їх доступними для використання.

Він нагадує мозок у двох аспектах:

- Мережа отримує знання зі свого середовища в процесі навчання.
- Сила міжнейронних зв'язків, відома як синаптичні ваги, використовується для зберігання отриманих знань.

Процедура, яка використовується для виконання процесу навчання, називається алгоритмом навчання, функцією якого є модифікація синаптичних ваг мережі впорядкованим чином для досягнення бажаної мети дизайну.

1.2. Початок розвитку нейронних мереж. Перцептрон Розенблата

Початок (1950-1980-ті роки). Початок історії, що охоплює півстоліття, про те, як ми навчилися змушувати комп'ютери навчатися.

Давайте розглянемо найпростіший приклад машинного навчання. Візьміть кілька точок на двовимірному графіку та намалюйте лінію, яка відповідає їм якомога краще. Те, що ви щойно зробили, є узагальненням з кількох прикладів пар вхідних значень x і вихідних значень y до загальної функції, яка може зіставляти будь-яке вхідне значення з вихідним значенням. Ця задача відома як лінійна регресія (рис. 1.1.), і це чудова 200-річна техніка для екстраполяції загальної функції з деякого набору пар входів-виходів. І ось чому мати таку техніку чудово: існує незліченна кількість функцій, для яких важко розробити рівняння безпосередньо, але для яких легко зібрати приклади вхідних і вихідних пар у реальному світі – наприклад, функція, що відображає введення записаного звуку вимовленого слова на вихід того, чим це вимовлене слово є.

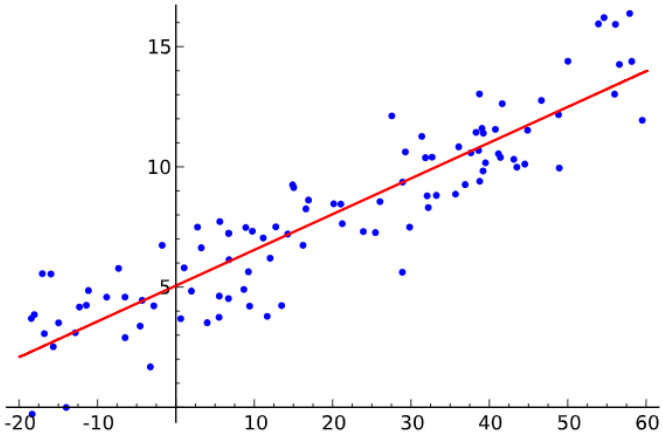


Рис. 1.1. Найпростіша задача машинного навчання.

Лінійна регресія є надто слабкою технікою, щоб вирішити проблему розпізнавання мовлення, але щойно нами було розглянуто суть контрольованого машинного навчання: «навчання» функції за допомогою навчального набору прикладів, де кожен приклад є парою вхід і вихід із функції. Зокрема, методи машинного навчання повинні виводити функцію, яка добре узагальнює вхідні дані, які не входять до навчального набору. В такому випадку ми можемо застосувати вивчену функцію до вхідних даних, для яких у нас немає виходу.

Цей принцип узагальнення настільки важливий, що майже завжди існує тестовий набір даних (додаткові приклади входів і виходів), який не є частиною навчального набору. Окремий набір можна використовувати для оцінки ефективності техніки машинного навчання, подивившись, для якої кількості вхідних прикладів метод правильно обчислює виходи. Ворогом узагальнення є перенавчання – вивчення функції, яка дуже добре працює для навчального набору, але погано на тестовому наборі. Оскільки дослідникам машинного навчання потрібні були засоби для порівняння ефективності їхніх методів, з часом з'явилися стандартні набори даних для навчання та тестування, які можна було використовувати для оцінки алгоритмів машинного навчання.

Узагальнюючи вищесказане, наша вправа на малювання ліній є дуже простим прикладом контрольованого машинного навчання: точки – це навчальний набір (X – це вхід, а Y – вихід), лінія – це апроксимована функція, і ми можемо використовувати лінію, щоб знайти значення Y для значень X, які не відповідають жодній із нам відомих точок.

Який зв'язок із лінійною регресією, якщо тут йдеться про нейронні мережі? Що ж, насправді лінійна регресія має певну схожість із першою ідеєю, задуманою саме як метод, який змушує машини навчатися: перцептрон Френка Розенблата.

Психолог Розенблат задумав перцептрон як спрощену математичну модель роботи нейронів у нашому мозку: він приймає набір бінарних вхідних сигналів (прилеглих нейронів), множить кожен вхідний сигнал на безперервну вагу (сила синапсу для кожного найближчого нейрона) та визначає порогові значення суми цих зважених вхідних даних: якщо сума достатньо велика – вихід нейрона 1, інакше – нуль (таким же чином нейрони або спрацьовують, або ні). Більшість вхідних даних для перцептрона є або деякими даними, або виходом іншого перцептрона. Також перцептрони мають один спеціальний вхід – так зване «зміщення», який має лише значення 1 та певну вагу. Схематичне зображення перцептрона Розенблата показано на рис. 1.2.

Перцептрон Розенблата побудовано на основі роботи Воррена МакКалоха та Уолтера Піттса, які показали, що модель нейрона, яка підсумовує двійкові входи та виводить 1, якщо сума перевищує певне порогове значення, а в інших випадках виводить 0, може моделювати основні логічні функції АБО/І/НІ. На початку розвитку штучного інтелекту це було великою справою – на той час панувала думка, що зробити комп'ютери здатними виконувати формальне логічне мислення по суті вирішить проблему штучного інтелекту (ШІ).

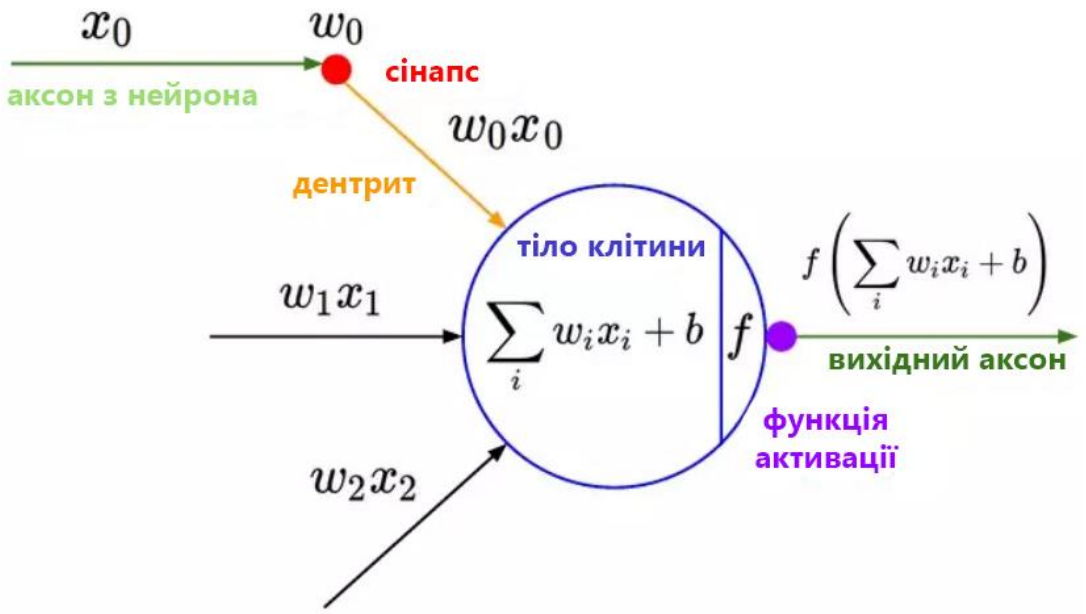


Рис. 1.2. Схема штучного перцептрону Розенблата.

Однак моделі Маккалоха-Піттса бракувало механізму навчання, який був вирішальним для того, щоб її можна було використовувати для ШІ. Ось де перцептрон відзначився – Розенблат винайшов спосіб змусити такі штучні нейрони навчатися, натхненний основоположною роботою Дональда Гебба. Гебб висунув несподівану та надзвичайно впливову ідею про те, що знання та навчання відбуваються в мозку головним чином через утворення та зміну синапсів між нейронами – коротко викладене як правило Гебба: *Коли аксон клітини А знаходиться достатньо близько, щоб викликати збудження клітини В, і багатократно або постійно бере участь у її активації, в одній або обох клітинах відбувається певний процес росту або метаболічна зміна, так що ефективність А, як однієї з клітин, що запускає В, збільшується.*

Перцептрон не дотримувався цієї ідеї в точності, але наявність вагових коефіцієнтів на вхідних даних дозволила створити дуже просту та інтуїтивно зрозумілу схему навчання: маючи навчальний набір прикладів із входами та виходами, перцептрон повинен «вивчити» функцію. Отже, давайте розглянемо алгоритм навчання перцептрона, зображеного на рис. 1.2.

Алгоритм навчання перцептрона:

1. Провести випадкову ініціалізація ваг перцептрона.
2. Для вхідних даних в навчальному наборі обчислити вихід перцептрона.
3. Якщо вихід перцептрона не збігається з істинним виходом:
    a. Якщо істинний вихід 0, а вихід перцептрона 1, **зменшити** ваги, що відповідають виходу 1.
    b. Якщо істинний вихід 1, а вихід перцептрона 0, **збільшити** ваги, що відповідають виходу 1.

4. Перейти до наступного тренувального прикладу та повторювати кроки 2-4 до тих пір, поки перцептрон більше не робить помилок.

Отже маємо лінійну функцію (зважену суму), подібну до лінійної регресії, із нелінійною функцією активації (порогове значення суми). Однак, на відміну від лінійної регресії задача полягає не генерації неперервного виходу, а в бінарній класифікації, коли виходів лише 2: істина/1, брехня/0.

В 50-ті роки ще не було комп'ютерів, подібних до сучасних, тому Розенблат реалізував ідею перцептрона у спеціальному апаратному забезпеченні (рис. 1.3) і показав, що його можна використовувати, щоб навчитися правильно класифікувати прості фігури за допомогою вхідних даних розміром 20×20 пікселів. І так народилося машинне навчання – був створений комп'ютер, який міг наближено виконувати функцію за відомими парами вхідних і вихідних даних. У цьому випадку він навчився невеликій іграшковій функції, але було неважко уявити корисні програми, такі як розпізнавання тексту.

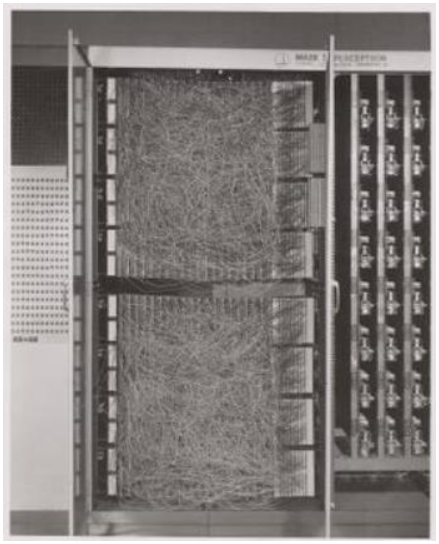


Рис. 1.3. Апаратна реалізація перцептрона Розенблата.

Однак, на даному етапі ми бачили лише, як один перцептрон здатний навчитися виводити одиницю чи нуль – як це можна розширити, щоб працювати з задачею класифікації з багатьма категоріями, такими як людський почерк (у якому є багато літер і цифр)? Така задача виявляється непідсильною для одного перцептрона, оскільки він має лише один вихід.

Функції з декількома виходами можна вивчити, маючи кілька перцептронів, що формують шар (рис. 1.4), так що всі ці перцептрони отримують однакові вхідні дані, і кожен відповідає за один вихід функції. Дійсно, нейронні мережі – це не що інше, як шари перцептронів або нейронів, як їх зазвичай називають сьогодні.

Типовим прикладом використання нейронної мережі є класифікація зображень рукописних цифр. Вхідними даними є пікселі зображення, і є 10

вихідних нейронів, кожен з яких відповідає одному з 10 можливих значень цифр. У цьому випадку лише один із 10 нейронів видає 1, а решта виводять 0.

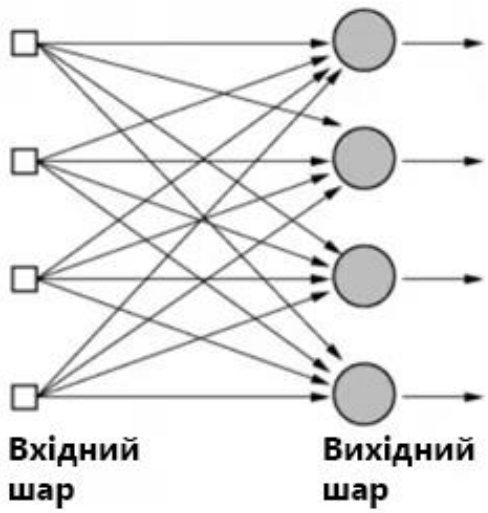


Рис. 1.4. Нейронна мережа із множинними виходами

Також можна уявити нейронні мережі зі штучними нейронами, відмінними від перцептрона. Наприклад, функція активації порогового значення не є абсолютно необхідною. У 1960 році Бернард Уідроу та Тед Хофф відкрили можливість просто виводити вхідну вагу за допомогою «адаптивного нейрона ADALINE», що використовує хімічні «мемістори», і показали, як ці «адаптивні лінійні нейрони» можуть бути включені в електричні схеми за допомогою «мемісторів» – резисторів з пам'яттю. Вони також показали, що відсутність порогової функції активації математично є кращою, тому що механізм навчання нейрона в такому випадку може базуватися на мінімізації помилки за допомогою старого доброго математичного аналізу. Завдяки відсутності різкого порогового стрибка від 0 до 1, міра того, наскільки змінюється помилка при зміні кожної ваги (похідна), може бути використана для зменшення помилки та визначення оптимальної ваги значення. Як ми побачимо, знаходження правильних вагових коефіцієнтів за допомогою похідних помилки навчання щодо кожного вагового коефіцієнта є саме тим, як нейронні мережі зазвичай навчаються донині.

Заглиблюючись в ADALINE трохи більше, прийдемо до наступного розуміння: пошук набору вагових коефіцієнтів для вхідних даних насправді є лише формою лінійної регресії. І знову ж таки, як і з лінійною регресією, цього буде недостатньо для вирішення складних проблем штучного інтелекту, таких як розпізнавання мови або комп’ютерного зору.

Маккалох, Піттс і Розенблат були в захваті від ідеї коннекціонізму – тобто що мережі з таких простих обчислювальних одиниць можуть бути значно потужнішими та вирішувати складні проблеми ШІ. Дослідники того часу передбачали, що незабаром ми отримаємо комп’ютери, що зможуть ходити, говорити, бачити, писати, відтворювати себе та усвідомлювати своє існування.

Подібні передбачення, безсумнівно, дратували інших дослідників штучного інтелекту, багато з яких зосереджувалися на підходах, заснованих на маніпулюванні символами з конкретними правилами, які випливають із математичних законів логіки. Марвін Мінскі, засновник MIT AI Lab, і Сеймур Пеперт, тодішній директор лабораторії, були одними з дослідників, які скептично поставилися до ажіотажу, і в 1969 році опублікували свій скептицизм у формі ретельного аналізу обмежень перцептрона в основоположній книзі під влучною назвою «Перцептрони». Критичний характер аналізу в книзі свідчить про те, що він дійшов висновку, що цей підхід до ШІ був тупиковим. Найбільш широко обговорюваним елементом цього аналізу є з'ясування меж перцептронів – вони не змогли, наприклад, вивчити просту булеву функцію XOR, оскільки вона не є лінійно роздільною. Вважається, що ця публікація допомогла започаткувати першу зиму штучного інтелекту – період після масової хвилі ажіотажу щодо штучного інтелекту, що характеризується розчаруванням, яке спричинило заморожування фінансування та публікацій.

1.3. Відлига зими штучного інтелекту

Отже, важливо відзначити, що аналіз перцептронів, проведений Мінським і Пейпертом, не просто показав неможливість обчислення XOR за допомогою одного перцептрона, але і те, що алгоритм навчання Розенблата не працював для нейронних мереж із декількома шарами перцептронів. Приклад нейронної мережі із декількома шарами зображено на рис. 1.5. Це була справжня проблема: просте правило навчання, описане раніше для перцептрона, не працює для кількох шарів.

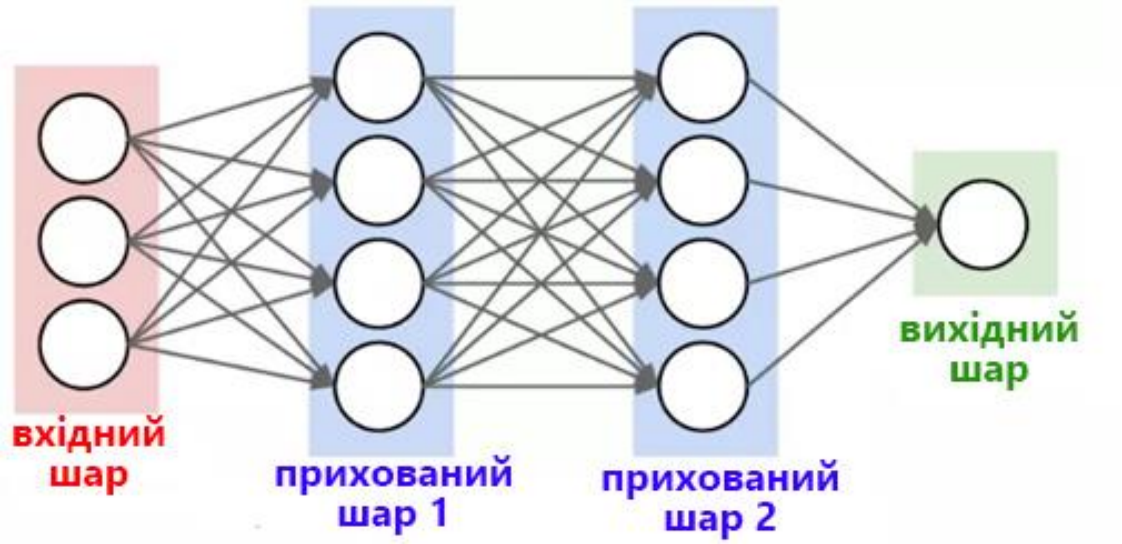


Рис. 1.5. Багатошарова нейронна мережа із прихованими шарами.

Давайте ще раз подивимось на крок 3 оновлення параметрів перцептрона:
3. Якщо вихід перцептрона не збігається з істинним виходом:

a. Якщо істинний вихід 0, а вихід перцептрона 1, **зменшити** ваги, що відповідають виходу 1.
b. Якщо істинний вихід 1, а вихід перцептрона 0, **збільшити** ваги, що відповідають виходу 1.

Причина, чому алгоритм не працює для кількох шарів, має бути інтуїтивно зрозумілою: у прикладі вказано лише правильний вихід для кінцевого вихідного шару, тож як нам знати, як налаштувати ваги перцептронів у шарах до цього? Відповідь, незважаючи на те, що її отримання зайняло деякий час, знову виявилась заснованою на математичному аналізі – на правилі ланцюга. Ключове усвідомлення полягало в тому, що якби нейрони нейронної мережі були не зовсім перцептронами, а створені для обчислення виходу за допомогою функції активації, яка все ще була нелінійною, але також диференційованою, як у випадку з Adaline, то похідну можна було б використовувати не лише для мінімізації помилки останнього шару, але і для всіх попередніх.

Отже, ми можемо використовувати математичний аналіз, щоб розділити вклад в кінцеву помилку кожного шару нейронної мережі та використати техніку оптимізації (як правило, стохастичний градієнтний спуск), щоб знайти оптимальну вагу для мінімізації помилки (рис. 1.6). Підхід обчислення помилки в нейронних мережах шар за шаром отримав назву методу зворотного поширення помилки та був відкритий незалежно декількома науковими колективами в 60-х роках.

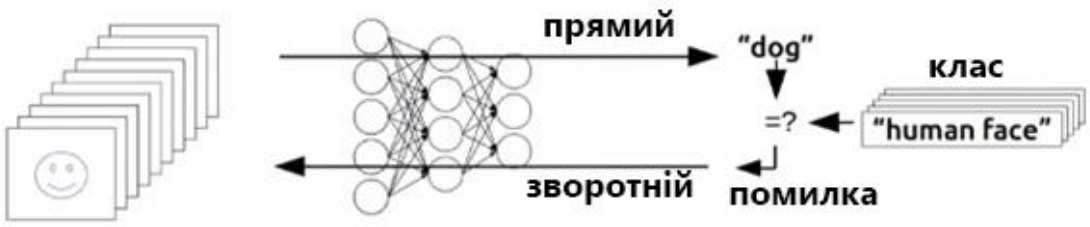


Рис. 1.6. Зворотне поширення помилки.

1.4. Розквіт нейронних мереж

З розкриттям секрету навчання багатошарових нейронних мереж тема знову стала гарячою, і високі амбіції Розенблата, здавалося, були досягнуті. Лише до 1989 року було опубліковано інше ключове відкриття, яке зараз повсюдно цитується в підручниках і лекціях: «Багатошарові мережі прямого зв'язку є універсальними апроксиматорами». По суті, це математично доведення, що кілька шарів дозволяють нейронним мережам теоретично реалізувати будь-яку функцію, і, звичайно, XOR.

Але це математика, де можна уявити безмежну пам'ять і обчислювальну потужність, якщо це знадобиться – чи дозволило зворотне розповсюдження

використовувати нейронні мережі для чогось у реальному світі? О, так. Також у 1989 році Yann LeCun та ін. в лабораторії AT&T Bell Labs продемонстрували дуже важливе реальне застосування зворотного розповсюдження в роботі «Зворотне поширення, застосоване до розпізнавання рукописного поштового індексу». Вам може здатися, що розпізнавання комп'ютером рукописних цифр не є чимось вражаючим, і сьогодні це дійсно так, але до цієї публікації нечіткі та двоякі каракулі виявилися серйозною проблемою для набагато більш охайних розумів комп'ютерів. Приклад розпізнавання показано на рис. 1.7.

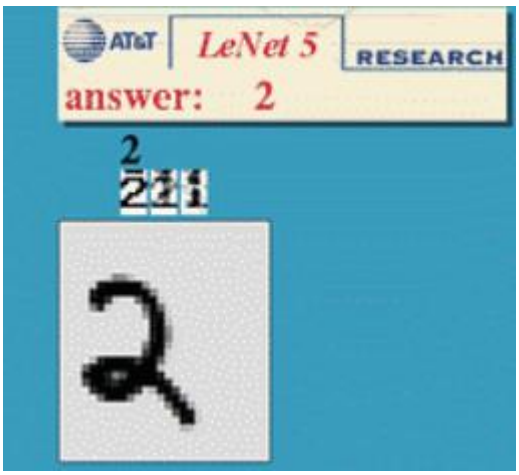


Рис. 1.7. Розпізнавання цифр нейронною мережею LeNet.

Публікація показала, що нейронні мережі цілком спроможні вирішити дану задачу. І, що набагато важливіше, це дослідження підкреслило практичну потребу в ключових модифікаціях нейронних мереж за межі простого зворотного поширення в бік сучасного глибокого навчання.

1.5. Контрольні питання

1. Якою є мотивація розвитку систем, що самонавчаються?
2. В чому полягає наукова новизна перцептрона Розенблата?
3. Неможливість моделювання якої функції спричинило зиму штучного інтелекту? Як дану проблему було вирішено?
4. Яку функцію активації використовує перцептрон Розенблата? В чому її переваги та недоліки?
5. На основі яких досліджень було побудовано перцептрон Розенблата? В чому полягає його подібність до роботи людського мозку?
6. Наведіть кроки алгоритму навчання перцептрона. Чи застосовний цей алгоритм до багатошарових нейронних мереж?
7. В чому особливість адаптивного нейрона ADALINE?
8. Наведіть кілька прикладів задач, що можна вирішити за допомогою штучних нейронних мереж.

Лекція 2 – Багатошаровий перцептрон. Метод зворотного поширення помилки

Останніми роками нейронні мережі стали, безумовно, найважливішою технологією машинного навчання для практичних застосувань, тому ми і присвячуємо цей курс їх вивченню. Попередній розділ вже заклав основи, які нам знадобляться. Зокрема, ми бачили, що моделі лінійної регресії, які містять лінійні комбінації фіксованих нелінійних базисних функцій, можуть бути виражені у вигляді нейронних мереж, що мають один шар параметрів з вагами та зміщенням. Подібним чином моделі класифікації, засновані на лінійних комбінаціях базисних функцій, також можна розглядати як одношарові нейронні мережі. Це дозволило нам представити кілька важливих понять, перш ніж ми приступимо до обговорення більш складних багаторівневих мереж.

Враховуючи достатню кількість відповідним чином вибраних базових функцій, такі лінійні моделі можуть апроксимувати будь-яке задане нелінійне перетворення від входів до виходів з будь-якою бажаною точністю і, отже, можуть здаватися достатніми для будь-якого практичного застосування. Однак ці моделі мають серйозні обмеження, тому почнемо наше обговорення нейронних мереж з вивчення цих обмежень і розуміння того, чому необхідно використовувати базисні функції, які самонавчаються з даних. Це призведе і до обговорення нейронних мереж, які мають більше одного шару параметрів, які можуть навчатись. Вони відомі як мережі прямого зв’язку або багатошарові перцептрони. Також обговоримо переваги наявності багатьох таких шарів обробки, що веде до ключової концепції глибоких нейронних мереж, які зараз домінують у сфері машинного навчання.

2.1. Обмеження функцій з фіксованим базисом

Моделі лінійних базисних функцій для класифікації базуються на лінійних комбінаціях базисних функцій $\phi_j(x)$ і мають вигляд

$$y(x, w) = f\left(\sum_{j=1}^{M} w_j \phi_j(x) + w_0\right) \tag{2.1}$$

де $f(\cdot)$ є нелінійною функцією активації виходу. Лінійні моделі для регресії приймають ту саму форму, але з заміною $f(\cdot)$ на функцію тотожності. Ці моделі допускають довільний набір нелінійних базисних функцій $\{\phi_i(x)\}$, і через загальність цих базисних функцій такі моделі в принципі можуть забезпечити вирішення будь-якої проблеми регресії або класифікації. Це вірно в тривіальному сенсі, оскільки якщо одна з базисних функцій відповідає бажаному перетворенню входу-виводу, тоді навчальний лінійний шар просто повинен скопіювати значення цієї базисної функції на вихід моделі.

Більш загально ми очікуємо, що досить великий і багатий набір базисних функцій дозволить будь-яку бажану функцію наблизити до довільної точності. Таким чином, здавалося б, що такі лінійні моделі становлять структуру загального призначення для вирішення проблем машинного навчання. На жаль, є деякі значні недоліки лінійних моделей, які виникають через припущення, що базисні функції $\phi_j(x)$ є фіксованими та не залежать від навчальних даних.

Прокляття розмірності. Щоб зрозуміти ці обмеження, розглянемо просту модель регресії для однієї вхідної змінної, заданої поліномом порядку M у формі:

$$y(x, w) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M \qquad (2.2)$$

давайте подивимося, що станеться, якщо ми збільшимо кількість входів. Якщо у нас є D вхідних змінних $\{x_1, \ldots, x_D\}$, то загальний поліном з коефіцієнтами до третього порядку набуде вигляду:

$$y(x, w) = w_0 + \sum_{i=1}^{D} w_i x_i + \sum_{i=1}^{D}\sum_{j=1}^{D} w_{ij} x_i x_j + \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{k=1}^{D} w_{ijk} x_i x_j x_k. \qquad (2.3)$$

Зі збільшенням D кількість незалежних коефіцієнтів w зростає $O(D^3)$, тоді як для полінома порядку M зростання кількості коефіцієнтів становить $O(D^M)$. Ми бачимо, що в просторах вищої розмірності поліноми можуть швидко стати громіздкими та мало практичними. Серйозні труднощі, які можуть виникнути в просторах багатьох вимірів, іноді називають прокляттям розмірності. Воно не обмежується поліноміальною регресією та є досить загальним.

2.2. Базисні функції, що залежать від даних

Ми побачили, що прості базисні функції, які вибираються незалежно від розв'язуваної проблеми, можуть зіткнутися зі значними обмеженнями, особливо в просторах високої розмірності. Якщо ми хочемо використовувати базисні функції в таких ситуаціях, то один із підходів полягав би у використанні експертних знань для ручного створення базисних функцій у спосіб, який є специфічним для кожного застосунку. Протягом багатьох років це був основний підхід у машинному навчанні. Базисні функції, які часто називають ознаками, визначатимуться шляхом поєднання знань предметної області та методу проб і помилок. Однак цей підхід мав обмежений успіх і був замінений підходами, керованими даними, у яких базисні функції вивчаються з навчальних даних. Знання предметної області все ще відіграють певну роль у сучасному машинному навчанні, але на більш якісному рівні в розробці архітектур нейронних мереж.

Оскільки дані у високовимірному просторі зазвичай займають невелику частину цього простору, нам не потрібні базисні функції, які щільно заповнюють весь вхідний простір, замість цього ми можемо використовувати базисні функції, які самі пов'язані з даними. Один із способів зробити це – мати одну базисну функцію, пов'язану з кожною точкою даних у навчальному наборі, що гарантує автоматичне пристосування базисних функцій до розташування базових даних.

Прикладом такої моделі є модель радіально-базисних функцій, властивість яких полягає в тому, що кожна базисна функція залежить лише від радіальної відстані (як правило, евклідової) від центрального вектора. Якщо базисними центрами вибрано значення вхідних даних $\{x_n\}$, тоді існує одна базисна функція $\phi_n(x)$ для кожної точки даних, яка, таким чином, охопить весь різновид даних. Типовим вибором для радіальної базисної функції є

$$\phi_n(x) = \exp\left(-\frac{\|x - x_n\|^2}{s^2}\right), \tag{2.4}$$

де s – параметр, що контролює ширину базисної функції. Хоча налаштувати таку модель можна швидко, основна проблема цього методу полягає в тому, що вона стає громіздкою з точки зору обчислень для великих наборів даних. Крім того, модель потребує ретельної регуляризації, щоб уникнути сильного перенавчання.

Зв'язаний підхід, який називається методом опорних векторів або SVM, вирішує зазначений недолік радіально-базисних функцій шляхом повторного визначення базисних функцій, які зосереджені на кожній з точок навчальних даних, а потім автоматичного вибору їх підмножини під час навчання. Як наслідок, ефективна кількість базових функцій у отриманих моделях, як правило, набагато менша, ніж кількість навчальних точок, хоча часто вона все ще є відносно великою та зазвичай збільшується разом із розміром навчальної множини. Метод опорних векторів також не видає ймовірнісних виходів і не узагальнюються більше ніж на два класи. Такі методи, як радіальні базисні функції та метод опорних векторів, в багатьох випадках були замінені глибокими нейронними мережами, які набагато краще використовують дуже великі набори даних. Крім того, як ми побачимо пізніше, нейронні мережі здатні вивчати глибокі ієрархічні представлення, які є вирішальними для досягнення високої точності прогнозування в більш складних програмах.

2.3. Багатошарові мережі

Ми побачили, що для застосування лінійних моделей форми (2.1) до задач, що включають великі набори даних, нам потрібно знайти набір базисних функцій, який налаштований на проблему, що розв'язується. Ключова ідея нейронних мереж полягає в тому, щоб вибрати базисні функції $\phi_j(x)$, які самі мають параметри, які можна вивчати, а потім дозволити коригувати ці параметри разом із коефіцієнтами $\{w_j\}$ під час навчання. Потім проводиться оптимізація всієї модель шляхом мінімізації функції помилки за допомогою градієнтних методів оптимізації, де функція помилки визначається спільно для всіх параметрів у моделі.

Звичайно, існує багато способів побудови параметричних нелінійних базисних функцій. Однією з ключових вимог є те, що вони повинні бути диференційованими функціями своїх параметрів, які можна вивчати, щоб можна було застосувати методи градієнтної оптимізації. Найбільш вдалим вибором було

використання базисних функцій, які мають ту саму форму, що й (2.1), так що кожна базисна функція сама є нелінійною функцією лінійної комбінації вхідних даних, де коефіцієнти лінійної комбінації є параметрами, які можна вивчати. Зауважимо, що цю конструкцію можна рекурсивно розширити, щоб отримати ієрархічну модель з багатьма шарами, яка є основою для глибоких нейронних мереж.

Розглянемо базову модель нейронної мережі, що має два шари параметрів, які можна вивчати. Спочатку будуємо M лінійних комбінацій вхідних змінних $x_1, \dots, x_D$ у формі

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \tag{2.5}$$

де $j = 1, \dots, M$, а верхній індекс (1) вказує на те, що відповідні параметри знаходяться на першому «шарі» мережі. Ми будемо називати параметри $w_{ji}^{(1)}$ вагами, а параметри $w_{j0}^{(1)}$ – зміщеннями (також в літературі зустрічається позначка b), тоді як величини $a_j^{(1)}$ називаються попередніми активаціями. Потім кожна з величин a_j перетворюється за допомогою диференційованої нелінійної функції активації $h(\cdot)$:

$$z_j^{(1)} = h\left(a_j^{(1)}\right), \tag{2.6}$$

які представляють виходи базисних функцій у (2.1). У контексті нейронних мереж ці базисні функції називають прихованими вузлами або прихованими нейронами. В рамках цього курсу ми дослідимо різні варіанти для нелінійної функції $h(\cdot)$, але тут зауважимо, що за умови, що похідну $h'(\cdot)$ можна оцінити, загальна функція мережі буде диференційованою. Відповідно до (2.1), ці значення знову лінійно комбінуються:

$$a_k^{(2)} = \sum_{j=1}^{M} w_{kj}^{(2)} z_j^{(1)} + w_{k0}^{(2)} \tag{2.7}$$

де $k = 1, \dots, K$, а K – загальна кількість виходів. Це перетворення відповідає другому шару мережі, і знову $w_{k0}^{(2)}$ є параметрами зміщення. Нарешті, $\left\{a_k^{(2)}\right\}$ перетворюються за допомогою відповідної функції активації вихідного блоку $f(\cdot)$, щоб отримати набір виходів мережі y_k.

Двошарову нейронну мережу можна представити у вигляді схеми, як показано на рисунку 2.1. Вхідні, приховані та вихідні змінні представлені вузлами, а вагові параметри представлені зв'язками між вузлами. Параметри зміщення позначаються зв'язками, що надходять від додаткових вхідних і прихованих змінних x_0 і z_0, які самі позначаються суцільними вузлами. Стрілки

позначають напрямок потоку інформації через мережу під час прямого поширення.

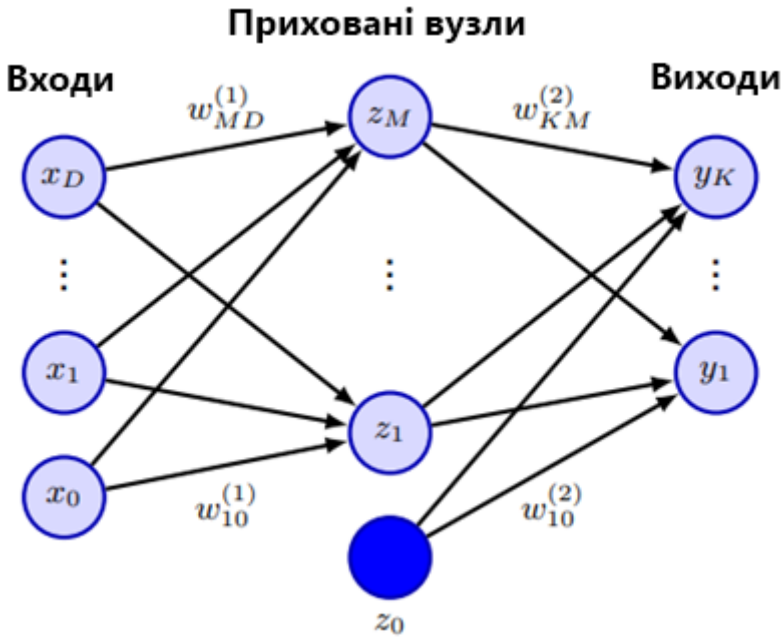


Рис. 2.1. Схема двошарової нейронної мережі.

2.4. Функції активації прихованих нейронів

Для прихованих вузлів єдиною вимогою до функцій активацій є те, що вони мають бути диференційованими, що залишає широкий діапазон можливостей. У більшості випадків усі приховані вузли в мережі матимуть однакову функцію активації, хоча в принципі немає причин, чому б різні варіанти не могли застосовуватися в різних частинах мережі.

Найпростішим варіантом для функції активації прихованих вузлів є функція тотожності, яка означає, що всі приховані нейрони стають лінійними. Однак для будь-якої такої мережі ми завжди можемо знайти еквівалентну мережу без прихованих блоків. Це випливає з того факту, що композиція послідовних лінійних перетворень сама по собі є лінійною трансформацією, і тому її репрезентативна здатність не більша, ніж у одного лінійного шару.

В повнозв’язних мережах найчастіше використовується нелінійна функція активації логістична сигмоїда (часто просто називають «сигмоїда»), зображена на рисунку 2.2, що задається формулою:

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{2.8}$$

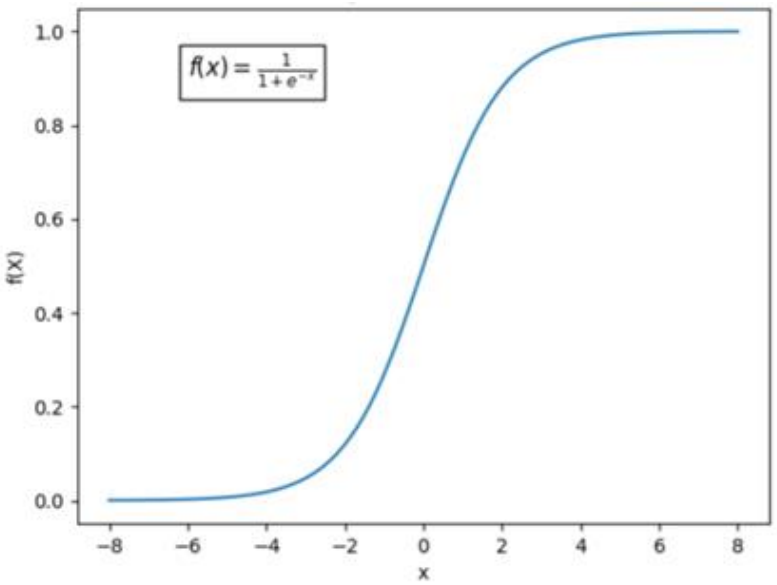


Рис. 2.2. Функція активації сигмоїда.

2.5. Властивості універсальної апроксимації та глибина

Лінійна модель, що відображує ознаки на виходи за допомогою простого множення матриці, за визначенням може представляти лише лінійні функції. Її перевага полягає в легкості навчання, оскільки багато функцій втрат призводять до проблем опуклої оптимізації при застосуванні до лінійних моделей. На жаль, ми часто хочемо вивчати нелінійні функції.

На перший погляд, можемо припустити, що вивчення нелінійної функції потребує проектування спеціалізованого сімейства моделей для типу нелінійності, який ми хочемо вивчити. На щастя, мережі прямого зв'язку з прихованими шарами забезпечують універсальну структуру апроксимації. Зокрема, універсальна теорема про наближення стверджує, що мережа прямого зв’язку з лінійним вихідним шаром і принаймні одним прихованим шаром з будь-якою функцією активації «здавлення» (такою як функція активації сигмоїди) може апроксимувати будь-яку вимірну за Борелем функцію з одного кінцевовимірного простору в інший з будь-якою бажаною ненульовою величиною похибки, за умови, що мережі задано достатньо прихованих нейронів. Похідні прямої мережі також можуть як завгодно добре апроксимувати похідні функції. Концепція борелівської вимірності виходить за рамки цієї лекції; для наших цілей достатньо сказати, що будь-яка неперервна функція на замкнутій і обмеженій підмножині $\mathbb{R}^n$ є вимірною за Борелем і тому може бути апроксимована нейронною мережею. Нейронна мережа також може апроксимувати будь-яке відображення функції з будь-якого кінцевовимірного дискретного простору в інший. У той час як початкові теореми були вперше сформульовані в термінах нейронів з функціями активації, які насичуються як для дуже негативних, так і для дуже позитивних аргументів, теореми універсального

наближення також були доведені для ширшого класу функцій активації, який включає зараз широко використовувану випрямлену лінійну функцію активації.

Універсальна теорема апроксимації означає, що незалежно від того, яку функцію намагаємося вивчити, ми знаємо, що велика нейронна мережа з прихованим шаром зможе представити цю функцію. Однак відсутня гарантія, що навчальний алгоритм зможе вивчити цю функцію. Навіть якщо нейронна мережа здатна представити функцію, навчання може бути невдалим з двох різних причин. По-перше, алгоритм оптимізації, який використовується для навчання, може не знайти значення параметрів, яке відповідає бажаній функції. По-друге, алгоритм навчання може вибрати неправильну функцію через перенавчання. Теорема про відсутність безкоштовного обіду показує, що не існує універсально кращого алгоритму машинного навчання. Мережі прямого зв'язку забезпечують універсальну систему для представлення функцій у тому сенсі, що за заданою функцією існує мережа прямого зв'язку, яка наближає функцію. Немає універсальної процедури для вивчення навчального набору конкретних прикладів і вибору функції, яка буде узагальнюватись на точки, що не входять до навчального набору.

Універсальна теорема про апроксимацію говорить, що існує мережа, достатньо велика для досягнення будь-якого ступеня точності, яку ми бажаємо, але теорема не говорить, наскільки великою буде ця мережа. Баррон надає деякі межі розміру одношарової мережі, необхідної для апроксимації широкого класу функцій. На жаль, у гіршому випадку може знадобитися експоненціальна кількість прихованих нейронів (можливо, по одному прихованому нейрону на кожну вхідну конфігурацію, яку потрібно розрізнити). Це найлегше побачити у двійковому випадку: кількість можливих двійкових функцій на векторах $v \in \{0,1\}^n$ дорівнює 2^{2^n}, і для вибору однієї такої функції потрібно 2^n бітів, що загалом вимагатиме $O(2^n)$ ступенів свободи.

Підводячи підсумок, мережі прямого зв'язку з одним шаром достатньо для представлення будь-якої функції, але шар може бути недосяжно великим і не в змозі правильно навчитися та узагальнити. У багатьох випадках використання глибших моделей може зменшити кількість нейронів, необхідних для представлення бажаної функції, і може зменшити кількість помилок узагальнення.

Існують сімейства функцій, які можуть бути ефективно апроксимовані архітектурою з глибиною, більшою за деяке значення d, але які вимагають набагато більшої моделі, якщо глибина обмежена меншою або рівною d. У багатьох випадках кількість прихованих вузлів, необхідних для неглибокої моделі, експоненціальна за n. Такі результати вперше були доведені для моделей, які не нагадують безперервні, диференційовані нейронні мережі, що використовуються для машинного навчання, але згодом були поширені на і на нейронні мережі. Перші результати були для схем логічних елементів. Пізніші роботи розповсюдили ці результати на лінійні порогові вузли з невід'ємними

вагами, а потім на мережі з безперервними активаціями. Багато сучасних нейронних мереж використовують випрямлені лінійні функції активації. Було продемонстровано, що неглибокі мережі з широким набором неполіноміальних функцій активації, включаючи випрямлені лінійні, мають властивості універсальної апроксимації, але ці результати не стосуються питань глибини чи ефективності – вони вказують лише на те, що достатньо широка мережа з випрямленими лінійними вузлами може представляти будь-яку функцію. Також було показано, що функції, які можна представити за допомогою глибокої мережі з випрямленими лінійними вузлами, можуть вимагати експоненціальної кількості прихованих одиниць з неглибокою (один прихований шар) мережею. Точніше, вони показали, що кусково-лінійні мережі можуть представляти функції з кількістю областей, яка є експоненціальною в глибині мережі.

На рис. 2.3. зображено інтуїтивне геометричне пояснення чому глибока нейронна мережа з випрямленими за модулем вузлами виду:

$$z_j^{(l)} = \left| a_j^{(l)} \right|, \tag{2.9}$$

має перевагу над неглибокою. Зліва направо кожен шар вузлів дозволяє «згортати» простір навколо лінії симетрії заданої функції. На правому рисунку використано лише 2 прихованих вузли. Таким чином, компонуючи ці операції згортання, отримуємо експоненціально велику кількість кусково-лінійних областей, які можуть захоплювати всі типи регулярних (наприклад, повторюваних) візерунків. А, отже, використання одношарової нейронної мережі із функціями активації типу (2.9) вимагало б значно більшої кількості вузлів.

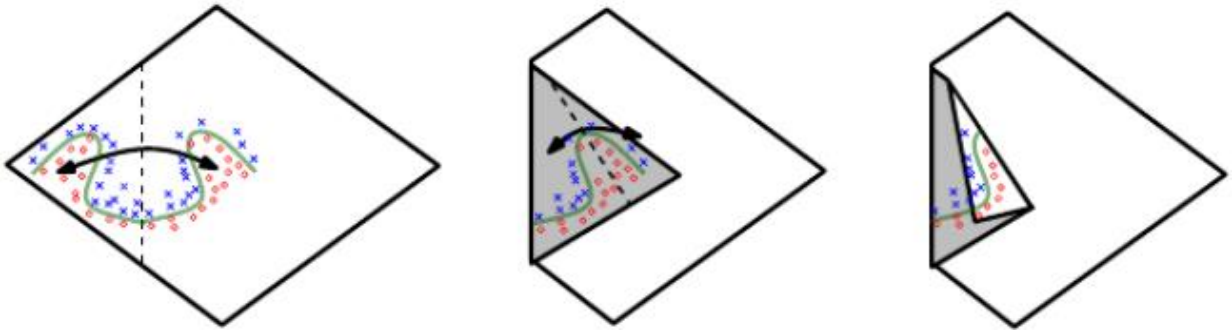


Рис. 2.3. «Згортання» простору багатошаровою нейронною мережею.

2.5. Методи градієнтного спуску

Отже, нами було розглянуто спосіб побудови багатошарової нейронної мережі, типи вузлів, властивості функцій активації. Також, ми дізнались, про універсальну теорему про апроксимацію. Перейдемо до навчання нейронної мережі. Навчання відбувається із використанням методу градієнтного спуску, що задається наступним алгоритмом:

Алгоритм 2.1. Градієнтний спуск.

Вхід:

- навчальний набір даних, що індексується через $n \in \{1, \dots, N\}$;
- кількість навчальних даних B, що оновлюється на одній ітерації алгоритму;
- обчислена функція помилки для поточних даних $E(w)$
- швидкість навчання η;
- інерція навчання μ;
- початковий вектор параметрів w.

Вихід: фінальний вектор параметрів w.

$n \leftarrow 1$
$\Delta w \leftarrow 0$
повторювати
    $\Delta w \leftarrow -\eta \nabla E(w) + \mu \Delta w$
    $w \leftarrow w + \Delta w$
    $n \leftarrow n + B$
    якщо $n > N$ **то**
        перемішати дані
        $n \leftarrow 1$
    кінець якщо
до збіжності
повернути w

Як можна побачити алгоритм є достатньо простим, однак, ми не знаємо як оцінити помилку $E(w)$ для кожного з шарів глибокої нейронної мережі. Для цього використовується метод зворотного поширення помилки.

2.6. Метод зворотного поширення помилки

Наша мета в цьому розділі полягає в тому, щоб знайти ефективну техніку для оцінки градієнта функції помилки $E(w)$ для загальної нейронної мережі прямого зв'язку з довільною топологією, довільними диференційованими нелінійними функціями активації та широким класом функцій помилок. Отримані формули потім будуть проілюстровані за допомогою простої багатошарової мережі, яка має один шар функцій активації типу сигмоїда разом із помилкою суми квадратів.

Багато функцій похибок, що становлять практичний інтерес, містять суму членів, по одному для кожної точки даних у навчальному наборі:

$$E(w) = \sum_{n=1}^{N} E_n(w). \tag{2.10}$$

Розглянемо задачу обчислення $\nabla E_n(w)$ для одного такого члена у функції помилки.

Одношарові мережі. Розглянемо спочатку просту лінійну модель, у якій виходи $\hat{y}_k$ є лінійними комбінаціями вхідних змінних x_i:

$$\hat{y}_k = \sum_i w_{ki} x_i, \tag{2.11}$$

разом із функцією помилки суми квадратів, яка для конкретної точки вхідних даних n набуває вигляду

$$E_n = \frac{1}{2} \sum_k (\hat{y}_{nk} - y_{nk})^2, \tag{2.12}$$

де $\hat{y}_{nk} = \hat{y}_k(x_n, w)$, а y_{nk} – відповідне істине значення. Градієнт цієї функції похибки відносно параметрів w_{ji} визначається як:

$$\frac{\partial E_n}{\partial w_{ji}} = \big(\hat{y}_{nj} - y_{nj}\big) x_{ni}. \tag{2.13}$$

Що можна інтерпретувати як «локальне» обчислення, що включає добуток «сигналу помилки» $\hat{y}_{nj} - y_{nj}$, пов'язаного з виходом зв'язку w_{ji}, і змінної x_{ni}, пов'язаної з входом зв'язку.

Загальна мережа прямого поширення. Загалом, мережа прямого поширення складається з набору вузлів, кожен з яких обчислює зважену суму своїх вхідних даних:

$$a_j = \sum_i w_{ji} z_i, \tag{2.14}$$

де z_i – це або активація іншого вузла, або вузол входу, який надсилає з'єднання до блока j, w_{ji} – вага, пов'язана із цим з'єднанням. Зміщення можна включити в цю суму, ввівши додатковий вузол, або вхід, з активацією, зафіксованою на +1, і тому нам не потрібно мати справу із зміщеннями явно. Сума в (2.14), відома як попередня активація, перетворюється за допомогою нелінійної функції активації $h(\cdot)$, щоб отримати активацію z_j вузла j у формі:

$$z_j = h\big(a_j\big) \tag{2.15}$$

Зауважимо, що одна або більше змінних z_i в сумі (2.14) можуть бути входом, і аналогічно вузол j у (2.15) може бути виходом.

Для кожної точки даних у навчальному наборі будемо припускати, що відповідний вхідний вектор вже надано у мережу та обчислено активації всіх прихованих і вихідних блоків у мережі шляхом послідовного застосування (2.14)

і (2.16). Цей процес називається прямим поширенням, оскільки його можна розглядати як прямий потік інформації через мережу.

Тепер розглянемо оцінку похідної E_n відносно параметрів w_{ji}. Виходи різних блоків залежать від конкретної точки n вхідних даних. Однак, щоб спростити позначення, опустимо індекс n в змінних мережі. Спочатку зауважимо, що E_n залежить від ваги w_{ji} лише через підсумований вхід a_j до вузла j. Тому ми можемо застосувати правило ланцюга для часткових похідних та отримаємо:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}. \tag{2.16}$$

Тепер введемо корисне позначення:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}, \tag{2.17}$$

де δ часто називають помилками з причин, які незабаром побачимо. Використовуючи (2.14), можемо написати:

$$\frac{\partial a_j}{\partial w_{ji}} = z_i. \tag{2.18}$$

Підставляючи (2.17) і (2.18) у (2.16), отримаємо:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i. \tag{2.19}$$

Рівняння (2.19) говорить нам, що необхідна похідна отримується простим множенням значення δ для вузла на виході ваги на значення z для вузла на вході ваги (де $z = 1$ для зміщення). Зауважимо, отримали ту саму форму, що й для простої лінійної моделі в (2.13). Таким чином, щоб оцінити похідні, нам потрібно обчислити лише значення δ_j для кожного прихованого та вихідного вузлів у мережі, а потім застосувати (2.19).

Як ми вже бачили, для вихідних вузлів маємо

$$\delta_k = \hat{y}_k - y_k, \tag{2.20}$$

за умови, що використовується тотожність в якості функцію активації вихідного вузла. Для оцінки δ для прихованих вузлів, знову використаємо правило ланцюга для часткових похідних:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}, \tag{2.21}$$

де сума проходить по всім вузлам k, до яких вузол j має з’єднання.

На рисунку 2.4. показано обчислення δ_j для прихованого шару j методом оберненого розповсюдження помилки з k інших δ, що пов’язані з δ_j зв’язком. Чорна стрілка показує напрямок прямого проходу, червона – зворотного. Зауважимо, що вузли, позначені k, містять інші приховані та/або вихідні вузли.

Записуючи (2.21), використовуємо той факт, що варіації a_j призводять до варіацій функції похибок лише через варіації змінних a_k.

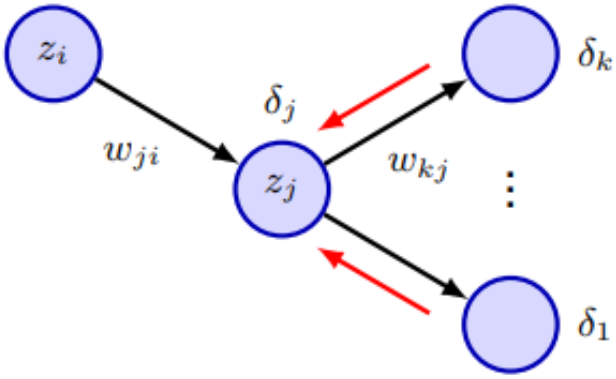


Рис. 2.4. Обчислення δ_j методом зворотного поширення для вузла j.

Якщо тепер замінимо визначення δ_j, надане (2.17), в (2.21) і використаємо (2.14) і (2.15), отримаємо таку **формулу зворотного поширення**:

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k, \qquad (2.22)$$

яка означає, що значення δ для конкретного прихованого блоку можна отримати шляхом поширення δ назад від вузлів вище в мережі, як показано на рисунку 2.4. Зауважимо, що підсумовування в (2.22) береться за перший індекс w_{kj} (що відповідає зворотному поширенню інформації через мережу), тоді як у рівнянні прямого поширення (2.14) воно береться за другим індексом. Оскільки ми вже знаємо значення δ для вихідних вузлів, звідси випливає, що шляхом рекурсивного застосування (2.22), можемо оцінити δ для всіх прихованих вузлів у мережі прямого поширення, незалежно від її топології.

У наведеному вище виведенні було зроблено неявне припущення, що кожен прихований або вихідний вузли мережі мають однакову функцію активації $h(\cdot)$. Однак виведення легко узагальнити для вузлів з індивідуальними функціями активації, просто відстежуючи, який вид $h(\cdot)$ поєднується з яким вузлом.

Процедура зворотного поширення підсумована в алгоритмі 2.2.

Алгоритм 2.2. Зворотне поширення помилки.

Вхід:

- вхідний вектор x_n;
- функція помилки $E_n(w)$ для входу x_n;
- функція активації $h(a)$.

Вихід: функція помилки похідних $\left\{\frac{\partial E_n}{\partial w_{ji}}\right\}$.

Прямий прохід:

для $j \in$ всім прихованим та вихідним вузлам **робити**

$$a_j \leftarrow \sum_i w_{ji} z_i$$

$$z_j \leftarrow h(a_j)$$

кінець циклу

Оцінка помилки:

для $k \in$ всім вихідним вузлам **робити**

$$\delta_k \leftarrow \frac{\partial E_n}{\partial a_k}$$

кінець циклу

Обернене розповсюдження помилки, в зворотному порядку:

для $j \in$ всім прихованим вузлам **робити**

$$\delta_j \leftarrow h'(a_j) \sum_k w_{kj} \delta_k$$

$$\frac{\partial E_n}{\partial w_{ji}} \leftarrow \delta_j z_i$$

кінець циклу

повернути $\left\{\frac{\partial E_n}{\partial w_{ji}}\right\}$.

2.7. Простий приклад

Наведене вище виведення алгоритму зворотного поширення застосовне до довільного виду для функції помилок, функцій активації та топології мережі. Щоб проілюструвати застосування цього алгоритму, розглянемо двошарову мережу форми, зображеної на рисунку 2.1, разом із помилкою суми квадратів. Вихідні вузли мають лінійні функції активації, так що $\hat{y}_k = a_k$, приховані вузли мають функції активації гіперболічний тангенс, задані як:

$$h(a) \equiv \tanh(a), \tag{2.23}$$

Де $\tanh(a)$ визначається формулою:

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}. \tag{2.24}$$

Корисною особливістю цієї функції є те, що її похідну можна виразити в дуже простій формі:

$$h'(a) = 1 - h(a)^2. \tag{2.25}$$

Також розглянемо функцію помилки суми квадратів. Для точки даних n помилка визначається як:

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(\hat{y}_k - y_k)^2. \tag{2.26}$$

де $\hat{y}_k$ – це активація вихідного вузла k, а y_k – відповідне цільове значення для конкретного вхідного вектора x_n.

Для кожної точки даних у навчальному наборі по черзі ми спочатку виконуємо пряме поширення за допомогою:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i, \tag{2.27}$$

$$z_j = \tanh(a_j), \tag{2.28}$$

$$\hat{y}_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j, \tag{2.29}$$

де D – розмірність вхідного вектора x, M – загальна кількість прихованих вузлів. Також $x_0 = z_0 = 1$, щоб параметри зміщення могли бути включені до вагових коефіцієнтів. Далі обчислюємо δ для кожного вихідного вузла за допомогою

$$\delta_k = \hat{y}_k - y_k. \tag{2.30}$$

Потім ми поширюємо ці помилки, щоб отримати δ для прихованих вузлів, використовуючи:

$$\delta_j = \left(1 - z_j^2\right)\sum_{k=1}^{K} w_{kj}^{(2)} \delta_k, \tag{2.31}$$

що випливає з (2.22) і (2.25). Нарешті, маємо похідні щодо ваг першого та другого шарів, що задані:

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j. \tag{2.32}$$

2.8. Контрольні питання

1. Наведіть формулу функцій з фіксованим базисом. В чому полягає так зване «прокляття розмірності»?
2. Як з загальної формули функцій з фіксованим базисом можна отримати радіально-базисні функції? А багатошарові нейронні мережі?
3. Наведіть формулу функції активації сигмоїда. Розрахуйте похідну даної функції активації.
4. Що означає «універсальна апроксимація» повнозв’язних нейронних мереж? Чи знайдеться функція, яку нейронні мережі не зможуть змоделювати?

5. Наведіть алгоритм градієнтного спуску. Застосуйте його для визначення параметрів простої нейронної мережі.
6. Що таке прихований шар нейронної мережі?
7. Як відбувається обчислення градієнта для прихованого шару методом зворотного поширення помилки?
8. Що таке прямий та зворотній прохід нейронної мережі? В якому з проходів обчислюються виходи мережі, а в якому оновлюються параметри?

Лекція 3 – Радіально-базисні мережі. Мережі зустрічного поширення

3.1. RBF-мережі для інтерполяції даних

Мережа RBF (radial-basis function), як і більшість інших нейронних мереж, призначена для апроксимації функцій, які задані в неявному вигляді набором шаблонів (навчальних образів). Теорема Ковера (Cover) говорить про те, що для набору образів у лінійному просторі роздільна здатність апроксиматора підвищується з підвищенням ступеню функції апроксиматора.

Доведено, що ймовірність P того, що певна випадково сформована множина з рівно ймовірним представництвом елементів двох класів кількістю N абсолютно розділяється гіперповерхнею m_1-того порядку, дорівнює

$$P(N, m_1) = \left(\frac{1}{2}\right)^{N-1} \sum_{m=0}^{m_1-1} \binom{N-1}{m}, \tag{3.1}$$

де $\binom{N-1}{m} = \frac{(N-1)!}{m!(N-1-m)!}$ – біноміальні коефіцієнти.

З (3.1) витікають дві основні особливості мереж RBF у порівнянні з багатошаровим перцептроном:

1) прихований шар має бути значно потужнішим за вхідний та вихідний;
2) активаційна функція прихованого шару має бути нелінійною по всіх вимірах з однаковим радіусом σ.

Остання властивість й обумовила назву таких мереж.

Нейронна мережа характеризується такими особливостями: має єдиний прихований шар, нейрони прихованого шару мають нелінійну активаційну функцію, синаптичні ваги всіх нейронів прихованого шару дорівнюють одиниці. Розглянемо наступні позначення:

- $c = (c_1, c_2, \ldots, c_H)$ – вектор координат центрів активаційних функцій нейронів прихованого шару;
- σ_j – ширина вікна активаційної функції j -го нейрона прихованого шару;
- $f(X, c) = e^{\frac{-\sum_{j=1}^{n}(x_j - c_j)^2}{\sigma_j^2}}$ – радіально-симетрична активаційна функція нейрона прихованого шару (див. рис. 3.1);
- w_{ij} – вага зв'язку між i −тим нейроном прихованого шару та j-тим нейроном вихідного шару.

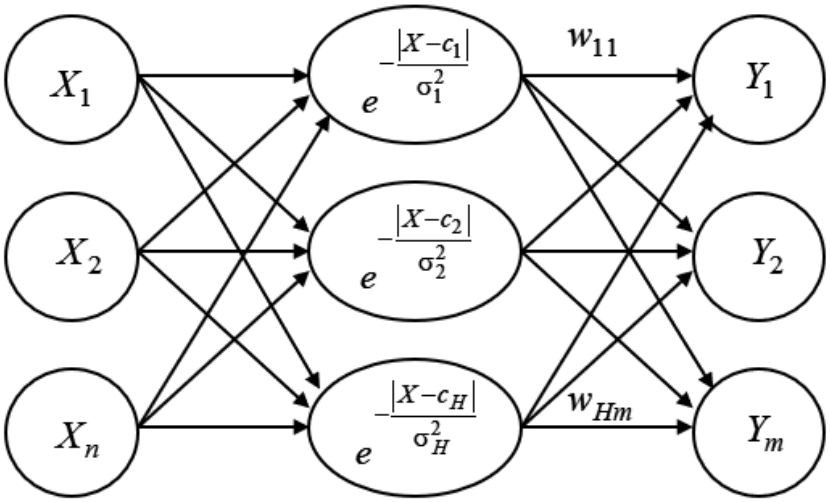


Рис. 3.1. Радіально-базисна нейронна мережа.

Відмінність радіально-базисних мереж від розглянутого раніше багатошарового перцептрона полягає в наявності лише одного прихованого шару, з яким всі нейрони вхідного шару пов'язані рівними одиничними зв'язками. Якщо в перцептроні як прихований шар (яких може бути кілька), так і вихідний є обчислюючими, то в радіально-базисній мережі обчислює лише один прихований шар, вихідний натомість забезпечує лінійну згортку нелінійних функцій. Найголовніша ж відмінність – це аргумент функції активації: в перцептроні він є скалярним добутком вхідного вектора та вектора параметрів, у RBF – відстань від вхідним вектором та центром даного нейрона.

Існує кілька методів навчання радіально-базисної мережі – як ітераційних з рекурсією, так і прямих. Вони можуть використовувати самоорганізацію або вчителя. Далі розглянемо один з методів навчання RBF-мережі, що не містить рекурсії. Алгоритм цього методу, що має назву випадкового вибору фіксованих центрів, є наступним:

Алгоритм 3.1. Алгоритм випадкового вибору фіксованих центрів.

Крок 1. Обрати розмір прихованого шару H рівним кількості тренувальних шаблонів Q . Синаптичні ваги нейронів прихованого шару прийняти рівними 1.

Крок 2. Розмістити центри активаційних функцій нейронів прихованого шару в точках простору вхідних сигналів мережі, які входять до набору навчальних образів $c_j = \overline{X_j}$, для всіх $j = \overline{1, H}$.

Крок 3. Обрати ширини вікон активаційних функцій нейронів прихованого шару σ_j для всіх $j = \overline{1, H}$ достатньо великими, але так, щоб перетин поверхонь, що визначаються активаційним функціями, був мінімальним в просторі вхідних образів.

Крок 4. Визначити ваги нейронів вихідного шару нейронної мережі w_{ij} для усіх $i = \overline{1,H}$, $j = \overline{1,m}$. Для цього пред'явити мережі весь набір навчальних образів. Вихід i-того нейрона прихованого шару для p-того образу буде таким:

$$Y_i = w_{i1}f(X_p, c_1) + w_{i2}f(X_p, c_2) + \ldots + w_{iH}f(X_p, c_H) = \\ = w_{i1}f(X_p, X_1) + w_{i2}f(X_p, X_2) + \ldots + w_{iH}f(X_p, X_H). \quad (3.2)$$

Для відповідності виходу мережі еталонному зразку потрібно, аби $Y_i = D_p$, де D_p – один з елементів набору заданих образів.

Прирівнявши внутрішній вираз з його ідеальним значенням, можемо отримати систему лінійних алгебраїчних рівнянь, яку зручно записати в матричній формі:

$$F \cdot W = D, \quad (3.3)$$

у якому $F = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1H} \\ f_{21} & f_{22} & \cdots & f_{2H} \\ \cdots & \cdots & \cdots & \cdots \\ f_{H1} & f_{H2} & \cdots & f_{HH} \end{pmatrix}$ – інтерполяційна матриця відстаней i −того зразку від j −того центру;

$W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ w_{H1} & w_{H2} & \cdots & w_{Hm} \end{pmatrix}$ – матриця початкових синаптичних вагових коефіцієнтів;

$D = \begin{pmatrix} d_{11} & d_{12} & \cdots & d_{1m} \\ d_{21} & d_{22} & \cdots & d_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ d_{H1} & d_{H2} & \cdots & d_{Hm} \end{pmatrix}$ – матриця заданих образів.

Рішення системи (3.3) досить просто знаходиться на вихідному просторі матричним методом:

$$W = F^{-1}D \quad (3.4)$$

Таким чином, ми одержимо шукані синаптичні вагові коефіцієнти, що забезпечують найкраще проходження інтерполяційної поверхні через навчальні образи в просторі вхідних образів.

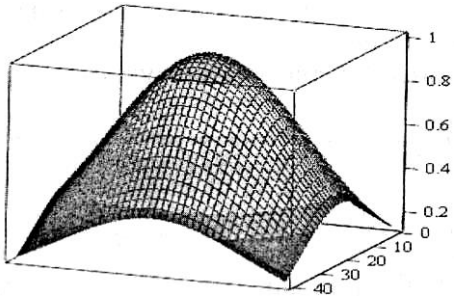


Рис 3.2. Активаційна функція

За відносною простотою побудови й навчання радіально-базисної нейронної мережі ховається цілий ряд її недоліків, які обумовлюють не надто часте застосування цього інструменту в задачах аналізу.

Насамперед, мережа RBF є вкрай чутливою до величини ширини вікон активаційної функції σ. Двомірний аналог активаційної функції зображено на рис. 3.2. Мало того, що для кожного нейрона σ має бути своя, вона ще й має забезпечувати не конфліктність нейронів між собою та, водночас, рівномірність охоплення всього діапазону визначення вхідного масиву даних. Для того аби уникнути «паралічу» мережі через неповну визначеність певних областей вхідних даних і не було великих помилок при апроксимації, вимагаємо виконання наступної умови:

$$-2 < \frac{\sqrt{\sum_{i=1}^{n}\left(x_{ij} - c_i\right)^2}}{\sigma} < 2 \tag{3.5}$$

(вибір границь інтервалу достатньо довільний, але його раціональність підтверджена експериментами) або, обмежившись вибором додатного σ:

$$0 < \frac{\sqrt{\sum_{i=1}^{n}\left(x_{ij} - c_i\right)^2}}{\sigma} < 2. \tag{3.6}$$

Враховуючи, що після нормування $x_{ij}, c_i \in [0; 1]$, одержимо $0 \leq \sum_{i=1}^{n}\left(x_{ij} - c_i\right)^2 \leq n$ для всіх навчальних образів. Тоді $\sigma > \sqrt{n}/2$.

Можна задатися великими значеннями радіусів, але при їх збільшенні зменшується ексцес і ростуть «хвости» графіків активаційної функції, що знову-таки призводить до «паралічу» мережі, але вже в інший бік – неможливості розрізнити образи між собою. Прийнятні результати були отримані в реальних задачах при

$$\frac{\sqrt{n}}{2} < \sigma < \frac{3\sqrt{n}}{2} \tag{3.7}$$

Ще одним недоліком застосування мереж RBF є необхідність виконання певних підготовчих операцій для адекватного навчання і використання навченої мережі RBF. По-перше, треба максимізувати спільну ентропію початкових образів, наприклад, за допомогою відомих методів головних компонент або «вибілювання» входів. Перетворені дані забезпечать якісне та швидке навчання на множині даних мінімальної потужності. Оскільки вказані методи достатньо трудомісткі, необхідно із всієї множини вхідних образів обирати ті, які мають максимальну сумарну попарну евклідову відстань. Наступним кроком має стати нормування.

Неітераційний метод навчання RBF-мережі не завжди є оптимальним методом навчання. Зокрема, коли кількість вхідних образів є великою, застосування градієнтних методів навчання дозволяє зменшити кількість нейронів прихованого шару.

Нарешті, сама мережа RBF має істотне обмеження: якщо багатошарові перцептрони забезпечують глобальну апроксимацію нелінійного відображення, радіально-базисні мережі – лише локальну.

Процес функціонування мережі RBF має ще багато особливостей. Але одну варто згадати: за допомогою такої мережі можна розв'язувати лише задачу інтерполяції.

В той же час мережі RBF мають кілька переваг, зокрема:

- Універсальна апроксимація: мережі RBF можуть апроксимувати будь-яку безперервну функцію з довільною точністю, враховуючи достатню кількість нейронів.
- Швидке навчання: процес навчання, як правило, відбувається швидше порівняно з іншими архітектурами нейронних мереж.
- Проста архітектура: проста трирівнева архітектура робить мережі RBF легшими для реалізації та розуміння.

Застосування мереж RBF:

- Класифікація: Мережі RBF використовуються в задачах розпізнавання образів і класифікації, таких як розпізнавання мови та класифікація зображень.
- Регресія: ці мережі можуть моделювати складні зв’язки в даних для завдань прогнозування.
- Апроксимація функцій: мережі RBF ефективні в апроксимації нелінійних функцій.

3.2. Мережі зустрічного поширення

На відміну від багатошарових перцептронів, мережі зустрічного поширення (МЗП) призначені для початкового швидкого моделювання. Автор МЗП Роберт Хехт-Нільсен вдало об'єднав в одній архітектурі переваги здатності до узагальнення мережі Тейво Кохонена і простоту навчання вихідної зірки Стефана Гроссберга, внаслідок чого мережа МЗП отримала властивості, яких немає у жодної з них окремо. Насамперед МЗП здатна до узагальнення і використовується для розпізнавання та відновлення образів, а також підсилення сигналів.

МЗП (рис. 3.3) працює з векторами, значеннями яких є дійсні величини або бінарні, тобто такі, що складаються з нулів та одиниць. В результаті навчання вхідні вектори асоціюються з вихідними і, якщо мережа є навченою, подача вхідних образів приводить до отримання вихідних.

Правильний вихід може бути отримано і у випадку неповноти або випадкової модифікації вхідного образу. Гіперповерхня, яку одержують в результаті функціонування МЗП, внаслідок принципу неперервності, дає можливість здійснювати прогнозування. Природно, що всередині гіперпаралелепіпеда навчальних образів прогнозування буде більш точним, а при розв'язанні задачі екстраполяції помилка буде значно більшою.

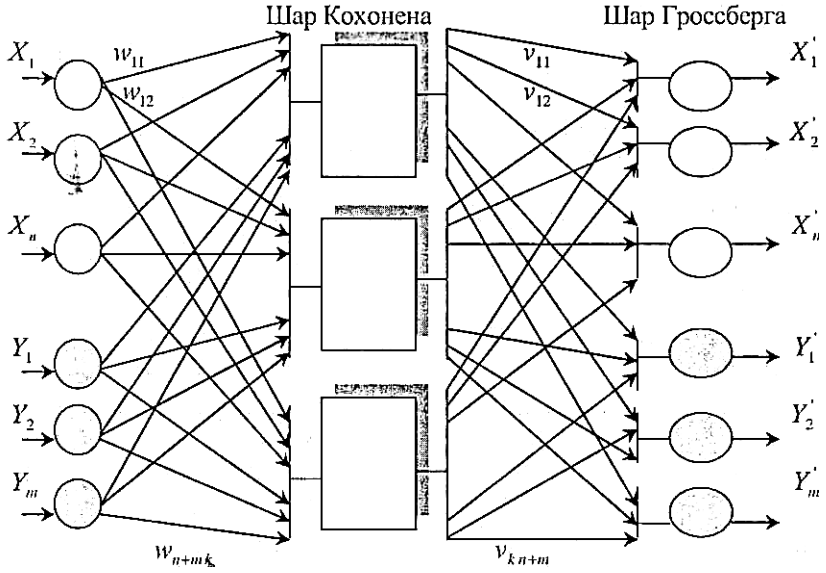


Рис. 3.3. Мережа зустрічного поширення.

Подібно до інших нейронних мереж, МЗП працює в двох режимах: навчання і використання. В першому випадку на входи подаються одночасно вектор X та вектор Y, внаслідок чого відбувається корекція вагових коефіцієнтів. В другому режимі можна подавати на вхід або X, або Y, а на виході одержуємо значення як X, так і Y.

Мережа, у загальному випадку функціонує наступним чином: на вхід подають перший образ X, в кожному нейроні шару Кохонена розраховується

активація $A_j = \sum_{i=1}^{n} w_{ij} x_i = W^T X$. Втім, вихід нейронів не обчислюється за якоюсь гладкою функцією. Використовується турнірний відбір кращого з кандидатів й приймається

$$OUT_p^{Koh} = \begin{cases} 1, & if \quad A_p = \max\limits_j A_j \\ 0 & else \end{cases}. \tag{3.8}$$

Очевидно, що на виході лише одного нейрона шару Кохонена буде «1», решта мають видавати «0». Переможцем, звичайно, буде той нейрон, ваги якого найбільш близькі до вхідного образу. На наступному кроці виконується корекція вагових коефіцієнтів, які інцидентні нейрону, що виграв.

Шар Гроссберга функціонує подібно до шару Кохонена. Активація нейронів його шару $B_j = \sum_{i=1}^{n} v_{ij} OUT_i^{Koh}$. На виході нейронів шару Гроссберга будуть ваги, інцидентні нейрону шару Кохонена, що «виграв», $OUT_j^{Gro} = B_j$.

Далі коригуємо ваги, інцидентні цьому нейрону, аби виходи мережі співпадали (були максимально близькими) до величин у зразку. Цикл завершується і на вхід мережі подається наступний образ.

В результаті багаторазового виконання вказаної процедури вагові коефіцієнти шару Гроссберга повинні співпадати або бути достатньо близькими до вхідних образів. Але все виявляється не так просто.

Розглянемо деякі аспекти **алгоритму функціонування МЗП**. Нехай початкові дані представлені в табл. 3.1 і один вхідний образ відповідає одному рядку цієї таблиці.

Таблиця 3.1. Початкові дані для навчання МЗП.

$\boldsymbol{X_1}$	$\boldsymbol{X_2}$	..	$\boldsymbol{X_n}$	$\boldsymbol{Y_1}$	$\boldsymbol{Y_2}$	..	$\boldsymbol{Y_m}$
x_1^1	x_2^1	..	x_n^1	y_1^1	y_2^1	..	y_m^1
x_1^2	x_2^2	..	x_n^2	y_1^2	y_2^2	..	y_m^2
..	..	..	..	..	..	..	..
x_1^k	x_2^k	..	x_n^k	y_1^k	y_2^k	..	y_m^k

Алгоритм 3.2. Навчання мережі зустрічного поширення.

Крок 1. Нормуємо кожен елемент таблиці в межах рядку. Наприклад, нормований аналог другого елемента першого рядка розраховуємо за формулою

$$\dot{x}_2^1 = \frac{x_2^1}{\sqrt{\sum_{i=1}^{n} \left(x_i^1\right)^2 + \sum_{j=1}^{m} \left(y_j^1\right)^2}} \tag{3.9}$$

і замінюємо в табл. 3.1 кожен елемент на його нормований аналог.

Крок 2. Генеруємо випадковим чином вагові коефіцієнти і обов'язково їх нормуємо, тим самим скорочуючи процес навчання.

Крок 3. Подаємо на вхід мережі рядок матриці $Z = \{X, Y\}$ і розраховуємо скалярні добутки з векторами вагових коефіцієнтів, які зв'язані з усіма нейронами шару Кохонена.

Крок 4. Серед всіх скалярних добутків вибираємо добуток із максимальним значенням і налаштовуємо вагу відповідного нейрона згідно з виразом

$$W_{new} = W_{old} + \alpha(X - W_{old}), \tag{3.10}$$

де W_{old} – попереднє значення ваги, W_{new} – його нове значення, α – коефіцієнт навчання, який спочатку має дорівнювати близько $1/\sqrt{2}$ і поступово зменшуватися в процесі навчання. Вихід нейрона, що «переміг», дорівнює одиниці, всіх інших є нулем.

Крок 5. Вихідний вектор шару Кохонена подається на шар нейронів Гроссберга. В кожному нейроні шару Гроссберга звичайним способом розраховується активація.

Крок 6. Коригуємо всі ваги шару Гроссберга відповідно до виразу:

$$v_{ij}^{new} = v_{ij}^{old} + \beta\left(Y_j - v_{ij}^{old}\right)\chi_i^{koh} \tag{3.11}$$

де Y_j – j-та компонента реального вектора виходу; v_{ij}^{old} – попереднє значення вагового коефіцієнту шару Гроссберга, v_{ij}^{new} – нове значення того ж вагового коефіцієнту, β – швидкість навчання – має той же сенс, що і α; χ_i^{Koh} – ознака нейрона-переможця попереднього шару:

$$\chi_i^{Koh} = \begin{cases} 1, & if \quad OUT_i^{Koh} = 1 \\ 0, & if \quad OUT_i^{Koh} = 0 \end{cases}. \tag{3.12}$$

Крок 7. Якщо показані не всі образи з вихідної таблиці, повернення на крок 3, інакше – далі.

Крок 8. Для усіх навчальних прикладів перевіряється похибка між реальними та отриманими значеннями по Y_j^k. Якщо точність недостатня, зменшують значення α та β, рандомізують рядки таблиці 3.1 і повертаються до кроку 3, інакше – далі.

Крок 9. Закінчення алгоритму. Мережа готова до роботи.

Зробимо низку зауважень щодо алгоритму та його реалізації. Застосування викладеного алгоритму навчання є проблематичним, якщо значна кількість образів утворюють скупчення незначних розмірів в області навчання. У такому випадку результат роботи класичного алгоритму буде невірним і потрібно застосовувати попередню обробку даних.

Деякою оптимізацією запропонованого алгоритму є використання методу опуклої комбінації, в одному з варіантів якого пропонується прирівняти всі вагові коефіцієнти певній величині $w_{ij} = 1/\sqrt{n}$. Крім того, перед подачею на вхід прикладу кожну компоненту вхідного вектора X нормалізувати за виразом

$$\dot{x}_i = \delta x_i + \frac{1-\delta}{\sqrt{n}}. \quad (3.13)$$

Завдяки цьому, на початку навчання, коли $\delta = 0$, вхідні вектори співпадатимуть з ваговими коефіцієнтами. В процесі навчання δ зростає, поступово наближаючись до одиниці. Цей метод приводить до правильного розподілу вхідних векторів і є достатньо ефективним.

В іншому підході пропонується до істинних значень вхідного вектора додавати шум (зміщення). У третьому підході настроюють одночасно всі вагові коефіцієнти певного шару, а не лише нейрона, що «виграв».

Можливим є і такий варіант: якщо один з нейронів шару Кохонена частіше ніж інші стає переможцем, то поріг його спрацьовування збільшується.

Всі розглянуті вище методи відносяться до функціонування мережі в режимі класифікації, тобто віднесення невідомого вхідного образу до одного з попередньо відомих класів. Мережа зустрічного поширення може функціонувати також в режимі інтерполяції. Такий варіант відповідає тому, що настроюються вагові коефіцієнти декількох нейронів, що мають найбільше значення активації. При цьому, виходи нейронів шару Кохонена необхідно нормувати, використовуючи, наприклад, нечітку множину. Тоді значення виходів цих нейронів можна розуміти як функції належності вхідного вектора тому або іншому класу.

Мережа зустрічного поширення рекомендується для попереднього прогнозування або ж для зворотного прорахунку (яким мав би бути X для отримання бажаного Y). У порівнянні із багатошаровим перцептроном мережа МЗП навчається значно швидше, але точність результату є нижчою.

Застосування нейронних мереж в задачах інтерполяції, екстраполяції чи прогнозування є перспективним у випадку існування складних нелінійних залежностей із значеннями факторів, на які не накладені жодні обмеження. Водночас одержання точних результатів можливе лише за умови якісної формалізації задачі, попередньої підготовки даних та забезпечення уникнення надзвичайних ситуацій («паралічу», перенавчання, влучання в локальні оптимуми).

3.3. Контрольні питання

1. Порівняйте архітектури повнозв’язної мережі та радіально-базисної. В чому їх подібності та відмінності?
2. Як саме прихований шар радіально-базисної мережі покриває простір вхідних образів?

3. Які відмінності в роботі повнозв’язної мережі та радіально-базисної в залежності від швидкості та якості?
4. Чи є радіально-базисні функції застосовними для задачі екстраполяції? Чому?
5. Які структурні елементи містить мережа зустрічного поширення?
6. В чому відмінність в «протіканні» сигналу залежних і незалежних змінних між мережами прямого поширення та зустрічного?
7. Яку задачу вирішують мережі зустрічного поширення?
8. Якими є практичні зауваження до «канонічної» реалізації алгоритму навчання мережі зустрічного поширення? Які недоліки даного алгоритму вони виправляють?

Лекція 4 – Метод групового врахування аргументів

Нехай початкові дані зосереджені в матриці $A = (X_1, X_2, \ldots, X_n, Y)$, де X_i, для всіх $i = \overline{1,n}$ та Y – вектори-стовпчики розмірністю N, серед яких X_i – вхідні фактори, а Y – вихідна характеристика. Задача полягає в ідентифікації залежності

$$Y = F(X_1, X_2, \ldots, X_n) \tag{4.1}$$

у вигляді поліномома Колмогорова-Габора у загальному вигляді

$$Y = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{i,j=1}^{n} a_{ij} x_i x_j + \sum_{i,j,k=1}^{n} a_{ijk} x_i x_j x_k + \ldots \tag{4.2}$$

Відомо, що при збільшенні порядку полінома точність наближення ним функції *F(x)* зростає, а потім спадає. Інакше кажучи, нарощуючи максимальний ступінь полінома S, маємо наступний вигляд від цього ступеню середньоквадратичної похибки апроксимації (рис. 4.1). Якщо точність є максимальною, то цей процес закінчується.

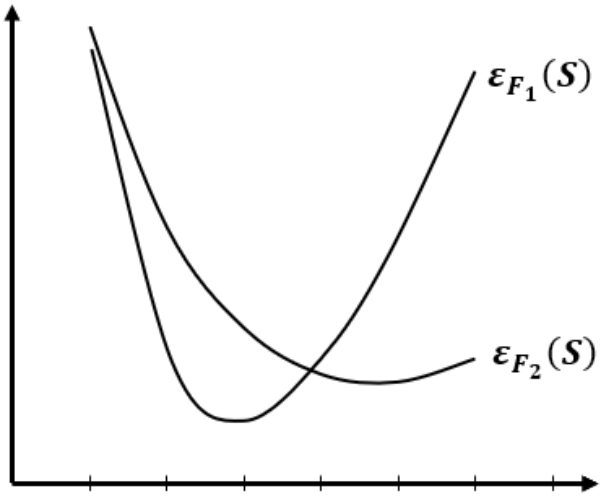


Рис. 4.1. Зміна точності різних моделей МГВА при збільшенні складності.

Особливістю МГВА є те, що він може бути застосований у випадку малої кількості точок експериментів, навіть значно меншої, ніж кількість членів полінома. Це пояснюється тим, що на кожному етапі моделювання апроксимація виконується не за допомогою повного поліному поточної складності, а за допомогою елементарної опорної функції.

Опорна функція обирається на першому етапі реалізації МГВА. Найчастіше в якості опорних використовуються наступні залежності:

1) мультиплікативна

$$y = a_0 + a_1 x_i x_j; \tag{4.3}$$

2) адитивна

$$y = a_0 + a_1 x_i + a_2 x_j\ ; \tag{4.4}$$

3) повна першого порядку

$$y = a_0 + a_1 x_i + a_2 x_j + a_3 x_i x_j; \tag{4.5}$$

4) повна другого порядку

$$y = a_0 + a_1 x_i + a_2 x_j + a_3 x_i^2 + a_4 x_j^2 + a_5 x_i x_j. \quad (4.6)$$

Для першої функції необхідні початкові дані хоча б трьох експериментів, для другої – 4, для третьої – 5, для четвертої – 7. Це пояснюється тим, що для визначення коефіцієнтів буде використано метод найменших квадратів, в якому необхідно мати хоча б одну ступінь свободи, аби отримана залежність мала ступінь довіри, відмінний від 0.

Позначимо $y_k = f(x_i, x_j)$, де f – одна із залежностей (4.3)-(4.6) або, можливо, подібна. На наступному кроці за допомогою МНК визначають коефіцієнти p рівнянь, де (без урахування повторів та діагональних елементів) $p = \frac{n(n-1)}{2}$.

Після того, як усі залежності $y_k = f(x_i, x_j), k = \overline{1, p}$ ідентифіковані за МНК, до справи вступає зовнішній критерій, за яким із усієї множини відбирають найкращі моделі.

Для забезпечення роботи зовнішнього критерію початкову вибірку слід поділити на навчальну та перевірочну. В залежності від обраного критерію якості, навчальна вибірка має містити 50-70% рядків початкової таблиці даних, але мінімум на один рядок більше, ніж порядок опорної функції.

Аби одразу відсіяти неперспективні моделі, у вигляді обмеження застосовується допоміжний критерій точності, відомий з дисперсійного аналізу

$$\delta_k^2 = \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i^k)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}, \quad (4.7)$$

де y_i – табличне значення шуканої залежності в i-тому спостереженні; $\hat{y}_i^k$ – прогнозоване значення в тому ж рядку таблиці за k-тою моделлю; $\bar{y}$ – середнє значення шуканої величини.

Якщо за (4.7) отримано значення $\delta_k^2 < 0{,}5$, кажуть, що модель перспективна й може застосовуватися далі, якщо $0{,}5 \le \delta_k^2 < 0{,}8$ – модель може використовуватись, але обережно, при $0{,}8 \le \delta_k^2 < 1$ використання k-тої моделі небажано, а при $\delta_k^2 \ge 1$ модель не може застосовуватися, оскільки вносить дезінформацію.

Серед моделей, що задовольняють обмеженню за (4.7), визначають Q кращих за критеріями регулярності та незміщеності (про них – далі). Q може бути константою (деякі дослідники встановлюють наперед $Q = n$, аби не мати проблем зі зміною розмірності задачі), зростати чи зменшуватися у функції етапу S.

Ті моделі залежності, які залишилися, перенумеровують і одержують нову таблицю зі значень $y_1, y_2, \ldots, y_Q$. На цьому перший крок селекції закінчено, якщо розрахувати та запам'ятати ефективність найкращої з моделей. На наступному кроці все викладене повторюється: за допомогою МНК визначають коефіцієнти таких самих опорних функцій, але вже від нових змінних

$$z_1 = f(y_1, y_1); z_2 = f(y_1, y_2); \ldots$$
$$z_Q = f(y_1, y_Q); z_{Q+1} = f(y_2, y_2); \ldots \quad (4.8)$$
$$z_p = f(y_n, y_n)$$

Зрозуміло, що процес має циклічний характер і повторюється, допоки значення зовнішнього критерію покращуються. Селекція припиняється, коли її подальше ускладнення неможливе за (4.7) або погіршує значення зовнішнього критерію. Тоді говорять, що оптимальна за складністю модель знайдена й здійснюють зворотній процес розкриття отриманого виразу.

Умови закінчення ітерацій не канонізовані і можуть бути, наприклад, такими:

- середнє значення помилки для наступного ряду селекції є більшим ніж найбільше (середнє) значення помилки для попереднього ряду;
- мінімальне значення помилки наступного ряду більше мінімального значення помилки попереднього ряду;
- максимальне значення помилки наступного ряду більше максимального значення помилки попереднього раду;
- модуль відхилення помилок наступного і попереднього ряду менше деякого числа.

4.1. Критерії регулярності

Опишемо зовнішні критерії, використання яких базується на принципі зовнішнього доповнення. В залежності від типу задачі О.Г. Івахненко запропонував розглядати такі критерії: регулярності, незміщеності та балансу змінних. Відомі два критерії регулярності:

- мінімум середньоквадратичної помилки на нових точках окремої контрольної послідовності;

$$\epsilon = \frac{\sum_{i=1}^{k}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{k} y_i^2} \to min; \quad (4.9)$$

- максимум коефіцієнта кореляції на тих же точках

$$RR = \frac{\sum_{i=1}^{k}(y_i \cdot \hat{y}_i)}{\sum_{i=1}^{k} y_i^2} \to max. \quad (4.10)$$

Розглянемо процедуру їх застосування. Початкові дані знаходяться у таблиці. Розділимо її рядки на дві частини (приблизно 60% на 40%), тоді $N = l + k$, де l – кількість точок експерименту в першій (навчальній) вибірці, k – у другій (контрольній). Значення l, нагадаємо, повинне бути більшим від числа доданків в опорній функції $f(x_i, x_j)$.

Використовуючи елементи навчальної вибірки, визначаємо коефіцієнти тієї з залежностей (4.3)-(4.6), яка прийнята за опорну функцію для усіх p сполучень стовпців, перевіряючи кожну модель за критерієм точності (4.7).

Далі розраховуємо значення критерію регулярності (4.9) чи (4.10) на точках контрольної вибірки, після чого впорядковуємо моделі від кращого значення критерію до гіршого і залишаємо з них певну кількість Q кращих. Після перенумерації вони складуть множину функцій наступного ряду селекції.

Перевагою критеріїв регулярності є плавність зміни їх значення при збільшенні складності моделі. Недоліком їх використання є низька точність при розв'язанні екстраполяційних задач. Тому критерії регулярності раціонально застосовувати для ідентифікації, інтерполяції та короткострокового прогнозу.

4.2. Критерії незміщеності

Відомі три види критерію незміщеності, що може обчислюватись на базі аналізу розв'язків, аналізу коефіцієнтів та як «критерій відносної незміщеності».

Критерій незміщеності, що базується на аналізі розв'язків (КН1). Для розрахунку КНl необхідно ранжувати всі точки експериментів за збільшенням або зменшенням значення дисперсії. Процедура ранжування описана нижче. Після ранжування точки експериментів нумерують і ділять на дві послідовності:

- до першої відносять точки з парними номерами, їх кількість N_1,

- до другої – точки з непарними, їх кількість N_2. Логічно, що $N_1 + N_2 = N$.

На першому ряді селекції перша послідовність є навчальною, друга – контрольною. Отримані на навчальній послідовності N_1 рівняння регресії позначимо $y_k^{'} = f(x_i, x_j)$.

Далі першу послідовність вважають контрольною, другу – навчальною. На навчальній послідовності знаходять рівняння регресії $y_k^{''} = f(x_i, x_j)$. Кількість рівнянь $y_k^{'}$ та $y_k^{''}$ повинна співпадати, випадок невиконання цієї умови не розглядаємо. Для кожного k-того рівняння розраховують середньоквадратичне відхилення

$$KH1_k = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_{k_i}^{'} - y_{k_i}^{''}\right)^2} \to min, \qquad (4.11)$$

де k_i – номер рядка у таблиці значень k-тої моделі. Моделі з найменшим значенням критерію вважаються найточнішими.

Як і за критерієм регулярності, за критерієм КН1 обирають Q найкращих рівнянь, що відповідають меншій оцінці критерію.

Для визначення фактора зупинки алгоритму визначають середнє зважене значення критерію незміщеності моделей даного покоління

$$\overline{KH1}_s = \frac{1}{Q}\sum_{k=1}^{Q} KH1_k \qquad (4.12)$$

На другому і подальших рядах селекції s процедура залишається тією самою. Селекція продовжується доти, доки середнє значення критерію незміщеності зменшується ($\overline{KH1}_s < \overline{KH1}_{s-1}$).

Критерій незміщеності, що базується на аналізі коефіцієнтів (КН2). Точки експериментів ранжуються за величиною дисперсії і поділяються навпіл на навчальну та контрольну послідовності. Точки з більшим значенням дисперсії потрапляють до навчальної послідовності, з меншим – до контрольної. Особливість критерію полягає в тому, що на кожному ряді селекції ранжування і розділення точок експерименту виконується наново. Крім того, зменшується свобода вибору згідно з формулою

$$Q(S) = \alpha n - \beta S, \tag{4.13}$$

де Q – число моделей, що переходять у нове покоління на даному ряді селекції; S – номер ряду; n – кількість вхідних змінних; $\alpha = 1 \dots 5$ (чим більше незалежних змінних, тим менше); $\beta = 0{,}1 \dots 0{,}2$. Формула (4.13) та пов'язана з нею процедура дають можливість швидкого розв'язання задач великої розмірності.

Значення критерію незміщеності оцінок коефіцієнтів моделі розраховується за формулою:

$$\text{КН2}_k = \frac{\sum_{i=1}^{P} \chi(a_i = b_i) \cdot a_i^2}{\sum_{i=1}^{P} a_i^2 + \sum_{i=1}^{P} b_i^2} \to max \tag{4.14}$$

де p – загальне число моделей, a_i – коефіцієнти i-тої моделі, отримані до зміни послідовностей; b_i – коефіцієнти тієї ж моделі, отримані після зміни; $\chi(a_i = b_i)$ – відносна доля співпадінь.

До наступного ряду селекції (в наступне покоління) пропускають Q за (4.13) моделей, що мають більше значення $KH2$.

Як і для $KH1$, розрахунки середнього значення залишених моделей для формування умови загальної зупинки алгоритму справедливі. Селекція продовжується доти, доки середнє значення $\overline{KH2}_s$ у ряді селекції збільшується.

Критерій незміщеності $KH2$ необхідно застосовувати або разом з критерієм регулярності, або в алгоритмах з повним перебором моделей. При відтинанні частини моделей він швидко сходиться до локального оптимуму – найближчої задовільної моделі.

Критерій відносної незміщеності (КН3). У цьому випадку використовують лише лінійні часткові описи (наприклад, $y = a_0 + a_1 x_i + a_2 x_j$). Але щоб уникнути втрати точності, простір початкових аргументів включає також коваріації (наприклад, x_1^2, x_2^2, $x_1 x_2$). Присвоюючи значення коваріацій новим змінним, отримаємо узагальнені аргументи x_i, кількість яких може значно переважати кількість стовпців у початковій таблиці.

Оскільки частковий опис на другому ряді має вигляд $z = a_0 + a_1 y_i + a_2 y_j$, то ортогоналізований частковий опис матиме вигляд

$$z = y_i + A\hat{y}_j \tag{4.15}$$

де $\hat{y}_j$ – вектор, ортогоналізований по відношенню до x_i.

Якщо у часткових описах значення змінних центровані та нормовані за середнім значенням, то в ортогоналізованих часткових описах вільний член $a_0 = 0$, а інші коефіцієнти мають такі значення:

$$\begin{cases} a_1 = 1; \\ a_2 = A = \frac{\sum_{i=1}^{n}(y_i y_j)}{\sum_{i=1}^{N} y_i^2} \end{cases} \quad (4.16)$$

Ортогоналізація – це перетворення $\hat{y}_j$ по відношенню до базового y_i, в результаті якого одержуємо $\sum_{i=1}^{n}(y_i, \hat{y}_j) = 0$. Для цього достатньо знайти такі модельні значення $\hat{y}_j = y_j - Ay_i$, де A визначається за (4.16).

Таким чином, алгоритм МГВА з використанням критерію відносної незміщеності має такі базові кроки:

Крок 1. Початкові дані ділимо на дві частини (описано раніше).

Крок 2. На першій послідовності визначаємо значення коефіцієнтів A' в рівнянні регресії, на іншій – коефіцієнти A''. Кращими вважаємо ті описи, у яких

$$КНЗ_k = \frac{A_k^{'} - A_k^{''}}{A_k} \to \min_k \quad (4.17)$$

Ряди селекції при цьому матимуть вигляд: $y = a_1 x_i$; $z = y_i + A\hat{y}_j$; $t = z_i + A\hat{z}_j$ і так далі до зупинки за глобальним критерієм.

4.3. Критерій балансу змінних

Критерій балансу змінних є найефективнішим при екстраполяції, тобто застосуванні МГВА у середньостроковому та довгостроковому прогнозуванні. Його визначення може виконуватись як емпірично, так і штучно.

При емпіричному визначенні критерію балансу змінних із подальшого розгляду на даному ряді селекції виключаються ті моделі, які дають заздалегідь невірні результати, що лежать за межами можливих значень прогнозованої залежності. Крім того, на значення прогностичних моделей можуть накладатися додаткові умови, наприклад, $\hat{y}_i = f(x_i, x_j) \le y$ (модельне значення не має перевищувати відомого з таблиці) або $\hat{y}_i = f(x_i, x_j) \in [y_{min}; y_{max}]$ (модельне значення не має виходити за певний діапазон).

Штучні умови балансу не є наслідком принципу фізичної реалізації моделі і визначаються дослідником. Найчастіше – це функції, що є комбінацією сум чи різниць вхідних факторів. При цьому, до переліку вхідних факторів X додаються незалежні змінні X', які зазвичай не мають фізичного сенсу.

Наприклад, для задачі побудови екстраполяційної моделі за трьома факторами x_1, x_2 та x_3 можуть бути введені наступні змінні:

$$s_1 = x_1 + x_2 + x_3; s_2 = x_1 + x_2 - x_3; \ldots; s_6 = -x_1 - x_2 - x_3 \quad (4.18)$$

Ці змінні разом з початковими змінними x_1, x_2 та x_3 утворюють вектор вхідних факторів. Розраховані s_i заносяться у таблицю початкових даних та надалі використовуються разом з іншими змінними.

За описаною вище методикою із застосуванням МНК намагаються отримати модель оптимальної складності $Y = F(x_1, x_2, x_3)$. При цьому, на кожному кроці моделі оцінюють не за звичними критеріями незміщенності чи регулярності, а наступним чином: припустимо, що на певному ряду отримані сімейство моделей вигляду

$$\hat{y}^k = f(X) = a_0^k + \sum_{i=1}^{m} a_i^k \cdot \prod_{j=1}^{3} x_j^{b_k}, k = \overline{1, p}. \quad (4.19)$$

Найкращою з усіх p моделей буде вважатися така, яка матиме мінімальне розсіювання на інтервалі спостереження

$$\Phi = \sum_{i=1}^{6} E_i^2 = \sum_{i=1}^{6} \sum_{t=1}^{N} \left(S_{it} - \hat{S}_{it}\right) = \sum_{i=1}^{6} \sum_{t=1}^{N} \left(S_{it} - \hat{S}^k(x_1, x_2, x_3, A, B, m)\right). \quad (4.20)$$

Таким чином, за всіма табличними значеннями відомих X ми обчислюємо найкращу модель для прогнозування Y, обчислюючи на тому ж інтервалі за допоміжними змінними якість моделі.

Вигляд критерію (4.20) та необхідність створення й обчислення додаткових векторів даних вказують на очевидну трудомісткість застосування даного критерію на практиці. Тому при його використанні необхідно застосовувати процедури зменшення кількості комбінацій перебору, у тому числі й враховуючи принципи фізичної реалізації та здорового глузду. Кількість додаткових змінних обирається дослідником і має знаходитися в межах здорового глузду та можливостей обчислювальної машини.

Слід пам’ятати, що критерій балансу змінних за математичною сутністю дуже близький (а в певних задачах ідентичний) критерію мінімуму незміщеності.

4.4. Алгоритм поділу початкової вибірки даних

Реалізація МГВА, як ми показали раніше, пов'язана із необхідністю поділу генеральної сукупності даних на дві вибірки – навчальну та контрольну. Найбільш поширеним, проте не єдиним, є підхід, за яким до навчальної послідовності обирають точки експериментів з більшим значенням дисперсії, а до контрольної – з меншим. Це пояснюється тим, що область навчання повинна бути якнайширшою, а контрольні точки, в більшості своїй, знаходитися всередині неї.

Для практичної реалізації пропонується наступний варіант алгоритму:

Алгоритм 4.1. Поділ початкової вибірки даних.

Крок 1. Визначити відсоткове співвідношення між кількістю елементів у навчальній і контрольній послідовності (від 70% / 30% до 50% / 50%, пов’язано з критерієм ефективності, що буде застосовуватись).

Крок 2. Для кожного стовпчика $X_i, i = \overline{1, n}$ розрахувати середнє значення його елементів

$$\overline{x}_i = \frac{1}{N}\sum_{j=1}^{N} x_{ji}, \tag{4.21}$$

та отримати середнє значення множини образів $(\overline{x}_1, \overline{x}_2, \dots, \overline{x}_n)$.

Крок 3. Знайти вибіркові дисперсії для кожного рядка таблиці за формулою

$$D_i = \frac{1}{n-1}\sum_{j=1}^{n}\left(x_{ij} - \overline{x}_j\right)^2, i = \overline{1, N} \tag{4.22}$$

Крок 4. Для впорядковування таблиці переставити рядки таким чином, щоб першим був рядок з найбільшим значенням дисперсії, а останнім – з найменшим.

Крок 5. У відповідності до результату кроку 4 розділити дані в таблиці на навчальну та контрольну послідовності.

Якщо розв'язується задача короткострокового прогнозу або інтерполяція всередині інтервалу наявних даних, тоді додатково шукають оптимальне співвідношення кількості образів у навчальній та контрольній послідовностях з метою отримання найпростішої та достовірної моделі.

4.5. Контрольні питання

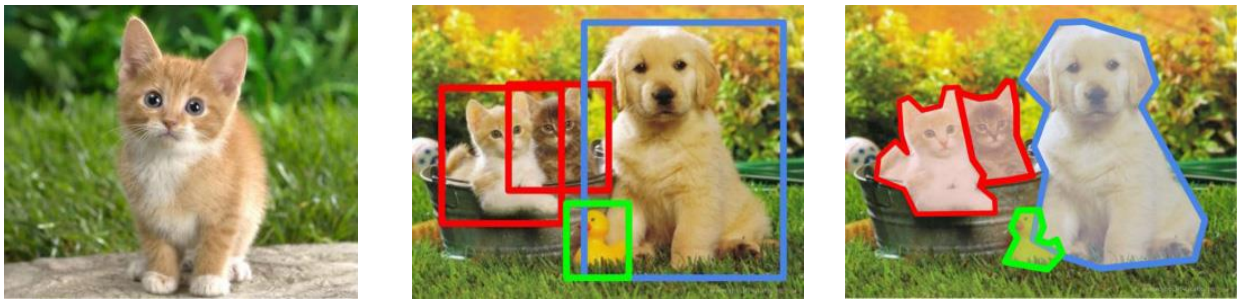
1. В якому вигляді метод групового врахування аргументів апроксимує цільову невідому функцію?
2. В чому перевага МГВА з точки зору кількості навчальних прикладів?
3. Які опорні функції можуть бути використані в МГВА?
4. Які зовнішні критерії направлені на вирішення задачі інтерполяції? Екстраполяції?
5. Як відбувається поділ початкової вибірки даних?
6. Які з зовнішніх критеріїв в ході навчання міняють місцями навчальну та контрольну вибірки?
7. Які відмінності між собою критеріїв незміщенності?
8. Які особливості критерію балансу змінних?

Лекція 5 – Згорткові нейронні мережі

Згорткова нейронна мережа (ЗНМ), також відома як Convolutional Neural Network – це спеціалізований тип алгоритму глибокого навчання, який в основному розроблений для задач, які вимагають розпізнавання об’єктів. ЗНМ використовуються в різних практичних сценаріях, таких як автономні транспортні засоби, системи відеоспостереження та інші.

Згорткові нейронні мережі (ЗНМ) найчастіше використовують для задач комп’ютерного зору. Основними задачами є:

1. Класифікація – для кожного вхідного зображення необхідно зазначити клас, до якого дане зображення належить. Класи зображення є деякою фіксованою наперед заданою множиною. Приклад задачі класифікації зображено на рис. 4.1а.
2. Пошук об’єктів (детекція) – для кожного вхідного зображення необхідно: 1) виділити об’єкти наявні на зображенні прямокутною рамкою; 2) кожному знайденому об’єкту зазначити клас до якого він належить. Див. рис. 4.1б.
3. Сегментація – для кожного вхідного зображення необхідно знайти об’єкти наявні на зображенні, виділити дані об’єкти якомога точніше маскою довільної форми та зазначити клас кожного об’єкту. Зображено на рис. 4.1в.



кіт **кіт, собака, качка** **кіт, собака, качка**

(а) (б) (в)

Рис. 5.1. Види задач комп’ютерного зору: (а) – задача класифікації, (б) – задача пошуку об’єктів, (в) – задача сегментації.

Існує кілька причин, чому ЗНМ важливі в сучасному світі, як зазначено нижче:

- ЗНМ відрізняються від класичних алгоритмів машинного навчання, таких як SVM і дерева рішень, своєю здатністю автономно вилучати ознаки, обходячи потребу в ручному проектуванні ознак і, таким чином, підвищуючи ефективність;
- згорткові шари надають ЗНМ інваріантність до зсуву, дозволяючи їм ідентифікувати та витягувати шаблони та ознаки з даних, незалежно від варіацій у положенні, орієнтації, масштабі чи трансляції;

– різноманітні попередньо навчені архітектури ЗНМ, включаючи VGG-16, ResNet50 і Inceptionv3, продемонстрували найвищу якість вирішення задач комп'ютерного зору. Ці моделі можна адаптувати до нових задач з відносно невеликою кількістю даних за допомогою процесу, відомого як тонке налаштування.

5.1. Натхнення для ЗНМ і паралелі із зоровою системою людини

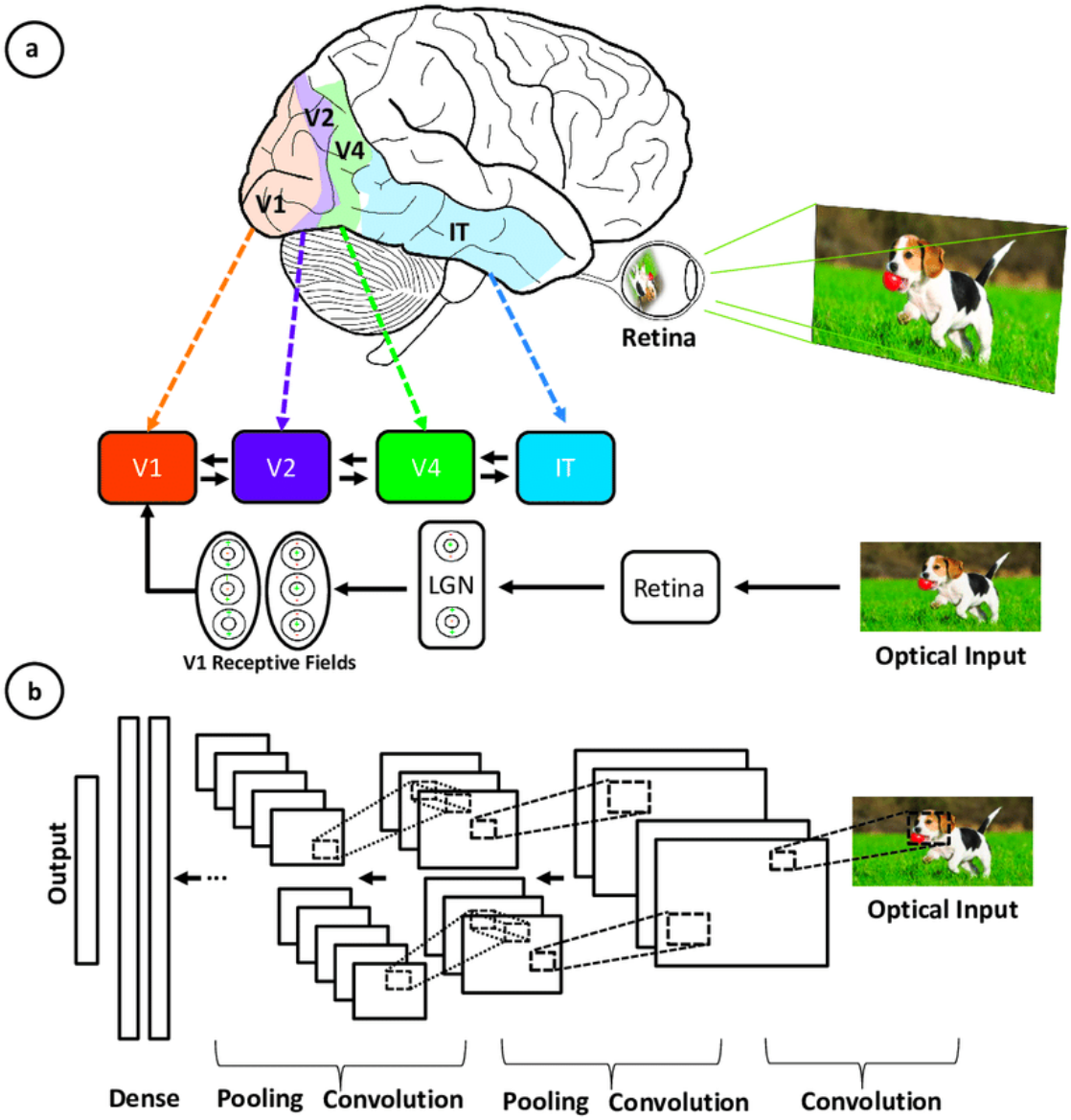


Рис. 5.2. Ілюстрація відповідності між областями, пов'язаними з первинною зоровою корою, і шарами згорткової нейронної мережі.

Створення згорткових нейронних мереж надихнула багаторівнева архітектура зорової кори людини. Нижче наведено деякі ключові подібності та відмінності (рис. 5.2):

- ієрархічна архітектура: як ЗНМ, так і зорова кора мають ієрархічну структуру з простими функціями, виділеними на ранніх рівнях, і більш складними функціями, побудованими на більш глибоких шарах. Це дозволяє поступово ускладнювати представлення візуальних входів;
- локальний зв'язок: нейрони зорової кори підключаються лише до локальної області вхідного сигналу, а не до всього поля зору. Подібним чином, нейрони в шарі ЗНМ підключаються лише до локальної області вхідного об'єму за допомогою операції згортки. Місцеве з'єднання підвищує ефективність;
- інваріантність зсуву: нейрони зорової кори можуть виявляти особливості незалежно від їх розташування в полі зору. Рівні субдискретизації в ЗНМ забезпечують певний ступінь інваріантності зсуву шляхом узагальнення локальних особливостей;
- кілька карт ознак: на кожному етапі візуальної обробки витягується багато різних карт ознак. ЗНМ імітують це за допомогою кількох карт ознак у кожному шарі згортки;
- нелінійність: нейрони в зоровій корі демонструють властивості нелінійної реакції. ЗНМ досягають нелінійності за допомогою функцій активації, таких як ReLU, що застосовуються після кожної згортки.
- ЗНМ імітують людську зорову систему, але є простішими, не мають складних механізмів зворотного зв'язку та покладаються на навчання із вчителем, а не без вчителя, що сприяє розвитку комп'ютерного зору, незважаючи на ці відмінності.

Отже, давайте розглянемо складові блоки типової нейронної мережі. Вона складається з:

1. Шар згортки – проводить вхідні зображення через набір згорткових фільтрів, кожен з яких активує певні функції із зображень.
2. Активаційна функція, найчастіше використовується ReLU – дозволяє прискорити і зробити ефективнішим навчання.
3. Пулінг або операція субдискретизації – спрощує виведення, виконуючи нелінійне зменшення простору, зменшуючи кількість параметрів, які потрібно ідентифікувати мережі.
4. Повнозв'язний шар – «вирівнює» двовимірні просторові функції мережі в вектор ознак для цілей класифікації.
5. Шар softmax розподіляє ймовірності для кожної категорії в наборі даних.

Порядок застосування зазначених вище функцій має значення. Деякі шари можуть повторюватись. Найтиповішою є наступна структура нейронної мережі:

1. Шар згортки, активаційна функція, пулінг – даний набір операцій і саме в такому порядку може бути повторюваний один чи декілька разів. Чим більше таких шарів, тим більш глибокою стає нейронна мережа.

2. Далі йде повнозв’язний шар – цей шар перетворює ознаки, отримані на виході з попереднього кроку до вигляду, що є зручним для задачі класифікації.
3. Останнім шаром йде softmax – задача цього шару перетворити вихід повнозв’язного шару на розподілення ймовірностей класів.

Далі розглядається визначення кожного з цих компонентів на прикладі класифікації рукописної цифри (рис. 5.3).

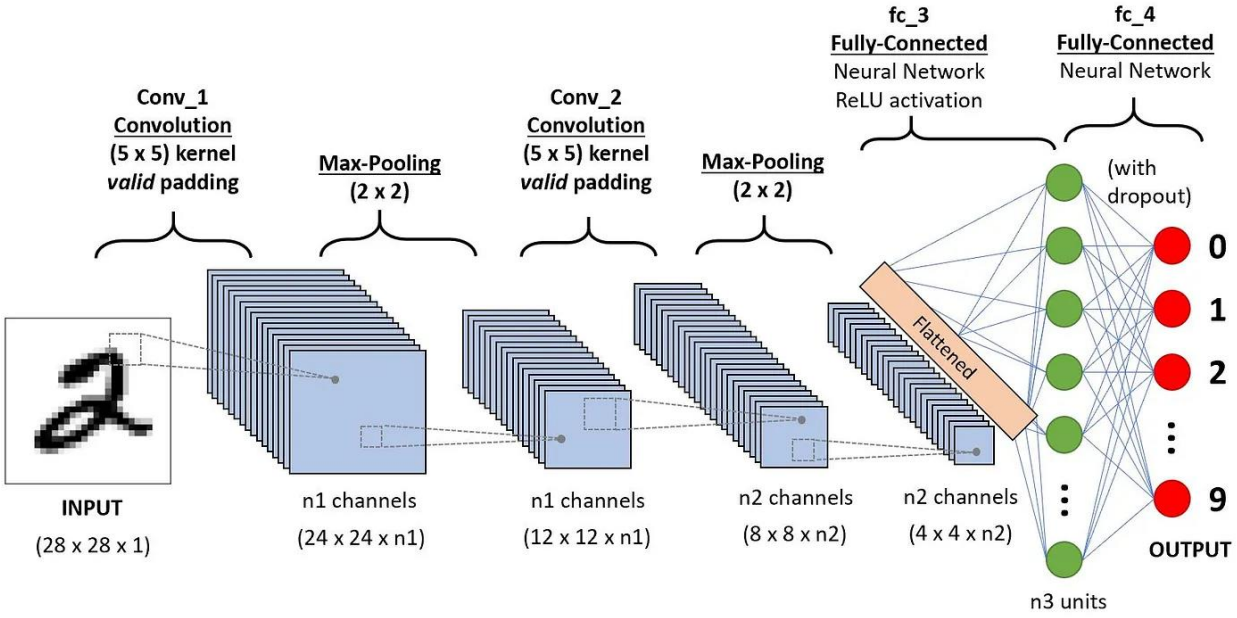


Рисунок 5.3. Архітектура ЗНМ, застосована до розпізнавання цифр.

5.2. Шари згортки

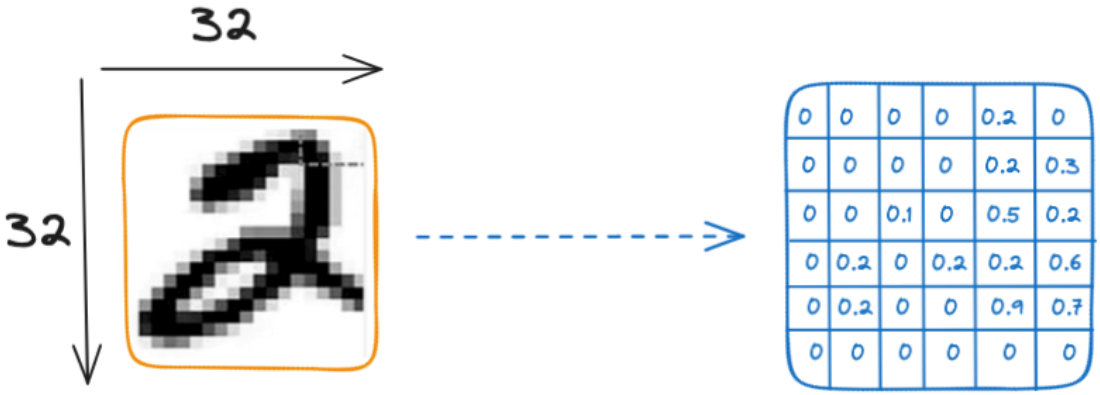
Це перший будівельний блок ЗНМ. Як випливає з назви, основна математична задача, яка виконується, називається згорткою, яка є застосуванням функції ковзного вікна до матриці пікселів, що представляє зображення. Ковзна функція, застосована до матриці, називається ядром або фільтром – назви, що можуть використовуватися як взаємозамінні.

У шарі згортки застосовуються кілька фільтрів однакового розміру, і кожен фільтр використовується для розпізнавання певного шаблону із зображення, наприклад викривлення цифр, країв, усієї форми цифр тощо.

Простіше кажучи, у шарі згортки ми використовуємо невеликі сітки (так звані фільтри або ядра), які переміщаються по зображенню. Кожна маленька сітка схожа на міні-збільшувальне скло, яке шукає на фотографії певні візерунки, як-от лінії, криві чи форми. Переміщаючись по фотографії, згортка створює нову сітку, яка виділяє, де знайшла шукані візерунки.

Наприклад, один фільтр може добре знаходити прямі лінії, інший – криві тощо. Використовуючи кілька різних фільтрів, ЗНМ може отримати гарне уявлення про всі різні шаблони, які складають зображення.

Давайте розглянемо зображення рукописної цифри (рис. 5.4) в градаціях сірого розміром 32 × 32. Значення в матриці наведені для ілюстрації.



(а) (б)

Рисунок 5.4. Ілюстрація вхідного зображення та його піксельне представлення. (а) – вхідне зображення, (б) – чисельне представлення пікселів.

Двовимірна згортка (2D convolution) – це досить проста операція: починаємо з ядра, що представляє собою матрицю ваг. Ядро «ковзає» над двовимірним зображенням, поелементно виконуючи операцію множення з тією частиною вхідних даних, над якою воно зараз знаходиться, а потім підсумовує всі отримані значення в один вихідний піксель.

Ядро повторює цю процедуру з кожною локацією (рис. 5.5), над якою воно «ковзає», перетворюючи двовимірну матрицю на іншу двовимірну матрицю ознак. Ознаки на виході є зваженими сумами (де ваги є значеннями самого ядра) ознак на вході, розташованих приблизно в тому ж місці, що вихідний піксель на вхідному шарі.

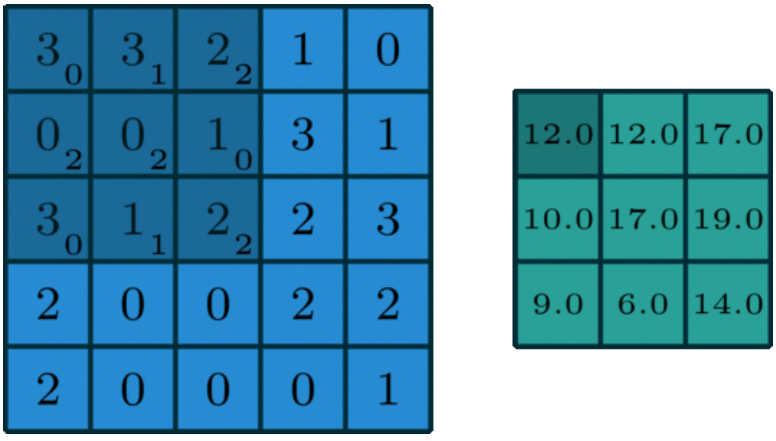


Рисунок 5.5. Один з кроків застосування операції згортки.

Розмір ядра згорткової нейронної мережі визначає кількість ознак, які будуть об'єднані для отримання нової ознаки на виході.

У наведеному вище прикладі ми маємо $5 \times 5 = 25$ ознак на вході і $3 \times 3 = 9$ ознак на виході. Для повнозв'язного шару ми мали б вагову матрицю $25 \cdot 9 = 225$ параметрів, а кожна вихідна ознака була би зваженою сумою всіх ознак на вході. Згортка дозволяє зробити таку операцію з усього 9-ма параметрами, адже кожна ознака на виході є результатом аналізу не кожної ознаки на вході, а тільки однієї вхідної, що знаходиться в «приблизно тому ж місці». Зверніть на це увагу, оскільки це буде важливим для подальшого обговорення.

Згортка, в цілому, все ще є лінійним перетворенням, але в той же час вона також є зовсім іншим видом перетворення. Для матриці з 64 елементами існує лише 9 параметрів, які повторно використовуються кілька разів. Кожен вихідний вузол отримує лише певну кількість входів (ті, що знаходяться всередині ядра). Немає жодної взаємодії з іншими входами, тому що вага для них дорівнює 0.

Локальні ознаки. Отже:

- ядра поєднують пікселі лише з невеликої локальної області для формування виходу. Тобто вихідні ознаки бачать лише вхідні ознаки із невеликої локальної області;
- ядро застосовується глобально по всьому зображення для створення матриці вихідних значень.

Таким чином, з методом зворотного розповсюдження помилки, що йде у всіх напрямках від вузлів класифікації мережі, ядра мають цікаву задачу вивчення ваг для створення ознак тільки з локального набору входів. Крім того, оскільки саме ядро застосовується по всьому зображенню, ознаки, які вивчає ядро, повинні бути достатньо спільними, щоб надходити з будь-якої частини зображення.

Якщо це були якісь інші дані, наприклад, дані про установки додатків за категоріями, то це стало б катастрофою, тому що наявність стовпців кількості установок додатків та типів додатків поруч не означає, що вони мають «локальні загальні ознаки», загальні з датами встановлення програм та часом використання. Звичайно, у кількох можуть бути основні ознаки вищого рівня, які можуть бути знайдені, але це не дає нам жодних підстав вважати, що параметри для перших двох такі самі, як параметри для останніх двох.

Пікселі, однак, завжди відображаються в послідовному порядку, а сусідні пікселі впливають на піксель поруч. Наприклад, якщо всі сусідні пікселі червоні, досить ймовірно, що піксель поряд також червоний. Якщо є відхилення, це цікава аномалія, яка може бути перетворена на ознаку і все це можна виявити при порівнянні пікселя зі своїми сусідами, з іншими пікселями у своїй місцевості.

Ця ідея – те, на чому були засновані більш ранні методи отримання ознак комп'ютерним зором. Наприклад, для виявлення граней можна використовувати фільтр виявлення граней Собеля – ядро з фіксованими параметрами, що діє так само, як стандартна одноканальна згортка (рис. 5.6).

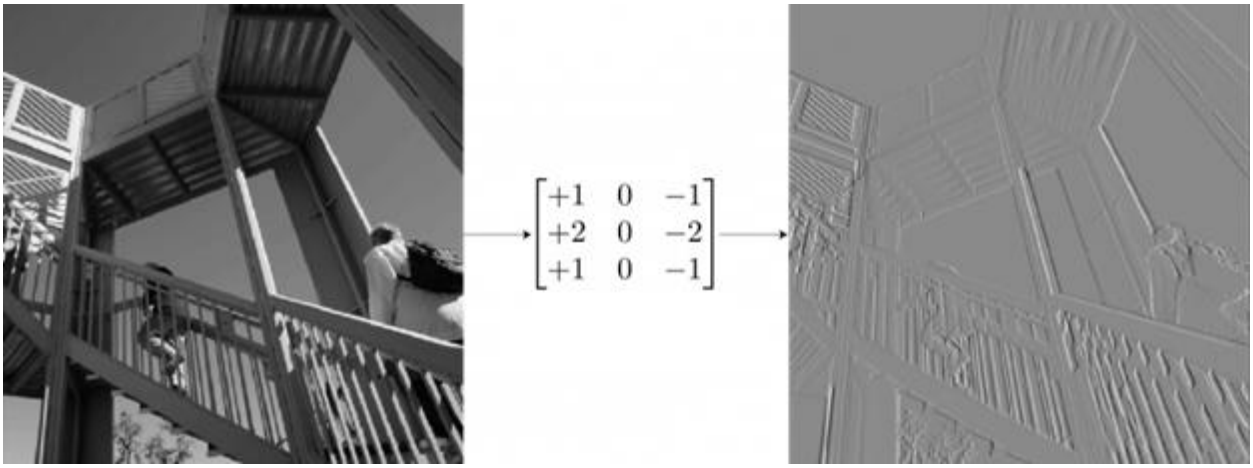


Рис. 5.6. Застосування ядра, що визначає межі.

Для сітки, що не містить граней (наприклад, неба на задньому фоні), більшість пікселів мають однакове значення, тому загальний висновок ядра в цій точці дорівнює 0. Для сітки з вертикальними гранями існує різниця між пікселями зліва і праворуч від грані, і ядро обчислює цю ненульову різницю, знаходячи ребра. Ядро за раз працює лише з сітками 3×3, виявляючи аномалії в певних місцях, але застосування до всього зображення достатньо для виявлення певної ознаки в будь-якому місці зображення!

Використовуючи вхідну матрицю та матрицю ядра, ми можемо виконати операцію згортки, застосовуючи скалярний добуток і працювати наступним чином (рис. 5.7):

1. Застосувати матрицю ядра від верхнього лівого кута до правого.
2. Виконати поелементно множення.
3. Просумувати значення добутків.
4. Отримане значення відповідає першому значенню (верхній лівий кут) у згорнутій матриці.
5. Перемістити ядро вниз відносно розміру ковзного вікна.
6. Повторювати кроки з 1 по 5, доки матриця зображення не буде повністю покрита.

Розмір згорнутої матриці залежить від розміру ковзного вікна. Чим вище в ковзне вікно, тим менше кінцева розмірність.

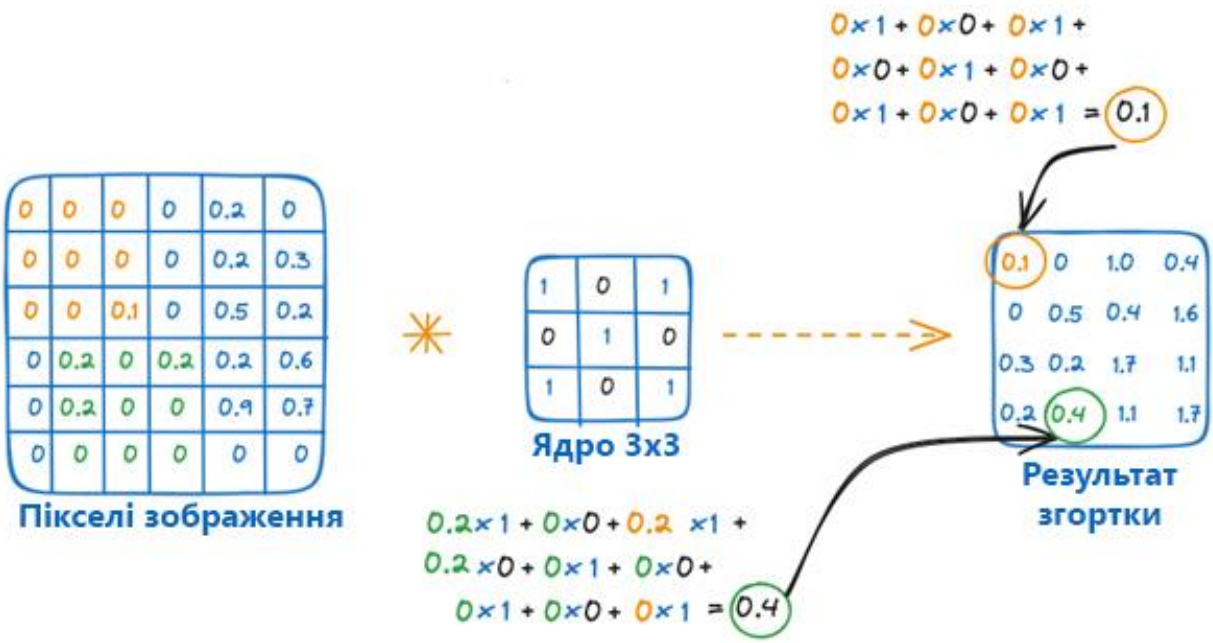


Рисунок 5.7. Застосування завдання згортки з кроком 1 із ядром 3×3.

5.3. Функція активації

Функція активації ReLU (рис. 5.8) застосовується після кожної операції згортки.

$$\mathrm{ReLU}(x) = \max(x, 0). \tag{5.1}$$

Ця функція допомагає мережі вивчати нелінійні зв'язки між об'єктами на зображенні, таким чином роблячи мережу більш надійною для визначення різних шаблонів і також допомагає пом'якшити проблеми зникнення градієнта.

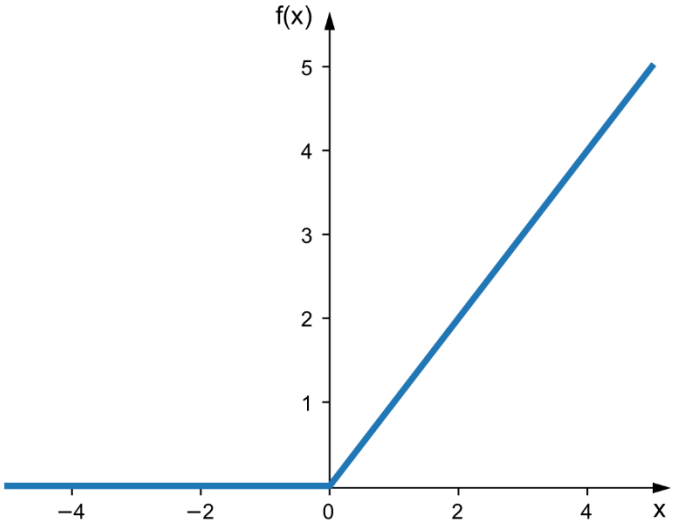


Рисунок 5.8. Функція активації ReLU.

5.4. Шар субдискретизації (Pooling)

Метою шару субдискретизації є вилучення найбільш значущих характеристик зі складної матриці. Це робиться шляхом застосування деяких операцій агрегації, які зменшують розмірність карти ознак (згорнутої матриці), отже, зменшуючи пам’ять, що використовується під час навчання мережі. Субдискретизація також актуальна для пом’якшення перенавчання.

Найпоширеніші функції агрегації, які можна застосувати:

- субдискретизація за максимумом, що є максимальним значенням карти ознак;
- субдискретизація за середнім – це середнє значення в ковзному вікні.

На рис. 5.9 наведено ілюстрацію субдискретизації за максимумом (MaxPooling).

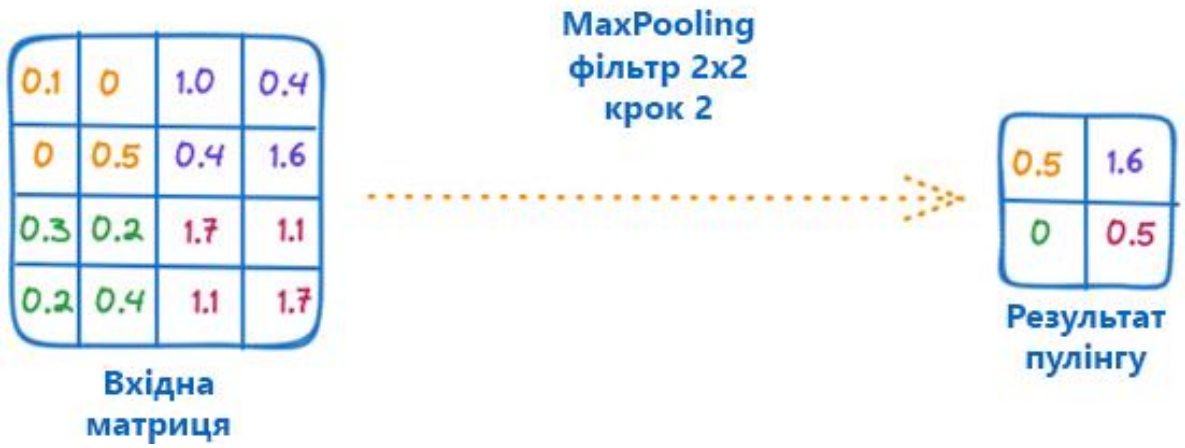


Рисунок 5.9. Застосування субдискретизації за максимумом з кроком 2 за допомогою фільтра 2×2.

Отже, нами було розглянуто основні структурні блоки згорткових нейронних мереж. Навчання такої мережі відбувається вже нам знайомим методом зворотного поширення помилки. Існує 2 підходи до навчання глибоких нейронних мереж:

1. Навчання мережі з нуля – найдовший, але найкраще пристосований для певної задачі.
2. Використання трансферного навчання для навчання існуючої мережі – вже навчену мережу, що розпізнає певні класи, навчають додатково на прикладах нових класів.

5.5. Контрольні питання

1. Для вирішення яких задач створено згорткові нейронні мережі?
2. Які основні задачі комп’ютерного зору ви знаєте?

3. В чому перевага автоматичного вилучення ознак з зображення над «ручним»?
4. Чи є місце повнозв’язному шару в згорткових нейронних мережах? Де його розташовують?
5. Чим операція субдискретизації відрізняється від згортки?
6. Чи є необхідною функція активації в згорткових мережах? Назвіть найбільш поширену функцію активації.
7. Які є способи навчання згорткових нейронних мереж?
8. В чому перевага згорткового блоку над повнозв’язним для задач комп’ютерного зору?
9. Де використання згорткового блоку в архітектурі згорткових нейронних мереж є недоречним?

Лекція 6 – Рекурентні нейронні мережі

В попередній лекції ми навчились передбачувати клас зображення за самим зображенням. Наше передбачення залежало лише від даного конкретного зображення x_i. В той самий час, є цілих клас задач, що вимагають передбачення для послідовності $x_1, \ldots, x_T$. Наприклад, звук, відео, текст, часовий ряд. Ми казатимемо, що кожна із таких послідовностей має часову компоненту t. Чи можна обійтись без послідовності? Уявіть собі, вам дають одне слово із тексту і просять передбачити тип документу: переписка із друзями, конспект, художня література. Так, іноді можна вгадати, але загалом за одним словом таку задачу якісно вирішити неможливо. Тому і було запропоновано так звані рекурентні нейронні мережі, що можуть враховувати історію спостережень, тобто часову компоненту. Далі давайте поговоримо про типи послідовностей із якими працюють нейронні мережі.

6.1. Типи послідовностей

Існує чотири типи послідовностей в залежності від кількості входів і виходів:

1. Один до одного.
2. Один до багатьох.
3. Багато до одного.
4. Багато до багатьох.

Приклад послідовності «один до одного зображено» на рис. 6.1. В даному випадку маємо звичайну нейронну мережу для класифікації табличних даних або окремих зображень без часової компоненти. З такою мережею ми вже знайомі з попередніх лекцій.

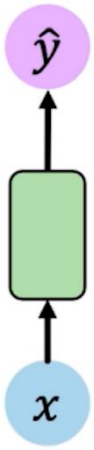


Рисунок 6.1. Тип послідовності «один до одного».

Приклад послідовності типу «один до багатьох» зображено на рисунку 6.2. Типовою задачею із таким типом послідовності є анотація зображень: на вхід до мережі подається одне статичне зображення, мережа в свою чергу має описати зміст зображення. Така задача відрізняється від звичайної класифікації зображень, оскільки відсутній фіксований перелік класів, натомість генерується речення, що якомога точніше описує зображення.

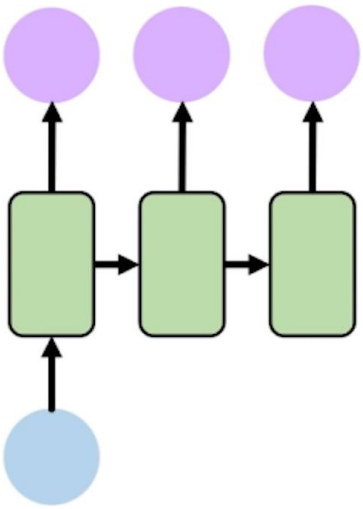


Рисунок 6.2. Тип послідовності «один до багатьох».

Приклад послідовності типу «багато до одного» зображено на рис. 6.3. При роботі з рекурентними мережами найчастіше ви будете зустрічати саме такий тип послідовності. В даному випадку на вхід подається поелементно часовий ряд, закодований текст або інша послідовність, на виході необхідно отримати одне значення. Однією із задач є класифікація емоційного забарвлення тексту, над якою ми будемо працювати в рамках індивідуального завдання в практичній роботі. На вхід маємо текст опису, наприклад, фільму, задача нейронні мережі визначити чи цей текст є «позитивним» або «негативним».

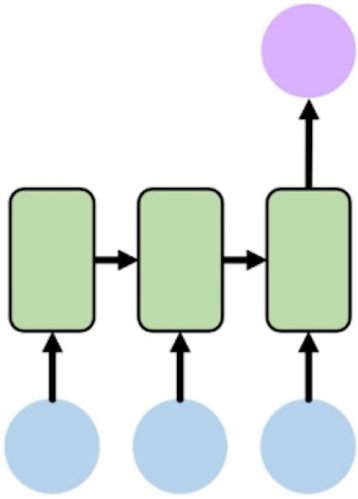


Рисунок 6.3. Тип послідовності «багато до одного».

Приклад послідовності «багато до багатьох» зображено на рис. 6.4. В даному випадку і на вході, і на виході маємо послідовності. Одним із прикладів цієї проблеми є переклад. У перекладі ми надаємо кілька слів з однієї мови як вхідні дані та прогнозуємо кілька слів з другої мови як вихідні дані.

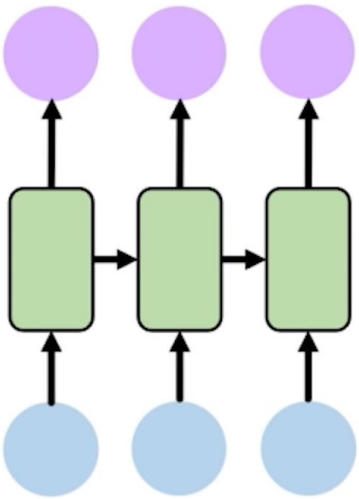


Рисунок 6.4. Тип послідовності «багато до багатьох».

До алгоритмів обробки послідовностей висуваються наступні вимоги:

1. Можливість роботи із послідовностями довільної довжини.
2. Відстеження далекої (у часі) залежності.
3. Розуміння порядку у сигналі.

6.2. Побудова рекурентного блоку

Розглянемо яким чином можливо побудувати блок рекурентної нейронної мережі. Для цього повернемось до моделі повнозв'язного шару, спрощено зображеної на рис. 6.5, де $x_t \in \mathbb{R}^m$ – це деякий фіксований вектор вхідних ознак, а $\hat{y}_t \in \mathbb{R}^n$ – вектор вихідних ознак. Такий шар оброблює вхід лише за деякий конкретний момент часу t.

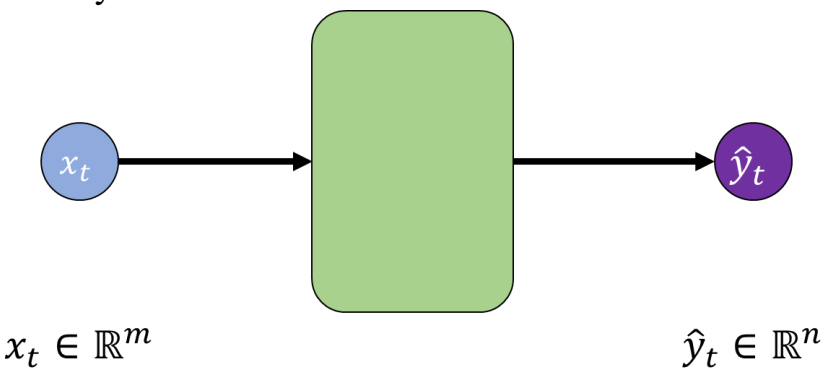


Рис. 6.5. Спрощений вигляд повнозв'язного шару.

Очевидно, що такий шар ніяким чином не враховує наявність послідовності в сигналі, та не може моделювати залежності в часі. Для виправлення зазначеного недоліку, зробимо наступні прості перетворення:

1. Розпишемо декілька повнозв'язних шарів для моментів у часі $t = 0,1,2$, як зображено на рис. 6.6.

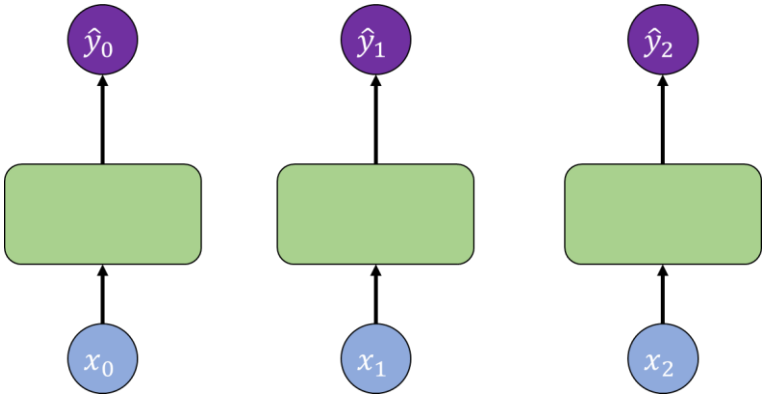


Рис. 6.6. Повнозв'язні шари для декількох моментів у часі.

2. Пов'яжемо окремі повнозв'язні шари додатковим зв'язком таким чином, щоб кожен повнозв'язний шар приймав деякий вектор із історією h_{t-1}, та видавав новий вектор із історією h_t, як показано на рис. 6.7.

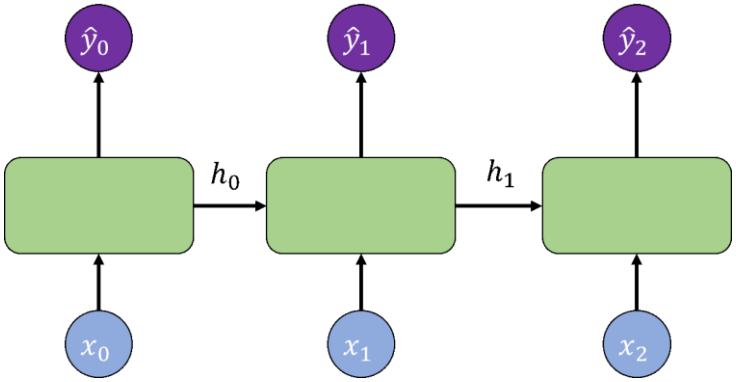


Рис. 6.7. Повнозв'язні шари із зв'язками між ними.

Отже таким чином, кожен блок буде рахувати вже 2 функції:

1. Функцію для передбачення виходу із блоку $\hat{y}_t = f(x_t, h_{t-1})$, де x_t – це елемент послідовності (слово), що подається на вхід, h_{t-1} – вектор із історією, що містить інформацію про попередні елементи послідовності, $\hat{y}_t$ – передбачення для поточного кроку t.
2. Функцію для передбачення вектору з історією для наступного блоку $h_t = g(x_t, h_{t-1})$, який також залежить від входу на поточному кроці x_t та історії h_{t-1}.

Те, що ми отримали, має назву рекурентного блоку RNN. Його зображено на рис. 6.8. Зверніть увагу на додаткові зв'язки між суміжними блоками. Такий блок вже може бути ефективно використаний для передбачення послідовностей.

Важливим ускладненням даного блоку є LSTM – блок із довгою короткостроковою пам’яттю, що здебільшого є аналогічним до блоку RNN, але моделює окремо довгі та короткі послідовності. На практиці найчастіше використовується саме блок LSTM.

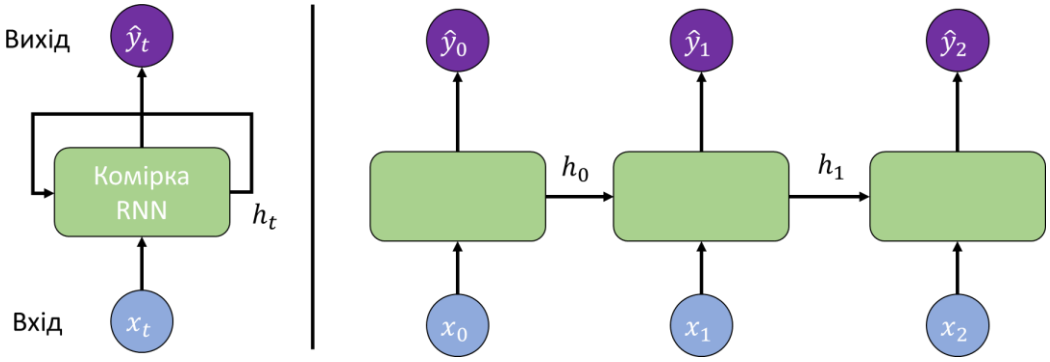


Рис. 6.8. Комірка рекурентної нейронної мережі.

Для навчання рекурентної нейронної мережі використовується алгоритм зворотного поширення в часі (BPTT).

Зворотне поширення в часі є розширенням традиційного алгоритму зворотного поширення, призначеного для роботи з рекурентними нейронними мережами (RNN), які обробляють послідовні дані. Це метод для обчислення градієнтів у RNN, який є важливим для навчання цих моделей.

У традиційних нейронних мережах прямого зв’язку кожен шар обробляє вхідні дані лише один раз, а потім передає їх на наступний шар. Натомість, RNN зберігають внутрішній стан (також відомий як прихований стан), який фіксує інформацію з попередніх часових кроків. Це дозволяє їм моделювати часові зв’язки в даних. Однак це також означає, що помилки можуть поширюватися з часом, що ускладнює обчислення градієнтів для навчання.

6.3. Метод зворотного поширення в часі

Схему алгоритму навчання рекурентної нейронної зображено на рис. 6.9. Основні кроки алгоритму складають:

Алгоритм 6.1. Зворотне поширення помилки в часі.

1. Розгортання: розбиття RNN на окремі блоки, що відповідають різним моментам в часі.
2. Прямий прохід: обчислення виходу розгорнутої мережі, обробляючи вхідну послідовність покроково.
3. Зворотній прохід: обчислення градієнтів помилок на кожному кроці часу за допомогою правила ланцюга, а саме:
    a. Обчислення градієнту функції втрат відносно виходу на кожному часовому кроці.

b. Зворотне поширення помилок через розгорнуту мережу, обчислення градієнтів втрат щодо параметрів моделі та вхідних послідовностей.

4. Накопичення градієнта: накопичення градієнтів, обчислених на кожному кроці часу, щоб отримати загальний градієнт для всієї послідовності.
5. Оновлення параметрів: оновлення параметрів моделі за допомогою накопичених градієнтів і алгоритму оптимізації.

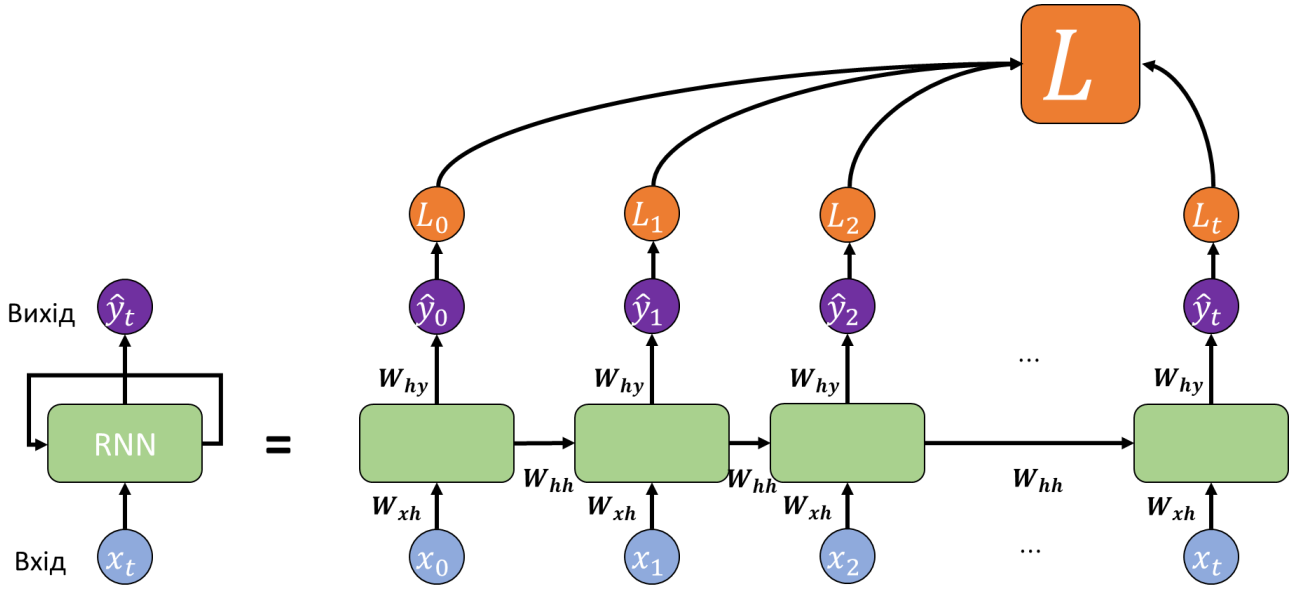


Рисунок 6.9. – Зворотне поширення в часі у RNN.

6.4. Контрольні питання

1. Чи є доречним використання повнозв'язного блоку нейронної мережі для моделювання послідовностей? Відповідь обґрунтуйте.
2. Назвіть типи послідовностей із якими працюють нейронні мережі. Для кожного типу наведіть приклади задач.
3. За допомогою яких модифікацій повнозв'язний блок можливо перетворити на рекурентний блок нейронної мережі?
4. В чому мета контексту h в рекурентних нейронних мережах? Чи є він необхідним?
5. Які з типів послідовностей доречно вирішувати за допомогою рекурентного блоку?
6. Які матриці параметрів наявні в рекурентній мережі?
7. Що змінюється в рекурентному блоці із зміною часового кроку t деякої послідовності?
8. Як необхідно змінити метод зворотного поширення помилки для застосування в рекурентних нейронних мережах? Яку назву такий метод має?

Лекція 7 – Нейронна мережа із довгою короткостроковою пам’яттю

Багато мов, у тому числі українська, складаються з ряду слів, розділених пробілами, разом із знаками пунктуації, і тому є прикладом послідовних даних. На даний момент ми зосередимося на словах, а потім повернемося до розділових знаків.

Перша задача полягає в тому, щоб перетворити слова в числове представлення, придатне для використання в якості вхідних даних для глибокої нейронної мережі. Одним із простих підходів є визначення фіксованого словника слів, а потім введення векторів довжини, що дорівнює розміру словника, разом із представленням з «одним активним станом» для кожного слова, у якому k-те слово у словнику кодується вектором, що має 1 у позиції k і 0 у всіх інших позиціях. Наприклад, якщо «вовк» є третім словом у нашому словнику, тоді його векторне представлення буде $(0,0,1,0,\dots,0)$. Очевидною проблемою представлення з одним активним станом є те, що реалістичний словник може мати кілька сотень тисяч записів, що ведуть до векторів дуже високої розмірності. Крім того, він не фіксує жодної подібності чи зв’язку, які можуть існувати між словами. Обидві проблеми можна вирішити шляхом відображення слів у просторі меншої розмірності за допомогою процесу, що називається вкладанням слів, у якому кожне слово представлено як щільний вектор у просторі, як правило, у кілька сотень вимірів.

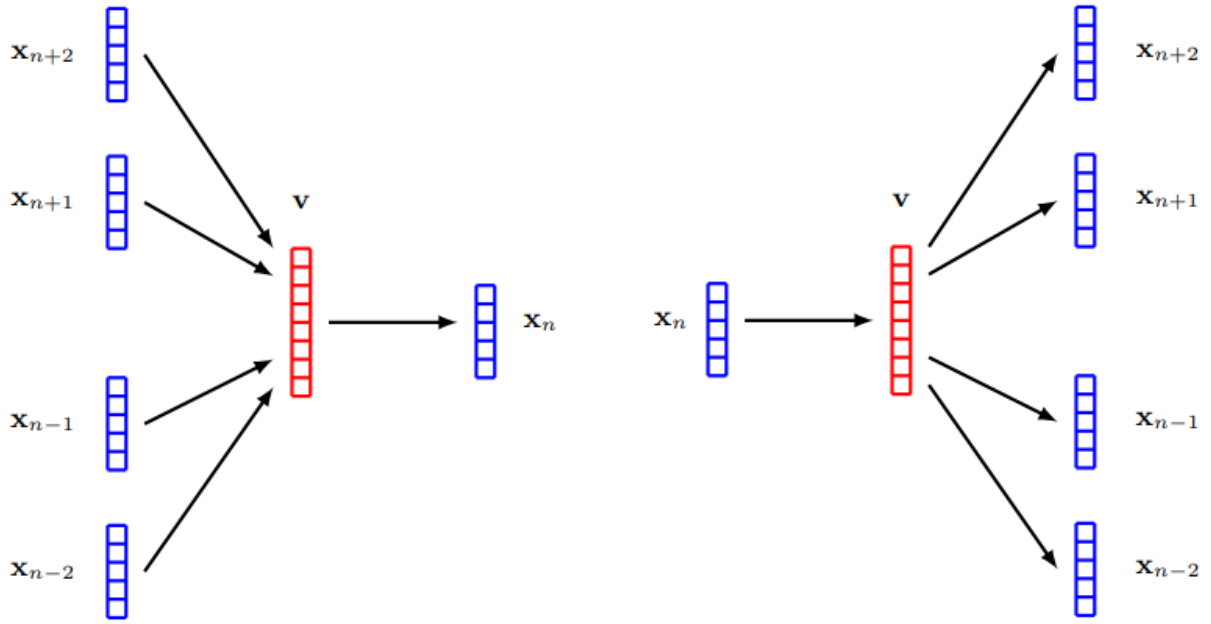
7.1. Вкладання слів

Процес вкладання може бути визначений матрицею E розміром $D \times K$, де D – розмірність простору вкладання, а K – розмірність словника. Для кожного вхідного вектора x_n з кодуванням з одним активним станом ми можемо обчислити відповідний вектор вкладання за допомогою:

$$v_n = Ex_n. \tag{7.1}$$

Оскільки x_n має кодування з одним активним станом, вектор v_n просто задається відповідним стовпцем матриці E.

Ми можемо дізнатися матрицю E з корпусу (тобто великого набору даних) тексту. Для цього існує багато підходів. Розглянемо популярну техніку під назвою word2vec, яку можна розглядати як просту двошарову нейронну мережу. Будується навчальний набір, у якому кожна вибірка отримана шляхом розгляду «вікна» з M суміжних слів у тексті, де типове значення може бути $M = 5$. Зразки вважаються незалежними, а функція помилки визначається як сума функцій похибок для кожної вибірки. Є два варіанти цього підходу. У безперервній «торбі слів» цільовою змінною для навчання мережі є середнє слово, а решта контекстних слів формують вхідні дані, щоб мережу навчали «заповнювати прогалини». Тісно пов’язаний підхід, який називається skip-grams, міняє входи та виходи таким чином, що центральне слово представляється як вхід, а цільові значення є контекстними словами. Ці моделі показано на рис. 7.1.



(а) (б)

Рис. 7.1. Двошарові нейронні мережі для вивчення вкладання слів. (а) – підхід безперервної торби слів; (б) – skip-gram.

Цю процедуру навчання можна розглядати як форму самоконтрольованого навчання, оскільки дані складаються просто з великого корпусу тексту без міток, з якого навмання витягується багато маленьких віконечок із послідовностями слів. Мітки отримують із самого тексту шляхом «маскування» тих слів, значення яких намагається передбачити мережа.

Коли модель навчена, матриця вкладання E задається транспонуванням вагової матриці другого шару мережі для безперервного підходу сумки слів і ваговою матрицею першого шару для skip-gram. Слова, які семантично пов’язані, відображаються в сусідні позиції у просторі вкладання. Це цілком очікувано, оскільки споріднені слова частіше зустрічаються з подібними контекстними словами, ніж неспоріднені слова. Наприклад, слова «місто» та «столиця» можуть зустрічатися частіше як контекст для цільових слів, таких як «Париж» або «Лондон», і рідше як контекст для «помаранчевий» або «множник». Мережі легше передбачити ймовірність пропущених слів, якщо «Париж» і «Лондон» зіставити з найближчими векторами вбудовування.

Виявляється, що вивчений простір вбудовування часто має навіть багатшу семантичну структуру, ніж просто близькість споріднених слів, і це дозволяє виконувати просту векторну арифметику. Наприклад, концепцію «Париж для Франції, як Рим для Італії» можна виразити за допомогою операцій над векторами вкладення. Якщо ми використовуємо $v(\text{слово})$ для позначення вектора вкладення для «слова», тоді виявиться:

$$v(Paris) - v(France) + v(Italy) \approx v(Rome). \qquad (7.2)$$

Вкладення слів спочатку було розроблено як самостійний інструмент обробки природної мови. Сьогодні їх частіше використовують як етапи попередньої обробки для глибоких нейронних мереж. У цьому відношенні їх можна розглядати як перший шар глибокої нейронної мережі. Їх можна зафіксувати за допомогою стандартної попередньо навченої матриці вкладення або їх можна розглядати як адаптивний шар, який вивчається як частина загального наскрізного навчання системи. В останньому випадку шар вкладення можна ініціалізувати або за допомогою випадкових значень ваги, або за допомогою стандартної матриці складення.

7.2. Токенізація

Однією з проблем використання фіксованого словника слів є те, що він не може впоратися зі словами, яких немає у словнику або які написані з помилками. Він також не враховує знаки пунктуації чи інші послідовності символів, наприклад комп'ютерний код. Альтернативний підхід, який вирішує ці проблеми, полягає в тому, щоб працювати на рівні символів замість використання слів, щоб наш словник включав великі та малі літери, цифри, знаки пунктуації та пробіли. Недолік цього підходу, однак, полягає в тому, що він відкидає семантично важливу структуру слів мови, і наступна нейронна мережа повинна буде навчитися збирати слова з елементарних символів. Це також вимагало б значно більшої кількості послідовних кроків для певного тексту, тим самим збільшуючи обчислювальні витрати на обробку послідовності.

Ми можемо поєднати переваги представлень на рівні символів і на рівні слів, використовуючи етап попередньої обробки, який перетворює рядок слів і знаків пунктуації на рядок токенів, які, як правило, є невеликими групами символів і можуть містити загальні слова цілком, фрагменти довших слів, а також окремі символи, які можна зібрати в менш поширені слова. Ця токенізація також дозволяє системі обробляти інші види послідовностей, наприклад комп'ютерний код. Це також означає, що варіанти того самого слова можуть мати пов'язані представлення. Наприклад, «cook», «cooks», «cooked», «cooking» і «cooker» пов'язані між собою та мають спільний елемент «cook», який сам по собі може бути представлений як один із токенів.

У практичних застосуваннях глибокого навчання природної мови вхідний текст зазвичай спочатку відображається в символізованому представленні. Однак у решті цієї лекції ми будемо використовувати представлення на рівні слів, оскільки це полегшує ілюстрацію та мотивацію ключових концепцій.

7.3. Проблема довгострокової залежності

Вище нами було розглянуто, які способи кодування слів для використання в глибоких нейронних мережах існують. Тепер давайте повернемось до проблеми рекурентних нейронних мереж з попередньої лекції. Основна проблема вивчення довгострокових залежностей полягає в тому, що градієнти, що поширюються на багатьох етапах, мають тенденцію або зникати (більшість часу), або вибухати (рідко, але з великою шкодою для оптимізації). Навіть якщо ми припустимо, що параметри такі, що рекурентна мережа є стабільною (може зберігати спогади, з градієнтами, що не вибухають), труднощі з довгостроковими залежностями виникають через експоненціально менші ваги, надані довгостроковим взаємодіям порівняно з короткочасними.

Рекурентні мережі передбачають композицію тієї самої функції кілька разів, один раз за часовий крок. Під час композиції багатьох нелінійних функцій результат є дуже нелінійним, зазвичай із більшістю значень, пов'язаних із маленькою похідною, деякі значення з великою похідною та багатьма чергуваннями між збільшенням і зменшенням. На рис. 7.2. показано побудовану лінійну проекцію 100-вимірного прихованого стану шару з функцією активації $tanh$ на один вимір, нанесений на вісь y. Вісь x – це координата початкового стану вздовж випадкового напрямку в 100-вимірному просторі. Таким чином, ми можемо розглядати цей графік як лінійний переріз високовимірної функції. Графіки показують функцію після кожного кроку в часі або, еквівалентно, після кожної кількості композицій функції.

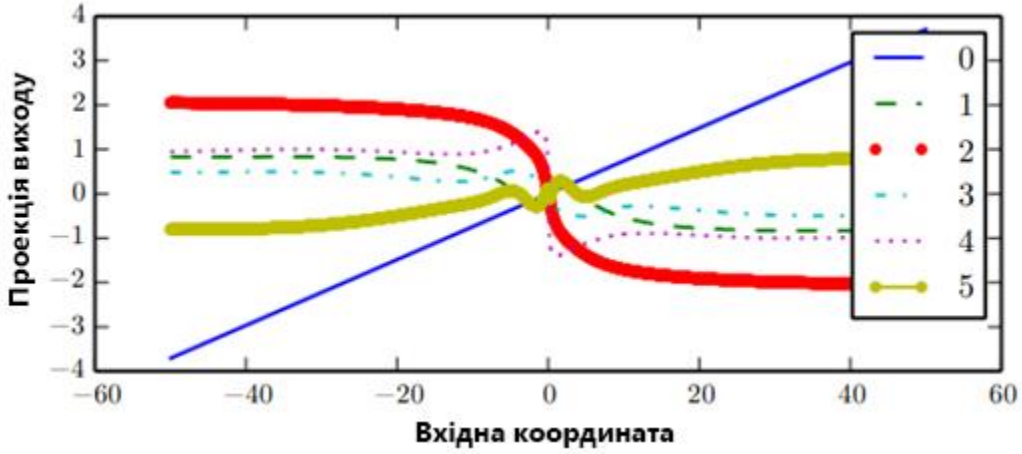


Рис. 7.2. Багатократна композиція нелінійних функцій.

Зокрема, композиція функцій, яка використовується рекурентними нейронними мережами, дещо нагадує множення матриць. Можна розглядати рекурентне співвідношення:

$$h^{(t)} = W^T h^{(t-1)} \tag{7.3}$$

як дуже просту рекурентну нейронну мережу без нелінійної функції активації та входів x. Дане співвідношення може бути спрощено до:

$$h^{(t)} = (W^t)^T h^{(0)}, \tag{7.4}$$

і якщо W допускає власний розклад виду:

$$W = Q\Lambda Q^T, \tag{7.5}$$

якщо Q – ортогональна матриця, рекурентне співвідношення може бути далі спрощено до:

$$h^{(t)} = Q^T \Lambda^t Q h^{(0)}. \tag{7.5}$$

Тобто власні значення Λ матриці W зводяться до степеня t, внаслідок чого власні значення з величиною, меншою за одиницю, спадають до нуля, а власні значення з величиною, більшою за одиницю, вибухають. Будь-який компонент $h^{(0)}$, який не узгоджується з найбільшим власним вектором, зрештою буде відкинуто.

Ця проблема є характерною для рекурентних нейронних мереж. У скалярному випадку уявіть, що вага w множиться сама на себе багато разів. Добуток w^t або зникне, або вибухне залежно від величини w. Однак, якщо ми створимо нерекурентну мережу, яка має різну вагу $w^{(t)}$ на кожному часовому кроці, ситуація буде іншою. Якщо початковий стан задається 1, то стан у момент часу t задається як $\prod_t w^{(t)}$. Припустимо, що значення $w^{(t)}$ генеруються випадковим чином, незалежно одне від одного, з нульовим середнім і дисперсією v. Дисперсія добутку $O(v^n)$. Щоб отримати деяку бажану дисперсію v^*, ми можемо вибрати окремі ваги з дисперсією $v = \sqrt[n]{v^*}$. Таким чином, дуже глибокі мережі прямого зв'язку з ретельно підібраним масштабуванням можуть уникнути проблеми зникнення та вибуху градієнта.

Проблема зникнення та вибухового градієнта для RNN була незалежно відкрита окремими дослідниками. Здавалося б, що проблеми можна уникнути, просто залишаючись в області простору параметрів, де градієнти не зникають і не вибухають. Однак, як було показано, для того, щоб пам'ять мережі RNN була стійкою до невеликих збурень, параметри RNN мають увійти в область, де градієнти зникають. Зокрема, якщо модель здатна представити довгострокові залежності, градієнт довгострокової взаємодії має експоненціально меншу величину, ніж градієнт короткострокової взаємодії. Це не означає, що неможливо навчитися, а те, що для вивчення довгострокових залежностей може знадобитися дуже багато часу, тому що сигнал про ці залежності буде, як правило, прихований найменшими флуктуаціями, що виникають через короткочасні залежності. На практиці експерименти показують, що в міру збільшення діапазону залежностей, які необхідно охопити, оптимізація градієнтними методами стає дедалі складнішою, при цьому ймовірність успішного навчання традиційної RNN через метод градієнтного спуску швидко досягає 0 для послідовностей довжиною лише 10 або 20.

Таким, чином для більш довгих послідовностей необхідна розробка іншого механізму зберігання та відстеження довгих послідовностей, що і буде розглянуто в наступному підрозділі.

7.4. Мережа із довгою короткочасною пам'яттю

Розумна ідея введення самоциклів для створення шляхів, де градієнт може протікати протягом тривалого часу, є основним внеском початкової моделі довгої короткочасної пам’яті (Long Short-Term Memory, LSTM). Вирішальним доповненням було те, що вагомість цього самоциклу залежить від контексту, а не є фіксованою. Зробивши вагу цього самоциклу керованою іншим прихованим блоком, часову шкалу інтеграції можна динамічно змінювати. У цьому випадку ми маємо на увазі, що навіть для LSTM з фіксованими параметрами часовий масштаб інтеграції може змінюватися на основі вхідної послідовності, оскільки постійні часу виводяться самою моделлю. LSTM було визнано надзвичайно успішним у багатьох програмах, таких як необмежене розпізнавання рукописного тексту, розпізнавання мовлення, створення рукописного тексту, машинний переклад та анотація зображень.

Блок-схему LSTM проілюстровано на рис. 7.3. Комірки рекурентно підключаються одна до одної, замінюючи приховані вузли звичайних рекурентних мереж. Вхідна функція обчислюється за допомогою штучного нейрона. Як видно з рис. 7.3 у LSTM є три основних види вузлів, які називаються «воротами входу» (input gate), «ворота забування» (forget gate), «ворота виходу» (output gate), а також власне рекурентна комірка із прихованим станом.

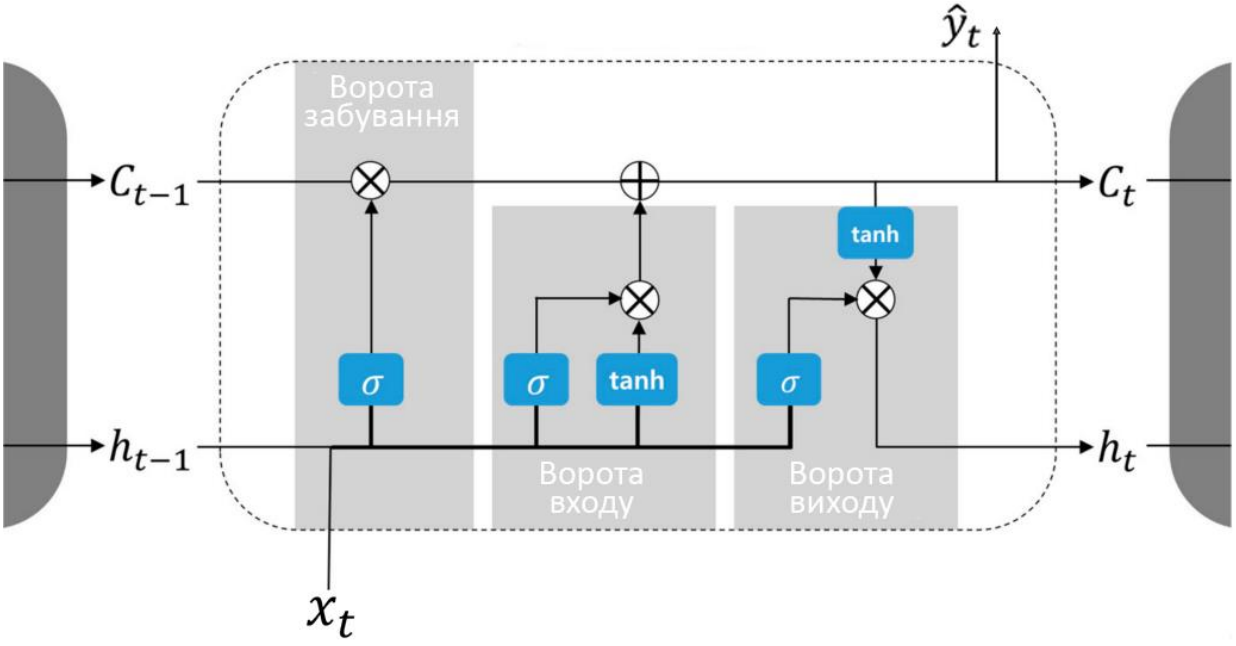


Рис. 7.3. Блок-схема комірки мережі LSTM.

Формально кажучи, якщо позначити через x_t вхідний вектор в час t, через h_t – вектор прихованого стану в час t, через W_i (з різними другими індексами) – матриці ваг, що застосовуються до входу, через W_h – матриці ваг у рекурентних з'єднаннях, а через b – вектори вільних членів, ми отримаємо наступне формальне визначення того, як працює LSTM: на черговому вході x_t, маючи прихований стан попереднього кроку h_{t-1} і власне стан комірки c_{t-1}, ми послідовно обчислюємо:

– кандидата в стан комірки:

$$c'_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_{c'}); \quad (7.6)$$

– ворота входу:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i); \quad (7.7)$$

– ворота забування:

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right); \quad (7.8)$$

– ворота виходу:

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_0); \quad (7.9)$$

– стан комірки:

$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t; \quad (7.10)$$

– вихід блоку:

$$h_t = o_t \odot \tanh(c_t), \quad (7.11)$$

де $\odot$ – поелементний добуток. В формулах (7.6) – (7.11) наведено канонічну реалізацію мережі LSTM.

На вхід LSTM, як і в «звичайній» RNN, подаються два вектори: новий вектор з вхідних даних x_t і вектор прихованого стану h_{t-1}, отриманий із прихованого стану цієї комірки на попередньому кроці. Крім того, всередині кожного LSTM-блоку є «комірка пам'яті» – вектор, який виконує функцію пам'яті. Вектор комірки на кроці t ми позначили вище через c_t, а c'_t, який виходить у першому ж рівнянні, – це вектор, отриманий із входу та попереднього прихованого стану, який стає кандидатом на нове значення пам'яті. Виходить він з x_t і h_{t-1} звичайним для нейронних мереж перетворенням: спочатку лінійна функція, потім гіперболічний тангенс, все як у звичайних нейронних мережах.

Але c'_t – це лише кандидат у нове значення пам'яті. Перш ніж його запишуть на місце c_{t-1}, значення-кандидат та старе значення проводять ще через двоє воріт: вхідні ворота i_t та ворота забування f_t. Подивіться на формулу 7.10. Тут нове значення виходить як лінійна комбінація зі старого з коефіцієнтами з воріт f_t і нового кандидата c'_t з коефіцієнтами з вхідних воріт i_t. Там, де значення вектора воріт забування f_t будуть близькими до нуля, старе значення c_{t-1} «забуватиметься», а там, де значення i_t будуть великими, новий вхідний вектор додасться до того, що було в пам'яті.

Зверніть увагу: покомпонентне множення призводить до того, що на черговому кроці може бути перезаписана лише частина пам'яті LSTM-комірки; і яка це буде частина, теж визначає сама комірка в залежності від того, що виходить

на виходах воріт забування f_t і вхідних воріт i_t. І більш того, оскільки ця лінійна комбінація «м'яка» і по дорозі все пропускається через сигмоїди σ, LSTM-комірка може не просто вибрати, записати нове значення або викинути його, а ще й зберегти будь-яку лінійну комбінацію старого та нового значення, причому коефіцієнти можуть бути різними у різних компонентах вектора; і ці рішення комірка приймає залежно від конкретного входу. Все це робить LSTM-комірки дуже гнучкими; а якщо тепер згадати, що таких комірок у нас багато, з них складається рекурентна мережа, стає зрозуміло, чому LSTM-комірки виявилися настільки успішною модифікацією.

Але це все були абстрактні міркування: ми б хотіли, щоб гнучкість архітектури LSTM дозволяла навчати довгострокові залежності, і теоретично вона, звичайно, може це зробити. Однак архітектура звичайних рекурентних мереж теж теоретично дозволяє навчити все, що завгодно, але на практиці так не виходить; чим же LSTM принципово відрізняється від стандартних RNN?

Справа в тому, що LSTM завдяки своїй архітектурі вирішує проблему зникаючих градієнтів, яка заважала рекурентним мережам навчати довгострокові залежності. Щоб краще побачити різницю, давайте на якийсь час уявімо собі LSTM без воріт забування (власне, саме в такому вигляді LSTM і з'явився спочатку); у наших позначеннях вважатимемо, що $f_t = 1$ у всіх компонентах та для всіх t. Тоді вектор «пам'яті» комірки буде обчислюватися як

$$c_t = c_{t-1} + i_t \odot c'_t. \tag{7.12}$$

А це означає, що

$$\frac{\partial c_t}{\partial c_{t-1}} = 1. \tag{7.13}$$

Цей ефект, при якому в рекурсивному обчисленні стану комірки немає ніякої нелінійності, в літературі називається «каруселлю константної помилки»: помилки в мережі з LSTM пропагуються без змін, і приховані стани LSTM можуть, якщо комірка не вирішить їх перезаписати зберігати свої значення необмежено довго. Це вирішує проблему «зникаючих градієнтів»: незалежно від матриці рекурентних ваг помилка сама загасати не буде.

Загальну схему канонічної реалізації LSTM за формулами (7.6)-(7.11) із вказанням матриць параметрів показано на рис. 7.4.

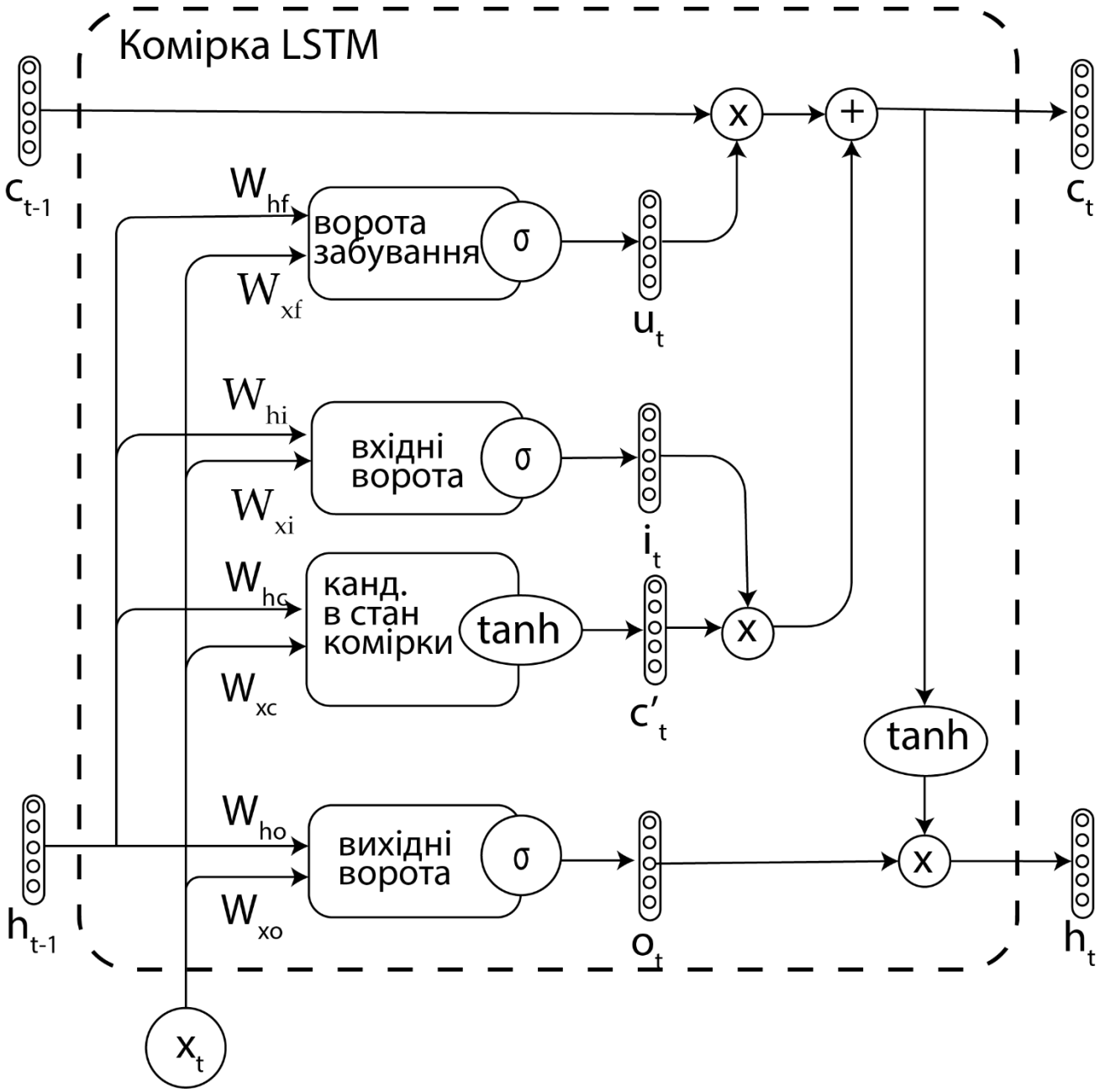


Рис. 7.4. Блок-схема LSTM із вказанням параметрів воріт.

7.5. Контрольні питання

1. Назвіть основні підходи для створення вкладання слів.
2. Які властивості мають векторні представлення вкладання слів?
3. В чому полягає суть процесу токенізації слів?
4. Яка проблема існує в моделюванні довгих залежностей за допомогою рекурентної нейронної мережі?
5. Наведіть основні типі воріт в комірці LSTM. Які функції виконують кожні з них?
6. Які функції активації використовуються в LSTM?

7. Порівняйте шлях градієнта в методі зворотного поширення помилки в часі для RNN і LSTM. Де градієнт буде стухати швидше?
8. Як пов’язані короткостроковий h та довгостроковий c контексти? Як вони оновлюються?

Лекція 8 – Відновлення сигналів за допомогою штучних нейронних мереж

Автокодувальник – це нейронна мережа, навчена копіювати вхідні дані на вихід мереж. Всередині автокодувальник має прихований шар h, який описує *код*, який використовується для представлення вхідних даних. Мережа може розглядатися як така, що складається з двох частин: функції кодувальника $h = f(x)$ і декодувальника, який виробляє реконструкцію $r = g(h)$.

Загальну структуру мережі автокодувальника представлено на рис. 8.1. Мережа відображає вхід x на вихід r (що зветься реконструкцією) через внутрішнє представлення або код h. Автокодувальник має 2 компоненти: кодувальник f (відображення x в h), декодувальник g (відображення h в r).

Якщо автокодувальнику вдається просто навчитися встановлювати $g(f(x)) = x$ усюди, то це не особливо корисно. Натомість автокодувальники розроблені таким чином, що вони не можуть навчитися ідеально копіювати. Зазвичай вони обмежені способами, які дозволяють копіювати лише приблизно та копіювати лише вхідні дані, які нагадують навчальні дані. Оскільки модель змушена визначати пріоритетність того, які аспекти вхідних даних слід скопіювати, вона часто вивчає корисні властивості даних.

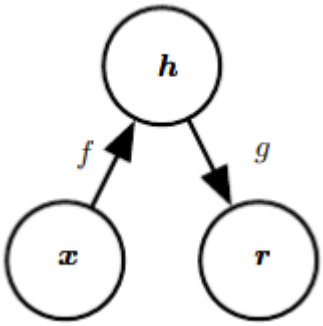


Рис. 8.1. Загальна структура мережі автокодувальника.

Сучасні автокодувальники узагальнили ідею кодувальника та декодувальника за межі детермінованих функцій до стохастичних відображень $p_{\text{кодувальник}}(h|x)$ та $p_{\text{декодувальник}}(x|h)$.

Ідея автокодувальників була частиною історичного ландшафту нейронних мереж протягом десятиліть. Традиційно автокодувальники використовувалися для зменшення розмірності або вивчення ознак. Нещодавно теоретичні зв'язки між автокодувальниками та моделями латентних змінних вивели автокодувальники на передній план генеративного моделювання. Автокодувальники можна розглядати як окремий випадок мереж прямого зв'язку, і їх можна навчати з усіма тими ж методами, як правило, градієнтний спуск за градієнтами, обчисленими методом зворотного поширення. На відміну від

загальних мереж прямого зв'язку, автокодувальники також можна навчити за допомогою рециркуляції, алгоритму навчання, заснованого на порівнянні активації мережі на оригінальному вході з активаціями на реконструйованому вході. Рециркуляція вважається більш біологічно правдоподібною, ніж зворотне розповсюдження, але рідко використовується для програм машинного навчання.

8.1. Стискаючі автокодувальники

Копіювання входу на вихід може здатися марним, але зазвичай нас не цікавить вихід декодувальника. Замість цього ми сподіваємося, що навчання автокодувальника виконувати завдання копіювання вхідних даних призведе до того, що h набере корисних властивостей.

Один із способів отримати корисні ознаки від автокодувальника – обмежити розмірність h величиною, меншою за x. Автокодувальник, розмірність коду якого менша за вхідну, називається *стискаючим*. Вивчення неповного представлення змушує автокодувальник захоплювати найбільш помітні характеристики навчальних даних.

Процес навчання описується просто як мінімізація функції втрат

$$L(x, g(f(x))), \tag{8.1}$$

де L є функцією втрат, яка штрафує $g(f(x))$ за відмінність від x, наприклад середньоквадратична помилка.

Коли декодувальник є лінійним і L є середньоквадратичною помилкою, стискаючий автокодувальник навчається охоплювати той самий підпростір, що й метод головних компонент. У цьому випадку автокодувальник, навчений виконувати завдання копіювання, вивчив головний підпростір навчальних даних як побічний ефект.

Автокодувальники з нелінійними функціями f кодувальника і нелінійними функціями g декодувальника можуть вивчати більш потужне нелінійне узагальнення методу головних компонент. На жаль, якщо кодувальнику та декодувальнику надано надто велику ємність, автокодувальник може навчитися виконувати задачу копіювання, не витягуючи корисну інформацію про розподіл даних. Теоретично можна уявити, що автокодувальник з одновимірним кодом, але із дуже потужним нелінійним кодувальником, може навчитися представляти кожен навчальний приклад $x^{(i)}$ за допомогою коду i. Декодувальник може навчитися відображати ці цілочисельні індекси назад до значень конкретних навчальних прикладів. Цей конкретний сценарій не зустрічається на практиці, але він чітко ілюструє, що автокодувальник, навчений виконувати задачу копіювання, може не дізнатися нічого корисного про набір даних, якщо ємність автокодувальника стане занадто великою.

8.2. Регуляризовані автокодувальники

Стискаючі автокодувальники з розмірністю коду, меншою за вхідну, можуть вивчати найважливіші особливості розподілу даних. Як було описано вище, ці автокодувальники не можуть навчитися нічому корисному, якщо кодувальнику та декодеру надано занадто велику ємність.

Подібна проблема виникає, якщо прихованому коду дозволено мати розмірність, що дорівнює вхідній інформації, а також у випадку розтискаючого автокодувальника, коли прихований код має розмірність, більшу за вхідну. У цих випадках навіть лінійний кодувальник і лінійний декодувальник навчитися копіювати вхідні дані на вихід, не вивчивши нічого корисного про розподіл даних.

В ідеалі можна було б успішно навчити будь-яку архітектуру автокодувальника, вибравши розмірність коду та потужність кодувальника та декодувальника на основі складності розподілу, що моделюється. Регуляризовані автокодувальники надають можливість це зробити. Замість того, щоб обмежувати ємність моделі, зберігаючи кодувальник і декодувальник неглибокими, а розмір коду малим, регуляризовані автокодувальники використовують функцію втрат, яка заохочує модель мати інші властивості, окрім здатності копіювати вхідні дані на вихід. Ці інші властивості включають розрідженість подання, малість похідної подання та стійкість до шуму або відсутніх вхідних даних. Регуляризований автокодувальник може бути нелінійним і розтискаючим, але все одно вивчить щось корисне про розподіл даних, навіть якщо ємність моделі достатньо велика, щоб вивчити тривіальну функцію тотожності.

На додаток до описаних тут методів, які найбільш природним чином інтерпретуються як регуляризовані автокодувальники, майже будь-яку генеративну модель із латентними змінними та оснащену процедурою висновку (для обчислення латентних уявлень на вхідних даних) можна розглядати як особливу форму автокодувальника. Два підходи до генеративного моделювання, які підкреслюють цей зв’язок з автокодувальники, є нащадками машини Гельмгольца, наприклад, варіаційний автокодувальник і генеративні стохастичні мережі. Ці моделі природним чином навчаються високоємним, розтискаючим кодуванням вхідних даних і не вимагають регуляризації, щоб ці кодування були корисними. Їхнє кодування природно корисне, оскільки моделі були навчені приблизно максимізувати ймовірність навчальних даних, а не копіювати вхідні дані у вихідні дані.

8.3. Розріджений автокодувальник

Розріджений автокодувальник – це автокодувальник, критерій навчання якого передбачає штраф розрідженості $\Omega(h)$ на кодовому шарі h, на додаток до помилки реконструкції:

$$L\left(x, g\left(f(x)\right)\right) + \Omega(h), \tag{8.2}$$

де $g(h)$ – це вихідний сигнал декодувальника, $h = f(x)$ – вихідний сигнал кодувальника.

Розріджені автокодувальники зазвичай використовуються для вивчення ознак для іншої задачі, наприклад класифікації. Автокодувальник, який було регуляризовано, щоб бути розрідженим, повинен реагувати на унікальні статистичні характеристики набору даних, на якому його було навчено, а не просто діяти як тотожна функція. Таким чином, навчання виконанню задачі копіювання зі штрафом за розрідженість може дати модель, яка вивчила додатково корисні ознаки.

Можна думати про штраф $\Omega(h)$ просто як про член регуляризатора, доданий до мережі прямого зв'язку, головною задачею якої є копіювання вхідних даних на вихід (з цільовою функцією неконтрольованого навчання) і, можливо, також виконання певного контрольованого завдання (з цільовою функцією контрольованого навчання), що залежить від цих розріджених ознак.

Замість того, щоб розглядати штраф розрідженості як регуляризатор для завдання копіювання, розглянемо всю структуру розрідженого автокодувальника як наближене навчання максимальної правдоподібності генеративної моделі, яка має латентні змінні. Припустимо, що ми маємо модель з видимими змінними x і прихованими змінними h, з явним спільним розподілом $p_{model}(x, h) = p_{model}(h)p_{model}(x|h)$. Тоді $p_{model}(h)$ називають апріорним розподілом моделі за латентними змінними, що представляє переконання моделі до того, як побачити x. Логарифм правдоподібності можна розкласти як

$$\log p_{model}(x) = \log \sum_h p_{model}(h, x) \tag{8.3}$$

Тобто автокодувальник наближає цю суму за допомогою точкової оцінки лише для одного дуже ймовірного значення h. Це схоже на генеративну модель розрідженого кодування, але h є виходом параметричного кодувальника, а не результатом оптимізації, яка виводить найбільш імовірний h. З цієї точки зору, з вибраним h ми максимізуємо

$$\log p_{model}(h, x) = \log p_{model}(h) + \log p_{model}(x|h) \tag{8.4}$$

Член $\log p_{model}(h)$ може стимулювати розрідженість. Наприклад, апріорний розподіл Лапсаса,

$$p_{model}(h_i) = \frac{\lambda}{2} e^{-\lambda|h_i|}, \tag{8.5}$$

відповідає розрідженому штрафу з абсолютним значенням. Виражаючи логарифм апріорного розподілу як штраф за абсолютним значенням, отримуємо:

$$\Omega(h) = \lambda \sum_i |h_i|, \tag{8.6}$$

$$-\log p_{model}(h) = \sum_i \left(\lambda|h_i| - \log\frac{\lambda}{2}\right) = \Omega(\mathrm{h}) + const, \tag{8.7}$$

де постійний член залежить лише від λ, а не від h. Зазвичай розглядаємо λ як гіперпараметр і відкидаємо постійний член, оскільки він не впливає на вивчення параметра. Інші розподіли, такі як t-розподіл Стьюдента, також можуть стимулювати розрідженість. З цієї точки зору розрідженості як результату впливу $p_{model}(h)$ на апроксимоване навчання максимальної правдоподібності, штраф за розрідженість взагалі не є членом регуляризації. Це лише наслідок розподілу моделі по її латентним змінним. Ця точка зору забезпечує іншу мотивацію для навчання автокодувальника: це спосіб апроксимованого навчання генеративної моделі. Це також надає іншу причину чому ознаки, отримані автокодувальником, корисні: вони описують латентні змінні, які пояснюють вхід.

Ідея одного із способів досягнення фактичних нулів у h для розріджених автокодувальників (і тих, що усувають шум) полягає в тому, щоб використовувати ReLU для створення шару коду. З апріорним розподілом, що фактично обнуляє представлення (наприклад, штраф абсолютної величини), таким чином можна опосередковано контролювати середню кількість нулів у представленні.

8.4. Автокодувальники, що усуваються шум

Замість того, щоб додавати штраф Ω до функції вартості, можемо отримати автокодувальник, який вивчає щось корисне, змінюючи член помилки реконструкції функції вартості.

Зазвичай, автокодувальник мінімізує певну функцію

$$L\Big(x, g\big(f(x)\big)\Big) \tag{8.8}$$

де L є функцією втрат, яка штрафує $g(f(x))$ за відмінність від x. Однією з функцій втрат може бути середньоквадратична помилка. Це заохочує $g \circ f$ навчитися бути просто тотожною функцією, якщо вони мають на це здатність.

Натомість автокодувальник, що усуває шум, мінімізує

$$L\Big(x, g\big(f(\tilde{x})\big)\Big), \tag{8.9}$$

де $\tilde{x}$ є копією x, яка була пошкоджена певною формою шуму. Таким чином, автокодувальни, що усувають шум, повинні скасовувати це пошкодження, а не просто копіювати вхідні дані.

Усунення шуму змушує f і g неявно вивчати структуру $p_{data}(x)$. Таким чином, автокодувальники, що усувають шум надають є ще одним прикладом того, як корисні властивості можуть з'являтися як побічний продукт мінімізації помилок реконструкції. Вони також є прикладом того, як розтискаючі моделі з великою ємністю можна використовувати як автокодувальники, якщо вжити заходів щодо запобігання вивчання тотожної функції.

8.5. Нормалізація шляхом штрафування похідних

Інша стратегія регуляризації автокодувальника полягає у використанні штрафу Ω, як у розріджених автокодувальниках,

$$L\left(x, g\left(f(x)\right)\right) + \Omega(h, x), \tag{8.10}$$

але з іншою формою Ω:

$$\Omega(h, x) = \lambda \sum_i \|\nabla_x h_i\|^2. \tag{8.11}$$

Це змушує модель вивчати функцію, яка не сильно змінюється, коли x змінюється незначно. Оскільки цей штраф застосовується лише до навчальних прикладів, він змушує автокодувальник вивчати ознаки, які збирають інформацію про навчальний розподіл. Регуляризований таким чином автокодувальник називається контрастним автокодувальником.

8.6. Потужність представлення, розмір шару та глибина

Автокодувальники часто навчаються лише з одношаровим кодувальником і одношаровим декодувальником. Однак це не обов'язкова вимога. Насправді використання глибоких кодувальників та декодувальників дає багато переваг.

Згадайте попередні розділи, що глибина в мережі прямого зв'язку має багато переваг. Оскільки автокодувальники є мережею прямого зв'язку, ці переваги також стосуються їх. Крім того, кодувальник сам є мережею прямого зв'язку, як і декодувальник, тому кожен із цих компонентів автокодувальника може окремо отримати переваги від глибини.

Одна з головних переваг нетривіальної глибини полягає в тому, що універсальна теорема про апроксиматор гарантує, що повнозв'язна нейронна мережа з принаймні одним прихованим шаром може представляти апроксимацію будь-якої функції (в межах широкого класу) з довільним ступенем точності, за умови, що вона має достатньо прихованих нейронів. Це означає, що автокодувальник з одним прихованим шаром здатний як завгодно добре представляти тотожну функцію в області даних. Однак відображення вхідних даних у код неглибоке. Це означає, що ми не можемо застосувати довільні обмеження, наприклад те, що код має бути розрідженим. Глибокий автокодувальник із принаймні одним додатковим прихованим шаром у самому кодувальнику може як завгодно добре апроксимувати будь-яке відображення вхідних даних у код, враховуючи достатню кількість прихованих одиниць.

Глибина може експоненційно зменшити обчислювальні витрати на представлення деяких функцій. Глибина також може експоненційно зменшити кількість навчальних даних, необхідних для вивчення деяких функцій. Експериментально глибокі автокодувальники дають набагато краще стиснення, ніж відповідні неглибокі або лінійні автокодувальники.

Далі давайте розглянемо інші типи нейронних мереж, що дозволяють відновлювати сигнал, а саме – мережі з асоціативною пам’яттю.

8.7. Мережі з асоціативної пам’яттю. Мережі Хопфілда та Хеммінга

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

Мережа Хопфілда використовує три прошарки: вхідний, прошарок Хопфілда та вихідний прошарок (рисунок 8.2). Кожен прошарок має однакову кількість нейронів. Входи прошарку Хопфілда під'єднані до виходів відповідних нейронів вхідного прошарку через змінні ваги з'єднань. Виходи прошарку Хопфілда під'єднуються до входів всіх нейронів прошарку Хопфілда, за винятком самого себе, а також до відповідних елементів у вихідному прошарку. В режимі функціонування, мережа скеровує дані з вхідного прошарку через фіксовані ваги з'єднань до прошарку Хопфілда. Прошарок Хопфілда коливається, поки не буде завершена певна кількість циклів, і змінний стан прошарку передається на вихідний прошарок. Цей стан відповідає образу, вже запрограмованому у мережу.

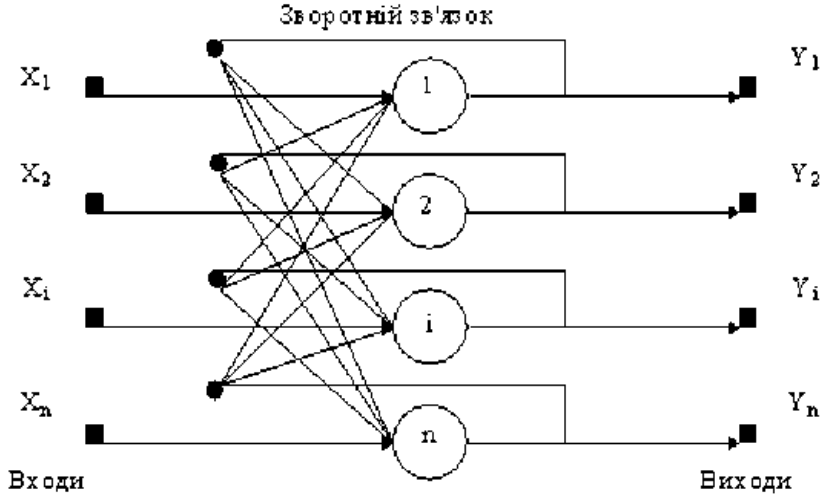


Рис. 8.2. Структурна схема мережі Хопфілда.

Навчання мережі Хопфілда вимагає, щоб навчальний образ був представлений на вхідному та вихідному прошарках одночасно. Рекурсивний характер прошарку Хопфілда забезпечує засоби корекції всіх вагів з'єднань. Недвійкова реалізація мережі повинна мати пороговий механізм у передатній функції. Для правильного навчання мережі відповідні пари "вхід-вихід" мають відрізнятися між собою.

Якщо мережа Хопфілда використовується як пам'ять, що адресується за змістом, вона має два головних обмеження. По-перше, число образів, що можуть бути збережені та точно відтворені є строго обмеженим. Якщо зберігається занадто багато параметрів, мережа може збігатись до нового неіснуючого образу, відмінного від всіх запрограмованих образів, або не збігатись взагалі. Межа ємності пам'яті для мережі приблизно 15% від числа нейронів у прошарку Хопфілда. Другим обмеженням парадигми є те, що прошарок Хопфілда може стати нестабільним, якщо навчальні приклади є занадто подібними. Зразок образу вважається нестабільним, якщо він застосовується за нульовий час і мережа збігається до деякого іншого образу з навчальної множини. Ця проблема може бути вирішена вибором навчальних прикладів більш ортогональних між собою.

Задача, розв'язувана даною мережею в якості асоціативної пам'яті, як правило, формулюється таким чином. Відомий деякий набір двійкових сигналів (зображень, звукових оцифровок, інших даних, що описують якийсь об'єкти або характеристики процесів), вважають зразковим. Мережа повинна вміти з зашумленого сигналу, поданого на її вхід, виділити («пригадати» по частковій інформації) відповідний зразок або «дати висновок» про те, що вхідні дані не відповідають жодному із зразків. У загальному випадку, будь-який сигнал може бути описаний вектором $x_1,\dots,x_i,\dots,x_n$, де n – число нейронів у мережі і величина вхідних і вихідних векторів. Кожний елемент x_i дорівнює або $+1$, або -1. Позначимо вектор, що описує k-*тий* зразок, через X^k, а його компоненти, відповідно, $x_i^k, \quad k=\overline{1,m}$, де m – число зразків. Якщо мережа розпізнає (або «пригадує») якийсь зразок на основі пред'явлених їй даних, її виходи будуть містити саме його, тобто $Y=X$, де Y – вектор вихідних значень мережі: $y_1,\dots,y_i,\dots,y_n$. У противному випадку, вихідний вектор не співпаде з жодний зразковим.

Якщо, наприклад, сигнали являють собою якесь зображення, то, відобразивши у графічному виді дані з виходу мережі, можна буде побачити картинку, що цілком збігається з однієї зі зразкових (у випадку успіху) або ж «вільну імпровізацію» мережі (у випадку невдачі).

Алгоритм 8.1. Функціонування мережі Хопфілда.

1. На стадії ініціалізації мережі вагові коефіцієнти прихованого шару встановлюються таким чином:

$$w_{ij}=\begin{cases}\sum_{k=1}^{m} x_i^k x_j^k, & if\ i\neq j\\ 0, & if\ i=j\end{cases}, \tag{8.12}$$

де i та j – індекси нейронів відповідно, вихідного і прихованого шарів; x_i^k та x_j^k – це i-тий та j-тий елементи вектора k-того зразка.

2. На входи мережі подається невідомий сигнал (t – номер ітерації). Його поширення безпосередньо встановлює значення виходів:

$$y_i(0)=x_i, \quad i=\overline{1,n}, \tag{8.13}$$

тому позначення на схемі мережі вхідних сигналів у явному виді носить чисто умовний характер. Нуль у дужці справа від y_i означає нульову ітерацію в циклі роботи мережі.

3. Розраховується новий стан нейронів

$$S(t+1) = \sum_{i=1}^{n} w_{ij} y_j(t), \quad j = \overline{1,n} \tag{8.14}$$

і нові значення виходів

$$y(t+1) = f\big(S(t+1)\big), \tag{8.15}$$

де f – порогова передатна функція, яка може мати вигляд однієї з представлених на рисунку 8.3.

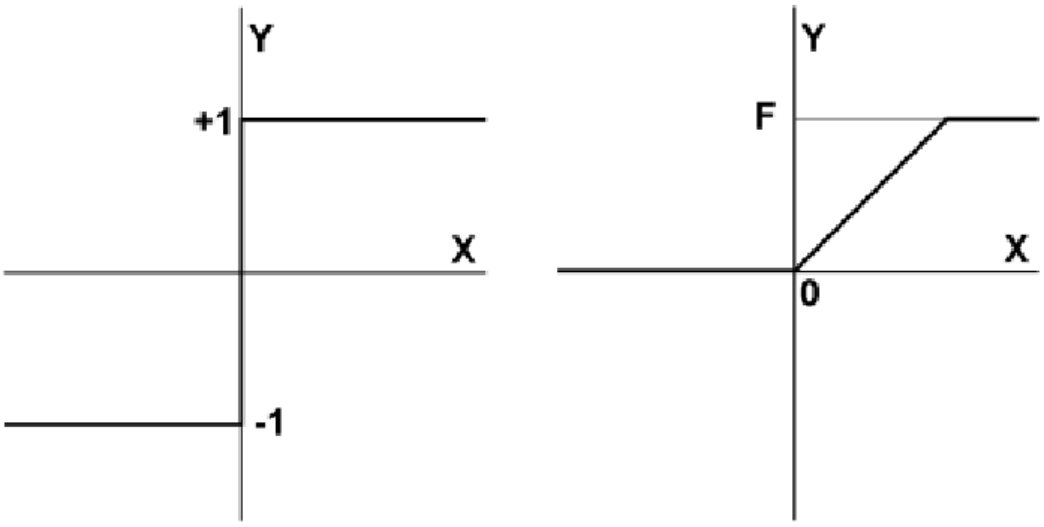


Рис. 8.3. Передатні функції прихованих нейронів мережі Хопфілда.

4. Перевіряється, чи змінилися вихідні значення виходів за останню ітерацію. Якщо так – перехід до пункту 2, інакше (якщо виходи стабілізувались) – кінець. При цьому вихідний вектор являє собою зразок, що найкраще відповідає вхідним даним.

Мережа Хеммінга є розширенням мережі Хопфілда. Ця мережа була розроблена Річардом Ліппманом у середині 80-х рр. Мережа Хемінга реалізує класифікатор, що базується на найменшій похибці для векторів двійкових входів, де похибка визначається відстанню Хемінга.

Відстань Хемінга визначається як число бітів, які відрізняються між двома відповідними вхідними векторами фіксованої довжини. Один вхідний вектор є незашумленим прикладом образу, інший є спотвореним образом. Вектор виходів навчальної множини є вектором класів, до яких належать образи. У режимі навчання вхідні вектори розподіляються до категорії, для яких відстань між зразковим вхідним вектором та поточним вхідним вектором є мінімальною.

Мережа Хеммінга має три прошарки:

– вхідний шар з кількістю вузлів, що дорівнює кількості окремих двійкових ознак;
– шар категорій (шар Хопфілда), з кількістю вузлів, що дорівнює кількості категорій або класів;
– вихідний шар, потужність якого відповідає числу вузлів у шарі категорій.

Мережа є простою архітектурою прямого поширення з вхідним шаром, повністю під'єднаним до шару категорій – рисунок 8.4. Кожен елемент у шарі категорій є зворотно під'єднаним до кожного нейрона у тому ж самому шарі й прямо під'єднаним до вихідного нейрону. Вихід з шару категорій до вихідного шару формується через конкуренцію.

Навчання мережі Хеммінга є подібним до методології Хопфілда. На вхідний шар надходить бажаний навчальний образ, а на виході вихідного шару надходить значення бажаного класу, до якого належить вектор. Вихід містить лише значення класу до якого належить вхідний вектор. Рекурсивний характер прошарку Хопфілда забезпечує збіжність корекції всіх вагових коефіцієнтів.

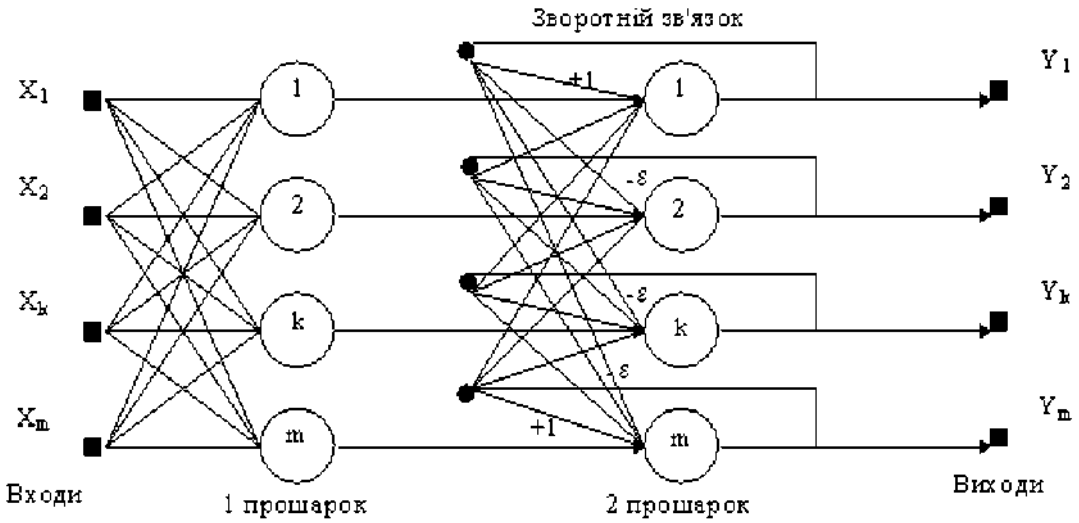


Рис. 8.4. Структурна схема мережі Хеммінга.

Алгоритм 8.2. Функціонування мережі Хеммінга.

Крок 1. На стадії ініціалізації ваговим коефіцієнтам першого прошарку і порогу передатної функції присвоюються такі значення:

$W_{ik} = x_i^k/2,\ i = \overline{1,n},\ k = \overline{1,m};$

$b_k = n/2,\ k = \overline{1,m}.$

Тут x_i^k – i-тий елемент k-того зразка.

Вагові коефіцієнти гальмуючих синапсів у другому шарі беруть рівними деякій величині $0 < \varepsilon < 1/m$ з від’ємним знаком. Синапс нейрона, пов'язаний із його ж виходом має вагу +1.

Крок 2. На входи мережі подається невідомий вектор $X = (x_1; \dots; x_i; \dots; x_n)$. Розраховуються стани нейронів першого прошарку (верхній індекс у дужках вказує номер прошарку):

$$y_j^1 = S_j^1 = \sum_{i=1}^{n} w_{ij} x_i + b_j, \quad j = \overline{1, m} \tag{8.16}$$

Після цього отримані значення ініціалізують значення виходів другого шару.

Крок 3. Обчислюються нові стани нейронів другого прошарку:

$$S_j^2(t+1) = y_j^2(t) - \varepsilon \sum_{k=1}^{m} y_k^2(t), \quad r \neq j, \quad j = \overline{1, m} \tag{8.17}$$

і значення їх виходів:

$$y_j^1 = f\left|S_j^2(t+1)\right|, \quad j = \overline{1, m}. \tag{8.18}$$

Передатна функція f має вид порога, причому величина b_j повинна бути достатньо великою, щоб будь-які можливі значення аргументу не призводили до насичення.

Крок 4. Перевіряється, чи змінилися виходи нейронів другого шару за останню ітерацію. Якщо так – перейти до кроку 3. Інакше – кінець.

Роль першого шару є умовною: скориставшись один раз на першому кроці значеннями його вагових коефіцієнтів, мережа більше не повертається до нього, тому перший шар може бути взагалі виключений із мережі.

Мережа Хеммінга має ряд переваг над мережею Хопфілда. Вона реалізує оптимальний класифікатор за критерієм мінімуму похибки, якщо похибки вхідних бітів є випадковими та незалежними. Для функціонування мережі Хеммінга потрібна менша кількість нейронів, оскільки середній прошарок вимагає лише один нейрон на клас, замість нейрону на кожен вхідний вузол. І, нарешті, мережа Хеммінга не страждає від неправильних класифікацій, які можуть трапитись у мережі Хопфілда. В цілому, мережа Хеммінга є як швидшою, так і точнішою за мережу Хопфілда.

Двоскерована асоціативна пам'ять була розроблена Бартом Козко і розширює модель Хопфілда. Множина парних образів навчається за образами, що представлені як біполярні вектори. Подібно до мережі Хопфілда, коли на вхід подається зашумлена версія якогось образу, визначається найближчий образ, асоційований з ним.

На рисунку 8.5 показаний приклад двоскерованої асоціативної пам'яті. Вона має стільки входів, скільки є вихідних нейронів. Два приховані шари реалізують окремі вектори пам’яті кожен з яких відповідає за розміром вхідним векторам. Середні шари повністю з'єднуються один з одним. Вхідний та вихідний шари потрібні для реалізації засобів введення та відновлення інформації з мережі.

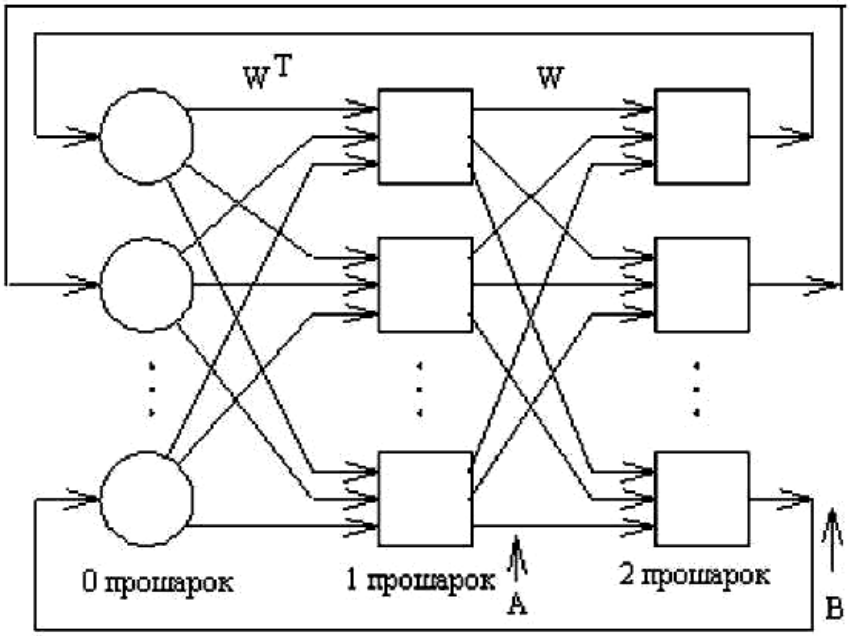


Рис. 8.5. Двоскерована асоціативна пам'ять.

Середні шари розроблені для збереження асоційованих пар векторів. Коли зашумлений вектор образу вважається вхідним, середні шари коливаються до досягнення стабільного стану рівноваги, який відповідає найближчій навченій асоціації і буде генерувати початковий навчальний образ на виході. Подібно до мережі Хопфілда, двоскерована асоціативна пам'ять є схильною до неправильного відшукування навченого образу, якщо надходить невідомий вхідний вектор, який не був у складі навчальної множини.

Двоскерована асоціативна пам'ять відноситься до гетероасоціативної пам'яті. Вхідний вектор надходить на один набір нейронів, а відповідний вихідний вектор продукується на іншому наборі нейронів. Вхідні образи асоціюються з вихідними.

Для порівняння: мережа Хопфілда є автоасоціативною. Вхідний образ може бути відновлений чи виправлений мережею, але не може бути асоційований з іншим образом. У мережі Хопфілда використовується одношарова структура асоціативної пам'яті, у якій вихідний вектор з'являється на виході тих же нейронів, на які надходить вхідний вектор.

Двоскерована асоціативна пам'ять, як і мережа Хопфілда, здатна до узагальнення, виробляючи правильні вихідні сигнали, незважаючи на спотворені входи.

Розглянемо схему двоскерованої асоціативної пам'яті. Вхідний вектор A обробляється матрицею вагових коефіцієнтів W мережі, у результаті чого продукується вектор вихідних сигналів мережі B. Вектор B обробляється транспонованою матрицею W^T вагових коефіцієнтів мережі, яка продукує сигнали, що представляють новий вхідний вектор A. Цей процес повторюється

доти, поки мережа не досягне стабільного стану, у якому ані вектор A, ані вектор B не змінюються.

Нейрони в шарах 1 і 2 функціонують, як і в інших парадигмах, обчислюючи суму зважених входів і значення передатної функції *F:*

$$b_j = F\left(\sum_{i=1}^{n} w_{ij} a_i\right), \quad i,j = \overline{1,n}\,, \tag{8.19}$$

або у векторній формі:

$$B = F(A \cdot W)\,, \tag{8.20}$$

де B – вектор вихідних сигналів нейронів шару 2, A – вектор вихідних сигналів нейронів шару 1, W – матриця вагових коефіцієнтів зв'язків між шарами 1 і 2, F – передатна функція.

Аналогічно

$$A = F(B \cdot W^T)\,, \tag{8.21}$$

де W^T є транспозицією матриці W.

В якості передатної функції використовується експонентна сигмоїда.

При цьому шар 0 не робить обчислень і не має пам'яті. Він є лише засобом розподілу вихідних сигналів шару 2 до елементів матриці W^T.

Формула для обчислення значень синаптичних вагових коефіцієнтів при навчанні має вигляд:

$$W = \sum_j A_j^T \cdot B\,, \tag{8.22}$$

де A_j і B_j – відповідно вхідні і вихідні вектори навчальної вибірки.

Таким чином, вагова матриця обчислюється як сума добутків всіх векторних пар навчальної вибірки.

Системи зі зворотним зв'язком мають тенденцію до коливань. Вони можуть переходити від стану до стану, ніколи не досягаючи стабільності. Доведено, що двоскерована асоціативна пам'ять безумовно стабільна при будь-яких значеннях вагових коефіцієнтів мережі.

Головні області застосування ДАП: моделювання асоціативної пам'яті та розпізнавання образів.

Ключові недоліки:

1) Ємність двоскерованої асоціативної пам'яті жорстко обмежена. Якщо n – кількість нейронів у вхідному прошарку, то число векторів, що можуть бути запам'ятовано в мережі не перевищує $m = \frac{n}{2\log_2 n}$. Таким чином, якщо n=1024, то мережа здатна запам'ятати не більш m=25 образів, кожний з яких повинен бути відновлюваним.

2) Двоскерована асоціативна пам'ять має деяку непередбачуваність у процесі функціонування, тобто можливі помилкові відповіді.

Ключові переваги:

1) У порівнянні з автоасоціативною пам'яттю (наприклад, мережею Хопфілда), двоскерована асоціативна пам'ять дає можливість будувати асоціації між векторами A і B, що у загальному випадку мають різні розмірності. За рахунок таких можливостей гетероасоціативна пам'ять має більш широкий клас застосувань, ніж автоасоціативна пам'ять.

2) Процес формування синаптичних вагових коефіцієнтів простий і швидкий. Мережа швидко збігається в процесі функціонування.

8.8. Контрольні питання

1. Які функції виконують кодувальник і декодувальник в мережі автокодувальнику?
2. Які існують способи боротьби з вивченням автокодувальником простого копіювання?
3. Яку функцію виконує внутрішнє представлення (код) h?
4. В чому особливість автокодувальників, що усувають шум?
5. Чи може існувати автокодувальник з більше, ніж одним прихованим шаром? Відповідь обґрунтуйте.
6. Як автокодувальники вирішують задачу відновлення сигналу?
7. В чому полягає ідея мереж з асоціативною пам’яттю?
8. Які ви знаєте відмінності мереж Хопфілда та Хеммінга?

Лекція 9 – Прикладні задачі машинного навчання

9.1. Огляд сфер застосування штучного інтелекту

Штучний інтелект (ШІ) став невід'ємною частиною сучасного суспільства, трансформуючи індустрії та покращуючи різні аспекти повсякденного життя за допомогою безлічі інноваційних програм. Здатність ШІ обробляти великі обсяги даних, розпізнавати закономірності та робити прогнози відкрила нові можливості в багатьох сферах.

Медицина. У секторі охорони здоров'я штучний інтелект застосовується у діагностиці, лікування та лікування хвороб. Алгоритми машинного навчання можуть аналізувати величезну кількість медичних даних, у тому числі історії пацієнтів, результати візуалізації та генетичну інформацію, щоб допомогти у ранньому виявленні захворювань і персоналізованих планах лікування. Наприклад, системи на основі штучного інтелекту можуть допомогти радіологам виявити тонкі аномалії на медичних зображеннях, які можуть вказувати на наявність раку чи інших захворювань. Ці системи не тільки підвищують точність діагностики, але й дозволяють швидше та ефективніше обслуговувати пацієнтів.

Ліки. ШІ є також важливим у відкритті та розробці ліків. Моделюючи молекулярні взаємодії та прогнозуючи ефективність потенційних ліків, штучний інтелект може прискорити процес дослідження та скоротити час і витрати, пов'язані з виведенням нових ліків на ринок. Крім того, роботи, керовані ШІ, використовуються для виконання складних хірургічних процедур з більшою точністю та меншою кількістю ускладнень, ніж традиційні методи.

Фінанси. Фінансова індустрія прийняла ШІ для підвищення ефективності, безпеки та обслуговування клієнтів. Алгоритми штучного інтелекту можуть аналізувати фінансові дані для виявлення шахрайства в реальному часі, допомагаючи банкам і фінансовим установам захистити активи своїх клієнтів. Моделі машинного навчання також використовуються для кредитного скорингу та оцінки ризиків, що дозволяє точніше оцінювати заявників на позику та зменшує ймовірність дефолту.

Машинний переклад. Сучасні системи машинного перекладу перейшли на використання нейронних мереж, що дозволило суттєво покращити якість перекладу. Наприклад, щоб використовувати мережі довготривалої короткочасної пам'яті (LSTM) для машинного перекладу, необхідно розробити архітектуру, яка використовує переваги послідовного характеру мовних даних. Використання кодувальника-декодувальника є популярним підходом.

Кодувальник відповідає за читання вхідного речення вихідною мовою та кодування його в безперервне векторне представлення. Цього можна досягти, подаючи вхідне речення по одному слову в мережу LSTM, яка виведе послідовність векторів, що представляють закодований вхід. Потім декодувальник використовує це закодоване представлення для створення

перекладу цільовою мовою. Декодувальник також є мережею LSTM, яка приймає закодований вхід і генерує вихідну послідовність по одному слову за раз.

Під час навчання модель навчається на парах речень вихідною та цільовою мовами, де метою є передбачити наступне слово в цільовому реченні з огляду на контекст, наданий кодувальником.

З точки зору конкретних деталей реалізації, можливе використання архітектури LSTM «багато-до-багатьох», де вхідна послідовність подається в кодувальник LSTM, а вихідна послідовність генерується декодувальником LSTM. Остаточний прихований стан кодера LSTM часто використовується в якості початкового прихованого стану для декодера LSTM, як це показано на рис. 9.1.

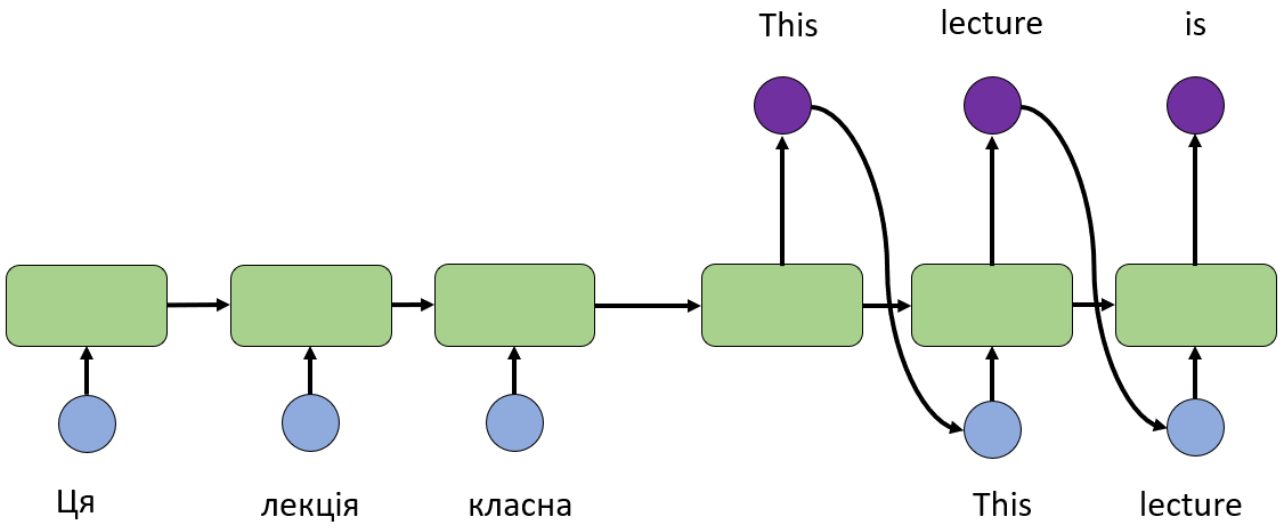


Рис. 9.1. Машинний переклад із використання мережі LSTM.

Також можна додати додаткові компоненти, такі як механізми уваги, щоб допомогти моделі зосередитися на певних частинах вхідного речення під час створення перекладу. Вибір функцій активації, кількості шарів та інших гіперпараметрів залежатиме від конкретної задачі та набору даних, що використовується.

Чат-боти. Чат-боти на базі штучного інтелекту стали загальноприйнятою частиною банківських та інвестиційних послуг, надаючи персоналізовану допомогу клієнтам цілодобово. Ці віртуальні помічники можуть виконувати широкий спектр завдань, від відповідей на базові запити до виконання транзакцій, звільняючи людей, щоб зосередитися на більш складних питаннях. Крім того, штучний інтелект використовується для розробки алгоритмічних торгових стратегій, які можуть швидко аналізувати ринкові тенденції та здійснювати операції з мінімальним втручанням людини.

Автономні транспортні засоби. ШІ є важливою складовою інновацій у транспорті, зокрема в розробці автономних транспортних засобів. Безпілотні автомобілі покладаються на вдосконалені системи штучного інтелекту, щоб сприймати навколишнє середовище, приймати рішення в режимі реального часу

та безпечно керувати собою. Ці технології можуть значно зменшити кількість дорожньо-транспортних пригод, зменшити затори та підвищити мобільність людей з обмеженими можливостями.

На рис. 9.2. показано автономний автомобіль компанії Google, а на рис. 9.3. показано як такий автомобіль «сприймає» навколишнє середовище. Як можна побачити, будується тривимірна реконструкція середовища, із виділеною дорожньою розміткою, іншими транспортними засобами (наприклад, велосипед справа), знаками дорожнього руху. В сучасних системах реконструкція будується із використанням нейронних мереж. Використовуючи дану реконструкцію системи автомобіля приймають рішення про подальші дії.

Рис. 9.2. Автомобіль Google із обладнанням для системи автономного водіння.

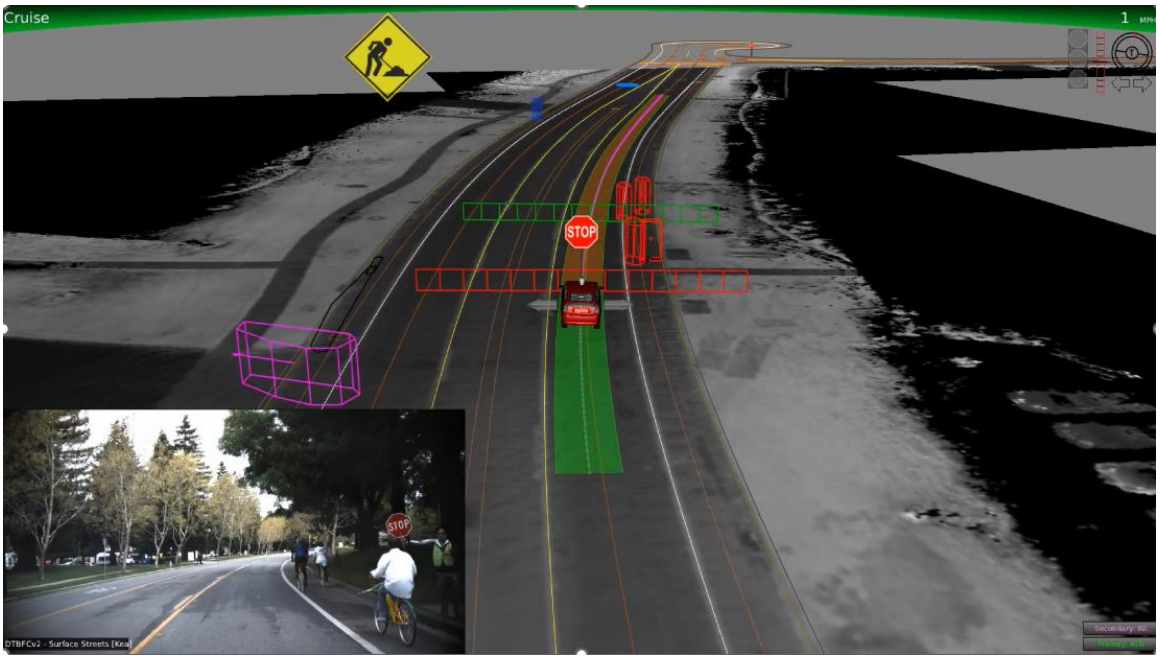


Рис. 9.3 – Візуалізація сприйняття навколишнього середовища системою комп’ютерного зору автономного автомобіля.

Авіація. В авіації штучний інтелект використовується для оптимізації маршрутів польотів, підвищення ефективності використання палива та підвищення безпеки шляхом прогнозного обслуговування компонентів літака. Системи управління повітряним рухом також отримують користь від штучного інтелекту, який може допомогти керувати складним повітряним простором і забезпечити безперебійну роботу навіть у часи пік. Крім того, логістичні рішення на основі штучного інтелекту трансформують управління ланцюгом поставок шляхом оптимізації рівня запасів, оптимізації маршрутів доставки та підвищення загальної ефективності.

Управління запасами. ШІ також використовується для управління запасами та прогнозування попиту. Аналізуючи дані про продажі та зовнішні фактори, такі як погода чи тенденції в соціальних мережах, системи штучного інтелекту можуть з високою точністю прогнозувати майбутній попит. Це допомагає роздрібним торговцям оптимізувати рівень своїх запасів, зменшуючи ризик надмірних запасів або вичерпання популярних товарів. Крім того, на складах розгортаються роботи на базі штучного інтелекту для автоматизації таких завдань, як підбір замовлень і сортування, що ще більше підвищує ефективність і знижує витрати.

Підтримання здорового раціону. Використовуючи нейронні мережі можливо аналізувати зображення їжі для оцінки їх калорійності. Зображення харчових продуктів подаються в модель машинного навчання, як правило, у згорткову нейронну мережу, яка навчається на великому наборі даних зображень харчових продуктів із мітками. Мережа вчиться виділяти релевантні характеристики із зображень і пов’язувати їх із певним діапазоном калорій, що дозволяє передбачати калорійність нових введених зображень. Ця технологія має численні застосування в таких сферах, як планування харчування, відстеження їжі та системи рекомендацій щодо харчових продуктів, що робить її цінним інструментом для людей, які прагнуть стежити за своїм раціоном. На рис. 9.4. показано інтерфейс одного з інструментів для стеження за раціоном харчування.

Освіта. ШІ змінює освіту, персоналізуючи навчальний досвід і надаючи цінну інформацію про успішність учнів. Інтелектуальні системи навчання використовують алгоритми штучного інтелекту для адаптації навчального контенту відповідно до індивідуальних потреб учня та темпу навчання. Ці системи можуть забезпечити негайний зворотний зв’язок, пропонувати додаткові ресурси для складних тем і відстежувати прогрес з часом.

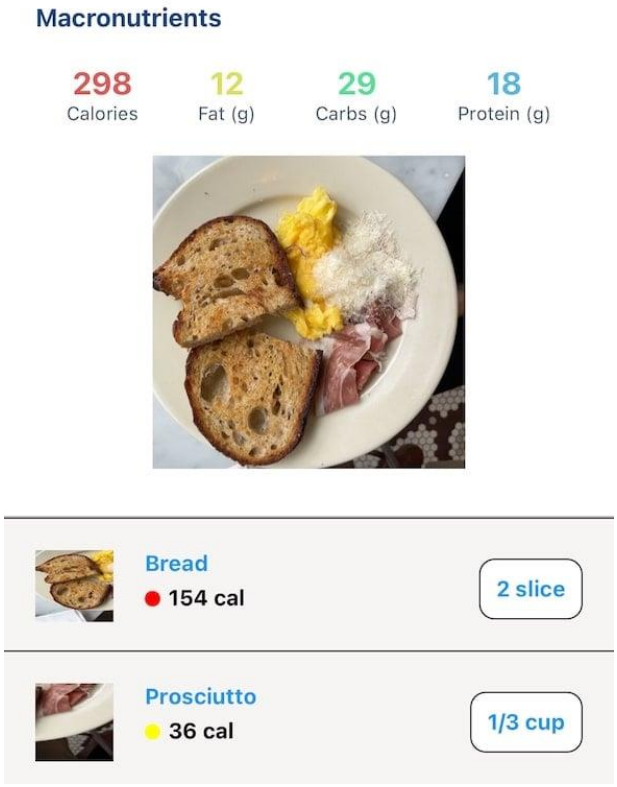


Рис. 9.4. Використання нейронних мереж для визначення калорій страви.

Технології обробки природної мови використовуються для розробки інтелектуальних платформ для вивчення мов, які допомагають учням покращити свої навички читання, письма та мовлення різними мовами. Інструменти аналітики на основі штучного інтелекту дають викладачам можливість отримати уявлення про залучення студентів, виявити прогалини в знаннях і відповідно адаптувати стратегії навчання.

Розваги. Індустрія розваг використовує штучний інтелект для покращення створення контенту. Алгоритми ШІ можуть аналізувати вподобання користувачів і звички перегляду, щоб надавати персоналізовані рекомендації щодо фільмів, телешоу, музики та інших форм медіа. Це не тільки покращує взаємодію з користувачем, але й допомагає постачальникам контенту оптимізувати свої пропозиції та підвищити залучення.

Штучний інтелект також використовується в розробці відеоігор для створення більш динамічного та чутливого ігрового середовища. Неігрові персонажі (NPC) на основі штучного інтелекту можуть демонструвати складну поведінку, адаптуватися до дій гравця та забезпечувати більш захоплюючий ігровий досвід. Крім того, інструменти на основі штучного інтелекту допомагають творцям створювати реалістичну анімацію, віртуальні світи та навіть писати музику, розсуваючи межі художнього вираження.

Таким чином, програми штучного інтелекту охоплюють широкий спектр галузей, змінюючи наш підхід до охорони здоров'я, фінансів, транспорту, роздрібної торгівлі, освіти та розваг. Від підвищення діагностичної точності в медицині до оптимізації ланцюгів постачання в логістиці, здатність ШІ обробляти дані, розпізнавати закономірності та приймати розумні рішення сприяє інноваціям і покращує якість життя людей у всьому світі. Оскільки штучний інтелект продовжує розвиватися, його вплив на різні сектори, безсумнівно, зростатиме, формуючи майбутнє, де технології та людська винахідливість бездоганно співпрацюватимуть разом.

9.2. Проблеми штучного інтелекту

Хоча штучний інтелект продемонстрував величезний потенціал у багатьох галузях промисловості, його практичне застосування не позбавлене викликів і проблем. Під час впровадження штучного інтелекту в реальних сценаріях виникає кілька проблем, пов'язаних із технічними обмеженнями, етичними міркуваннями та наслідками для суспільства.

Технічні проблеми. Однією з основних проблем практичного застосування ШІ є якість і доступність даних. Моделі штучного інтелекту значною мірою покладаються на величезну кількість даних, щоб навчатися та робити точні прогнози. Однак реальні дані часто є неповними, упередженими або містять помилки, що призводить до низької продуктивності або спотворення результатів. Наприклад, системі розпізнавання облич, яка навчається переважно на зображеннях людей зі світлою шкірою, може бути важко точно ідентифікувати людей із темнішими тонами шкіри.

Крім того, моделі штучного інтелекту можуть бути вразливими до змагальних атак, коли вхідними даними навмисно маніпулюють, щоб змусити модель робити неправильні прогнози або приймати рішення. Ця вразливість створює значні ризики, особливо в таких критичних програмах, як автономні транспортні засоби або кібербезпека, де наслідки збою системи можуть бути серйозними.

Етика. З точки зору етики однією з головних проблем є питання конфіденційності та спостереження. Системи штучного інтелекту часто вимагають доступу до конфіденційних персональних даних, що викликає питання про те, як ця інформація збирається, зберігається та використовується. Наприклад, інтелектуальні поліцейські інструменти на основі штучного інтелекту критикували за те, що вони потенційно порушують права особи та посилюють існуючі упередження в правоохоронних органах.

Крім того, відсутність прозорості в багатьох моделях штучного інтелекту, особливо в тих, що базуються на складних нейронних мережах, викликає занепокоєння щодо підзвітності. Коли система штучного інтелекту приймає рішення, яке має значні наслідки, як-от відмова у кредитній заявці або рекомендація лікування, надзвичайно важливо зрозуміти, як і чому було прийнято

таке рішення. Однак «чорна скринька» багатьох алгоритмів штучного інтелекту ускладнює відстеження причин, що стоять за їхніми результатами, що призводить до питань про те, хто несе відповідальність, коли щось йде не так.

Вплив на суспільство. Широке впровадження штучного інтелекту також викликає занепокоєння щодо переміщення робочих місць та економічної нерівності. Автоматизація на основі штучного інтелекту може призвести до значної втрати робочих місць у секторах, де завдання повторюються або легко автоматизуються, наприклад у виробництві, транспортуванні та обслуговуванні клієнтів. Хоча нові робочі місця можуть бути створені в таких сферах, як розробка та обслуговування ШІ, перехід може бути складним для тих, чиї навички застаріли.

Крім того, переваги ШІ нерівномірно розподілені в суспільстві. Ті, хто має доступ до передових технологій і ресурсів, мають більше шансів пожинати плоди, тоді як маргіналізовані спільноти можуть зіткнутися з подальшим виключенням. Цей цифровий розрив може посилити існуючу нерівність, створивши прірву між тими, хто може використовувати штучний інтелект для отримання економічної вигоди, і тими, кого залишив позаду технологічний прогрес.

Нормативно-правові питання. Швидкий розвиток штучного інтелекту випередив розвиток нормативно-правової бази, яка регулює його використання. Відсутність чітких інструкцій створює невизначеність для компаній і організацій, які прагнуть запровадити рішення штучного інтелекту, а також для людей, на чиє життя впливають ці технології. Наприклад, тривають дебати про те, як слід призначати відповідальність, якщо автономний транспортний засіб потрапив у аварію або коли рішення, кероване штучним інтелектом, завдає шкоди.

Крім того, міжнародний характер розробки та впровадження ШІ створює проблеми для регулювання. Різні країни мають різні підходи до конфіденційності даних, інтелектуальної власності та етичних стандартів, що ускладнює встановлення узгоджених глобальних інструкцій щодо використання ШІ.

Вирішення цих проблем. Щоб пом’якшити ці проблеми, потрібен багатогранний підхід. Це включає інвестиції в дослідження для підвищення надійності та прозорості моделей штучного інтелекту, розробку етичних рамок, які віддають пріоритет чесності та підзвітності, а також створення політики, яка підтримує справедливий доступ до технологій і захищає індивідуальні права. Крім того, заохочення міжнародного співробітництва може допомогти встановити загальні стандарти та методи управління ШІ.

Підсумовуючи, хоча штучний інтелект має величезні перспективи для трансформації різних галузей промисловості та покращення якості життя, його практичне застосування пов’язане з технічними, етичними, соціальними, нормативними та юридичними проблемами. Вирішення цих проблем вимагатиме узгоджених зусиль дослідників, політиків, лідерів галузі та широкої громадськості, щоб забезпечити відповідальне та справедливе використання переваг ШІ.

9.3. Штучний інтелект в освіті та академічна доброчесність

Генеративний штучний інтелект, який включає такі моделі, як мовні моделі, інструменти синтезу зображень та інші системи генерації творчого контенту, створює низку проблем при застосуванні в освітніх контекстах. Ці занепокоєння охоплюють від академічної доброчесності до якості навчального досвіду та потенціалу зловживання.

Одне з найгостріших питань – це вплив на академічну доброчесність. Генеративний ШІ може створювати есе, звіти та інші письмові завдання, які важко відрізнити від роботи, написаної людьми. Ця можливість викликає серйозне занепокоєння щодо плагіату. Студенти можуть використовувати ці інструменти для виконання завдань без справжнього втручання в матеріал або розвитку власного розуміння, підриваючи навчальний процес. Задача полягає в тому, щоб студенти несли відповідальність за власне навчання, а також усвідомлювали потенційні переваги письма за допомогою штучного інтелекту як інструменту для розвитку навичок, а не як скорочення.

Якість і релевантність вмісту. Хоча генеративний штучний інтелект може давати вражаючі результати, якість і релевантність контенту, який він генерує, не завжди можуть відповідати освітнім стандартам. Інформація, яку надають ці моделі, ґрунтується на шаблонах, отриманих із величезної кількості даних, що може призвести до неточностей, застарілої інформації або упереджених точок зору. Необхідно бути обережним, покладаючись виключно на матеріали, створені штучним інтелектом, для навчання, оскільки вони можуть не точно відображати найновіші дослідження або тонке розуміння складних тем. Крім того, генеративні моделі можуть не мати можливості надавати персоналізований і контекстуально відповідний контент, який узгоджується з конкретними навчальними цілями або потребами студентів.

Результати навчання та розвиток навичок. Використання генеративного ШІ в освіті викликає питання щодо впливу на результати навчання та розвиток навичок. Традиційна освітня практика часто наголошує на критичному мисленні, креативності та навичках вирішення проблем – здібностях, які важко розвинути, якщо учні надто покладаються на ШІ для створення контенту. Існує ризик того, що надмірне використання цих інструментів може призвести до зниження важливих когнітивних і аналітичних навичок, оскільки учні можуть не глибоко залучитися до матеріалу або практикувати процеси, необхідні для самостійного мислення та висловлювання. Збалансування використання генеративного ШІ з діяльністю, яка сприяє активному навчанню та розвитку навичок, має вирішальне значення, щоб уникнути цієї пастки.

Зловживання ШІ. Потенційне зловживання генеративним штучним інтелектом в освіті виходить за межі академічної нечесності до ширших питань залежності та покладення на технології. Студенти, які стають надмірно залежними від штучного інтелекту у своїй роботі, можуть мати проблеми із завданнями, які вимагають оригінального мислення, креативності або вирішення

проблем без технічної допомоги. Ця залежність може перешкодити їхній здатності функціонувати незалежно в майбутньому освітньому або професійному середовищі. Крім того, зловживання генеративними інструментами для ненавчальних цілей, наприклад створення неправдивої інформації або шкідливого вмісту, створює додаткові проблеми для викладачів і адміністраторів у підтримці безпечного та відповідального навчального середовища.

Щоб вирішити ці проблеми, важливо розробити чітку політику та рекомендації щодо використання генеративного ШІ в освіті. Це включає встановлення стандартів академічної доброчесності, забезпечення навчання та ресурсів для викладачів для ефективної оцінки та інтеграції інструментів штучного інтелекту, а також сприяння збалансованому підходу, який використовує технології, зберігаючи при цьому основні цінності освіти. Співпраця між освітянами, дослідниками, політиками та постачальниками технологій необхідна для створення освітньої екосистеми, яка максимізує переваги генеративного штучного інтелекту та одночасно пом’якшує його ризики.

9.4. Контрольні питання

1. Проведіть самостійне дослідження додаткових прикладів використання нейронних мереж.
2. Які архітектури нейронних мереж, вивчених раніше, ви могли би використати для кожної з задач?
3. Як би вирішення проблем штучного інтелекту запропонували би ви?
4. Як LSTM є застосовним до машинного перекладу?
5. Які архітектури нейронних мереж ви би використали для вирішення задачі визначення калорійності страви?
6. Яка ваша думка, кому належить авторське право на твір, створений штучним інтелектом?
7. В чому полягає небезпека надмірного використання генеративного штучного інтелекту в освіті, зокрема в студентських роботах?
8. Які принципи дотримання академічної доброчесності ви знаєте? Як вони застосовні до генеративного штучного інтелекту?

Лекція 10 – Генеративний штучний інтелект

Генеративні моделі використовують алгоритми машинного навчання, щоб вивчати розподіл із набору навчальних даних, а потім генерувати нові приклади з цього розподілу. Наприклад, генеративну модель можна навчити на зображеннях тварин, а потім використовувати для створення нових зображень тварин. Ми можемо думати про таку генеративну модель у термінах розподілу $p(x|w)$, у якому x є вектором у просторі даних, а w представляє параметри моделі, які можна вивчати. У багатьох випадках нас цікавлять умовні генеративні моделі форми $p(x|c,w)$, де c представляє вектор обумовлюючих змінних. У випадку генеративної моделі для зображень тварин можливо треба вказати, що згенероване зображення має бути певної тварини, наприклад кішки чи собаки, визначеної значенням c. Для реальних додатків, таких як генерація зображень, розподіли є надзвичайно складними, і, отже, впровадження глибокого навчання значно покращило якість генеративних моделей.

10.1. Генерація зображень. Змагальне навчання

Розглянемо генеративну модель, засновану на нелінійному перетворенні від латентного простору z до простору даних x. Введемо латентний розподіл $p(z)$, який може мати форму звичайної гаусіани

$$p(z) = \mathcal{N}(z|0, I), \tag{10.1}$$

разом із нелінійним перетворенням $x = g(z, w)$, визначеним глибокою нейронною мережею з параметрами w, які можна вивчати – так званий «генератор». Разом вони неявно визначають розподіл по x. Наша мета полягає в тому, щоб підібрати цей розподіл по набору даних навчальних прикладів $\{x_n\}$, де $n = 1, \dots, N$. Однак, параметри w неможливо визначити звичним нам шляхом оптимізації функції правдоподібності, оскільки ми ніяк не можемо оцінити останню. Ключова ідея генеративних змагальних мереж (Generative Adversarial Networks, GAN) полягає у введенні другої мережі дискримінатора, яка навчається спільно з мережею-генератором і яка забезпечує навчальний сигнал для оновлення параметрів генератора.

Мета мережі-дискримінатора полягає в тому, щоб розрізнити реальні приклади з набору даних і синтетичні, або «фальшиві», приклади, створені мережею-генератором. Мережа-дискримінатор навчається шляхом мінімізації звичайної функції помилки класифікації. І навпаки, мета мережі-генератора полягає в тому, щоб максимізувати вищезазначену помилку шляхом синтезу прикладів з того самого розподілу, що й навчальний набір. Таким чином, мережі генератора та дискримінатора працюють одна проти одної, звідси і термін «змагальна». Це приклад гри з нульовою сумою, у якій будь-який виграш однієї мережі означає програш для іншої. Це дозволяє мережі-дискримінатору надавати навчальний сигнал, який можна використовувати для навчання мережі-

генератора, і це перетворює проблему моделювання щільності розподілу без контрольованого навчання на форму контрольованого навчання.

На рис. 10.1 показано схематичну ілюстрацію GAN, у якій нейронну мережу-дискримінатор $d(x, \phi)$ навчено розрізняти реальні зразки з навчального набору, у цьому випадку зображення кошенят, і синтетичні зразки, створені генераторною мережею $g(z, w)$. Генератор прагне максимізувати помилку мережі дискримінатора, створюючи реалістичні зображення, тоді як мережа дискримінатора намагається мінімізувати ту саму помилку, стаючи кращим у розрізненні реальних прикладів від синтетичних.

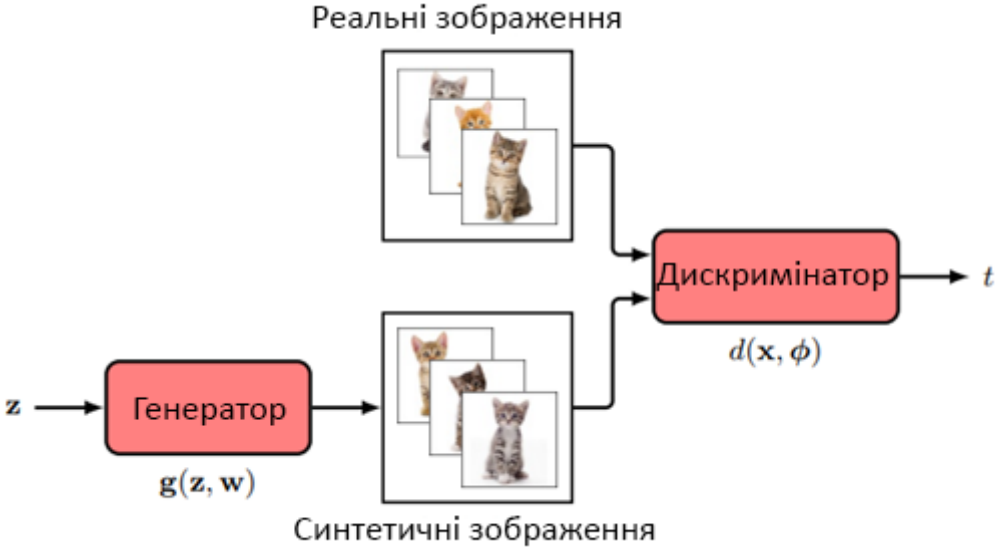


Рис. 10.1. Блок-схема генеративної змагальної мережі.

10.2. Функція втрат

Формально, визначимо бінарну цільову змінну, задану як

$$t = 1, \text{реальні дані}, \tag{10.2}$$

$$t = 0, \text{синтетичні дані}. \tag{10.3}$$

Мережа-дискримінатор має один вихідний блок із функцією активації сигмоїда, вихід якого представляє ймовірність того, що вектор даних x є дійсним:

$$P(t = 1) = d(x, \phi). \tag{10.4}$$

Будемо навчати мережу-дискримінатор за допомогою стандартної функції крос-ентропійної помилки:

$$E(w, \phi) = -\frac{1}{N}\sum_{n=1}^{N}(t_n \ln d_n + (1 - t_n)\ln(1 - d_n)). \tag{10.5}$$

де $d_n = d(x_n, \phi)$ є виходом мережі-дискримінатора для вхідного вектора n. Навчальний набір містить як реальні приклади даних, позначені x_n, так і синтетичні приклади, надані на виході мережі-генератора $g(z_n, w)$, де z_n є випадковою вибіркою з латентного розподілу простору $p(z)$. Оскільки $t_n = 1$ для

реальних прикладів і $t_n = 0$ для синтетичних прикладів, можна записати функцію помилки (10.5) у вигляді:

$$E_{GAN}(w, \phi) = -\frac{1}{N_{real}} \sum_{n \in real} \ln d(x_n, \phi) - \frac{1}{N_{synth}} \sum_{n \in synth} \ln(1 - d(g(z_n, w)), \phi)), \quad (10.6)$$

де зазвичай число N_{real} реальних точок даних дорівнює кількості N_{synth} точок синтетичних даних. Цю комбінацію мереж генератора та дискримінатора можна навчити наскрізно за допомогою стохастичного градієнтного спуску з градієнтами, оціненими за допомогою методу зворотного поширення помилки. Однак незвичайним аспектом є змагальне навчання, за допомогою якого помилка мінімізована щодо ϕ, але максимізована щодо w.

Цю максимізацію можна здійснити за допомогою стандартних градієнтних методів, замінивши знак градієнта на протилежний, так що оновлення параметрів стають:

$$\Delta\phi = -\lambda \nabla_{\phi} E_n(w, \phi), \quad (10.7)$$

$$\Delta w = \lambda \nabla_{w} E_n(w, \phi), \quad (10.8)$$

де $E_n(w, \phi)$ позначає помилку, визначену для точки даних n. Зауважимо, що два члени в (10.7) і (10.8) мають різні знаки, оскільки дискримінатор навчений зменшувати частоту помилок, тоді як генератор навчений її збільшувати. На практиці навчання чергується між оновленням параметрів генеративної мережі та оновленням параметрів дискримінаційної мережі, у кожному випадку роблячи лише один крок градієнтного спуску, після чого генерується новий набір синтетичних зразків. Якщо генератору вдасться знайти ідеальне рішення, тоді мережа-дискримінатор не зможе визначити різницю між реальними та синтетичними даними і, отже, завжди видаватиме результат 0,5. Після навчання GAN мережу-дискримінатор відкидають, а мережу-генератор можна використовувати для синтезу нових прикладів у просторі даних шляхом вибірки з латентного простору та поширення цих вибірок через навчену мережу-генератор. Можна показати, що для генеративних і дискримінаційних мереж, які мають необмежену гнучкість, повністю оптимізована GAN матиме генеративний розподіл, який точно відповідає розподілу даних.

На рис. 1.3. показано кілька синтетичних облич, згенерованих генеративною змагальною мережею.

Рис. 10.2. Синтезовані зображення облич мережею GAN.

Розглянута досі модель GAN генерує вибірки з безумовного розподілу $p(x)$. Наприклад, вона може генерувати синтетичні зображення собак, якщо її навчити на зображеннях собак. Також можливе створення умовних GAN, які беруть вибірку з умовного розподілу $p(x|c)$, у якому умовний вектор c може, наприклад, представляти різні види собак. Для цього і генератор, і мережа дискримінатора приймають c як додатковий вхід, а позначені приклади зображень, що містять пари $\{x_n, c_n\}$, використовуються для навчання. Після навчання GAN можна створити зображення з потрібного класу, встановивши c у відповідний вектор класу. Порівняно з навчанням окремих GAN для кожного класу, це має перевагу в тому, що спільні внутрішні представлення можна вивчати спільно в усіх класах, таким чином роблячи ефективніше використання даних.

10.3. Навчання GAN на практиці

Хоча GAN можуть давати високоякісні результати, їх непросто успішно навчити через змагальність навчання. Крім того, на відміну від стандартної функції мінімізації помилок, немає метрики прогресу, оскільки ціль може як зростати, так і знижуватися під час навчання.

Одна проблема, яка може виникнути, називається схлопування мод розподілу, у якому ваги мережі-генератора адаптуються під час навчання таким чином, що всі зразки латентної змінної z відображаються на підмножину можливих дійсних виходів. У крайніх випадках вихід може відповідати лише одному або невеликій кількості вихідних значень x. Потім дискримінатор присвоює значення $0{,}5$ цим примірникам, і навчання припиняється. Наприклад, GAN, навчений рукописним цифрам, може навчитися генерувати лише приклади

цифри «3», і хоча дискримінатор не в змозі відрізнити їх від справжніх прикладів цифри «3», він не розпізнає, що генератор не генерування повного діапазону цифр.

Зрозуміти труднощі навчання GAN можна, розглянувши рисунок 10.3, який показує простий одновимірний простір даних x із вибірками $\{x_n\}$, отриманими з фіксованого, але невідомого розподілу даних $p_{Data}(x)$. Також показано початковий генеративний розподіл $p_G(x)$ разом із зразками, взятими з цього розподілу. Оскільки дані та генеративний розподіл дуже різні, оптимальну дискримінаторну функцію $d(x)$ легко вивчити, вона має дуже різке падіння з практично нульовим градієнтом поблизу реальних або синтетичних зразків. Розглянемо другий член у функції помилки GAN (10.6). Оскільки $d(g(z,w),\phi)$ дорівнює нулю в області, охопленій згенерованими зразками, невеликі зміни в параметрах w генеративної мережі викликають дуже незначні зміни на виході дискримінатора, тому градієнти є малими і навчання йде повільно.

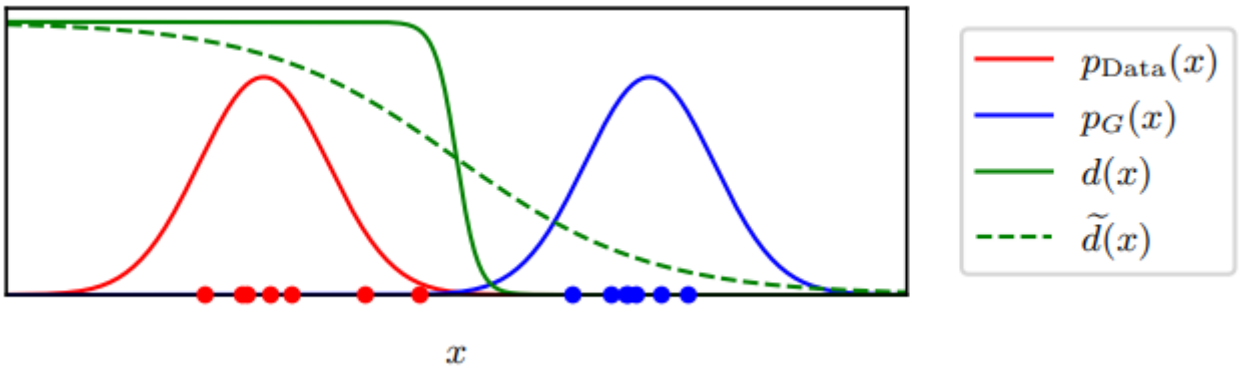


Рис. 10.3. Розподіли даних та генеративної змагальної мережі.

Зазначену проблему можна вирішити за допомогою згладженої версії $\tilde{d}(x)$ функції дискримінатора, проілюстрованої на малюнку 10.3, забезпечуючи тим самим сильніший градієнт для тренування генераторної мережі. Генеративна змагальна мережа із найменшими квадратами досягає згладжування шляхом модифікації дискримінатора для отримання виходу в дійсних значеннях, а не ймовірності в діапазоні $(0, 1)$, і шляхом заміни функції крос-ентропійної помилки на помилку суми квадратів. Альтернативно, техніка екземплярного шуму додає гаусівський шум як до реальних даних, так і до синтетичних зразків, що знову призводить до більш гладкої функції дискримінатора.

Численні інші модифікації функції помилок GAN і процедури навчання були запропоновані для покращення навчання. Одна зміна, що часто використовується для заміни члена генеративної мережі в оригінальній функції втрат:

$$-\frac{1}{N_{synth}}\sum_{n\in synth}\ln(1-d\big(g(z_n,w)\big),\phi)), \tag{10.9}$$

на модифіковану форму:

$$\frac{1}{N_{synth}} \sum_{n \in synth} \ln(d(g(z_n, w)), \phi)), \tag{10.10}$$

В той час як перша форма мінімізує ймовірність того, що зображення є підробкою, друга версія максимізує ймовірність того, що зображення справжнє. Різні властивості цих двох форм можна зрозуміти з рисунка 10.4. Коли генеративний розподіл $p_G(x)$ сильно відрізняється від справжнього розподілу даних $p_{Data}(x)$, величина $d(g(z,w))$ близька до нуля, і, отже, перша форма має дуже малий градієнт, тоді як друга форма має великий градієнт, що веде до швидшого навчання.

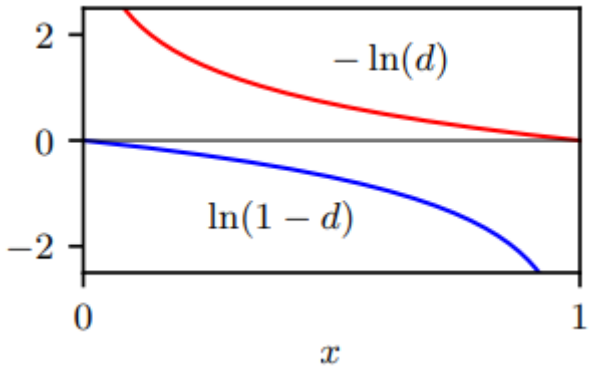


Рис. 10.4. Графіки $-\ln(d)$ та $\ln(1-d)$, що показують дуже різну поведінку градієнтів поблизу $d = 0$ та $d = 1$.

Більш прямий спосіб переконатися, що розподіл генератора $p_G(x)$ рухається до розподілу даних $p_{data}(x)$ – змінити критерій помилки, щоб відобразити, наскільки далеко один від одного знаходяться два розподіли в просторі даних. Це можна виміряти за допомогою відстані Вассерштейна. Для прикладу, уявіть розподіл $p_G(x)$ як купу землі, яку транспортують невеликими порціями для побудови розподілу $p_{data}(x)$. Метрика Вассерштейна – це загальна кількість переміщеної землі, помножена на середню пройдену відстань. З багатьох способів перегрупування купи землі для побудови $p_{data}(x)$ той, який дає найменшу середню відстань, використовується для визначення метрики.

10.4. Великі мовні моделі

Вище нами було розглянуто генеративні змагальні мережі, що можуть бути використані для генерації якісних зображень. Розглянемо тепер сучасні підходи щодо генерації тестів, а саме – великі мовні моделі (ВММ, Large Language Models, LLM).

В основі ВММ лежить глибоке навчання, яке включає нейронні мережі, що складаються з багатьох шарів вузлів (нейронів). Кожен нейрон приймає вхідні дані, обробляє їх і передає на наступний шар. Зв'язки між нейронами мають пов'язані ваги, які коригуються під час навчання, щоб мінімізувати помилку в

прогнозах моделі. Мета полягає в тому, щоб розпізнавати складні шаблони в даних, дозволяючи моделі розуміти та створювати людську мову.

В основі таких мереж лежить архітектура трансформера, що покращує та усуває обмеження попередніх мовних моделей, розглянутих раніше в нашому курсі, таких як рекурентні нейронні мережі (RNN) і мережі довгострокової короткочасної пам'яті (LSTM). Трансформери розроблені для більш ефективної обробки послідовних даних за допомогою механізму уваги.

Механізм уваги дозволяє моделі зосереджуватися на різних частинах вхідної послідовності під час прогнозування. Наприклад, у реченні «The cat sat on the mat» під час обробки слова «sat» модель може звернути увагу на слово «cat», щоб зрозуміти зв'язок між двома словами. Це досягається шляхом призначення різних оцінок уваги кожному слову у вхідній послідовності, що відображає їх релевантність поточному слову, яке обробляється. На рис. 10.5. показано візуалізацію механізму уваги для машинного перекладу. Зверніть увагу, що порядок слів речення в українській та англійській мовах різний, увага дозволяє мережі концентруватись на релевантних словах до поточного слова, що виводиться.

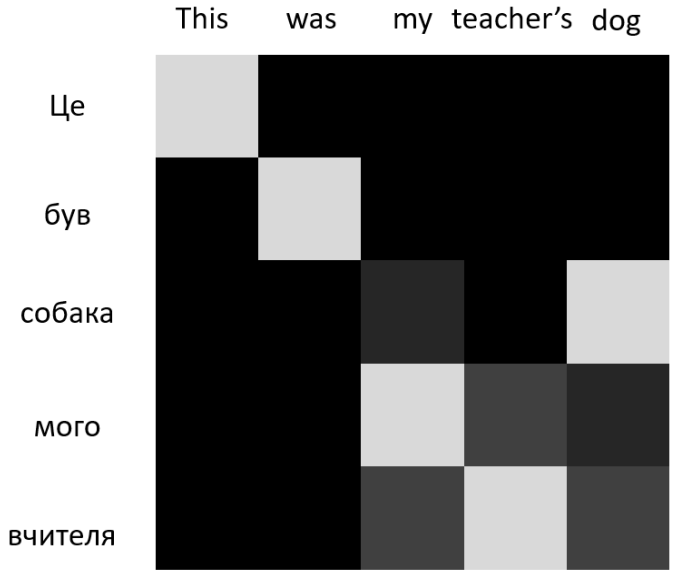


Рис. 10.5. Візуалізація механізму уваги при перекладі.

Механізм уваги може бути застосований не лише при перекладі або в ВММ. Можна також підвищити якість анотація зображень за допомогою уваги. На рис. 10.6. показано як при виводі різних слів, що описують зображення нейронна мережа генерує різні маски уваги (підсвічено світлим). Так, на слові «птиця» максимальний фокус саме на ній, а коли мова заходить про воду, птиця вже не в фокусі механізму уваги.

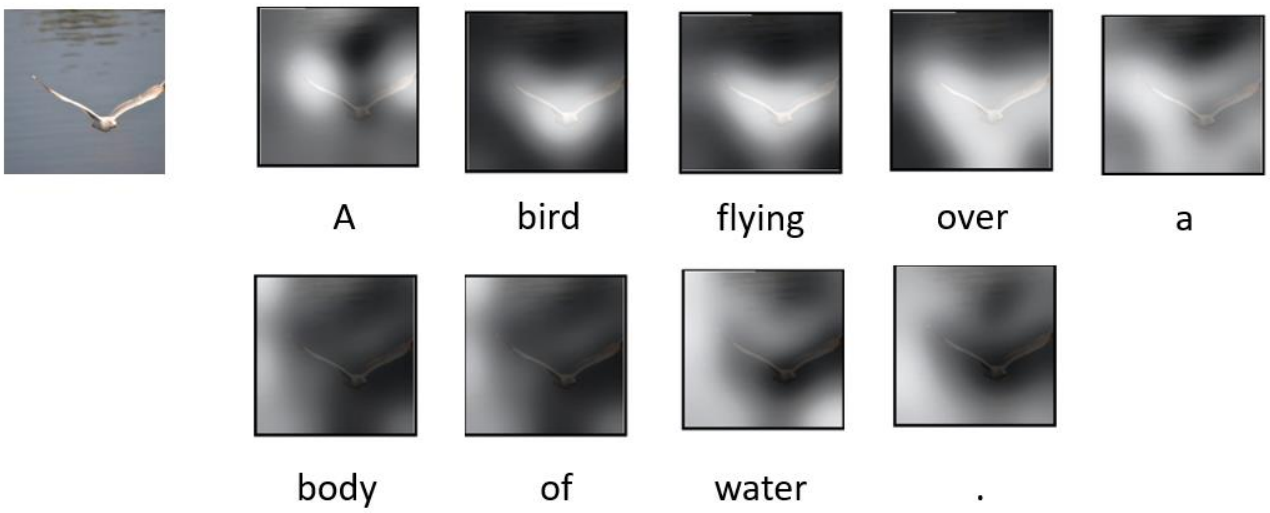


Рис. 10.6. Поступовий зсув уваги нейронної мережі під час анотації зображення.

Трансформери використовують кілька так званих «голів уваги», що працюють паралельно, відомі як багатоголова увага. Кожна голова вчиться зосереджуватися на різних аспектах речення. Одна голова може зосереджуватися на синтаксичній структурі, а інша – на семантичному значенні. Вихідні дані з цих голів об’єднуються та проходять через подальші рівні обробки, що дозволяє моделі отримувати більш глибоке розуміння вхідних даних.

Навчання моделі. Навчання великої мовної моделі включає кілька ключових кроків:

1. Збір даних: модель тренується на різноманітних і великих текстових даних, включаючи книги, статті, веб-сайти тощо. Ці дані є основою для моделі, що дозволяють їй вивчити структуру мови, лексику та контекст.
2. Попередня обробка: текстові дані токенізуються, тобто розбиваються на менші одиниці, такі як слова або підслова. Це полегшує обробку даних моделлю.
3. Некероване навчання: модель навчається передбачати наступне слово в реченні за попередніми словами. Це робиться за допомогою методу, що називається оцінкою максимальної правдоподібності, коли модель коригує свої параметри, щоб максимізувати ймовірність спостережуваних даних. Під час навчання модель обробляє мільйони або мільярди речень, навчаючись розпізнавати закономірності та зв’язки в даних.
4. Тонке налаштування: після етапу початкового навчання модель можна точно налаштувати на конкретних наборах даних, щоб покращити її продуктивність у конкретних задачах. Наприклад, ChatGPT може бути налаштований на дані діалогу, щоб покращити його можливості спілкування.

Генерація тексту. Коли користувач вводить текст у велику мовну модель, модель використовує отримані знання, щоб створити відповідь.

Алгоритм 10.1. Генерація тексту великою мовною моделлю.

1. Кодування вхідних даних: вхідний текст токенізується та перетворюється на числове представлення (вкладання слів), яке може обробити модель.

2. Контекстуальне розуміння: модель обробляє вхідні дані через свої шари, використовуючи механізми уваги для розуміння контексту. Він розглядає всю вхідну послідовність і призначає оцінку уваги, щоб зафіксувати зв'язки між різними словами.

3. Авторегресійна генерація: модель генерує відповідь по одному токену за раз. На кожному кроці вона передбачає найвірогідніший наступний токен на основі вхідних даних і токенів, які вона вже згенерувала. Цей процес триває, поки модель не згенерує повну відповідь.

4. Декодування: згенеровані токени перетворюються назад у зрозумілий людині текст, утворюючи остаточну відповідь, яка надається користувачеві.

Для забезпечення відповідності створеного вмісту вказівкам і уникнути створення шкідливого чи невідповідного тексту, використовується кілька методів:

- тонке налаштування: модель можна точно налаштувати на наборах даних, які змушують модель видавати безпечні та змістовні відповіді, зменшуючи ймовірність створення шкідливого вмісту;
- розробка підказок: на вхід моделі можуть подаватися текстові підказки, що спрямували б модель на генерування бажаних результатів;
- фільтрування: до створеного тексту можна застосувати додаткові механізми фільтрації для виявлення та видалення неприйнятного чи небезпечного вмісту до того, як він буде представлений користувачеві.

10.5. Огляд існуючих великих мовних моделей

Великі мовні моделі суттєво просунулися за останні кілька років, з кількома ключовими моделями, які лідирують у цій галузі. Ці моделі розроблені різними організаціями та мають різні можливості, архітектуру та джерела даних для навчання.

Однією із перших великих мовних моделей, що привернула увагу широкого кола людей, є GPT-3, розроблена OpenAI. GPT-3, що означає Generative Pre-trained Transformer 3, використовує архітектуру трансформера для обробки та генерації тексту, подібного до того, як спілкуються люди. Вона має 175 мільярдів параметрів, що зробило її на той час однією з найбільших і найпотужніших мовних моделей. GPT-3 здатний виконувати широкий спектр задач обробки природньої мови, включаючи продовження тексту, переклад, резюмування та навіть генерацію коду.

Ще одна відома модель – BERT (Bidirectional Encoder Representations from Transformers), розроблена Google. BERT відрізняється від GPT-3 тим, що вона розроблена для розуміння контексту слів у всіх напрямках, що робить її особливо

ефективною для таких задач, як відповіді на запитання та аналіз настроїв. Двонаправлений підхід BERT дозволяє фіксувати значення слова на основі навколишніх слів, що забезпечує точніше розуміння мови.

Google також розробив T5 (Text-to-Text Transfer Transformer), який розглядає кожну задачу обробки природньої мови як задачу відображення тексту в текст. Цей уніфікований підхід дозволяє T5 виконувати різноманітні задачі, такі як переклад, узагальнення та відповіді на запитання, використовуючи ту саму архітектуру моделі. T5 продемонстрував вражаючу якість у численних тестах завдяки своїй універсальності та масштабу навчальних даних.

RoBERTa від Facebook базується на BERT, оптимізуючи процес попереднього навчання. RoBERTa використовує більше даних і обчислювальних ресурсів для покращення якості BERT, досягаючи найсучасніших результатів за кількома тестами обробки природньої мови. Удосконалення RoBERTa включають навчання з довшими послідовностями та динамічну зміну шаблону маскування під час навчання.

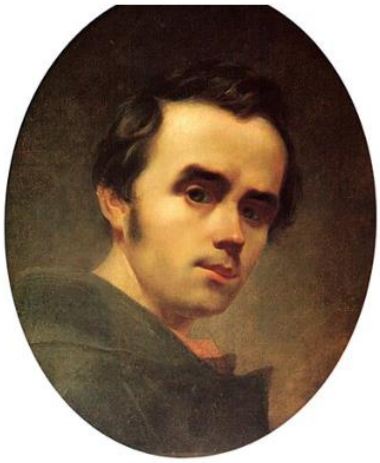
GPT-4 від OpenAI, наступник GPT-3, розширює можливості ВММ за допомогою ще більшої кількості параметрів і вдосконалених методів навчання. GPT-4 може вирішувати складніші задачі, генерувати більш зв'язних і релевантний текст та має покращену якість у таких сферах, як логічне мислення та фактологічна точність.

Крім того, існує XLNet, інша модель, розроблена Google, яка поєднує в собі переваги BERT і авторегресійних моделей, таких як GPT. XLNet використовує навчальну мету на основі перестановок, яка дозволяє захоплювати двонаправлений контекст, зберігаючи властивості авторегресії, які допомагають у генеративних завданнях. Такий підхід сприяє кращому виконанню завдань, що вимагають розуміння та генерації зв'язного тексту.

Інша модель – Gemini, розроблена Google DeepMind, – це сімейство великих мовних моделей, призначених для виконання широкого спектру завдань, таких як генерація тексту, розуміння та резюмування. Gemini розроблено для обробки не лише тексту, але й зображень та інших типів даних, що забезпечує більш багату взаємодію та відповіді. Її архітектура також базується на трансформерах.

Однією з широко використовуваних великих мовних моделей із безкоштовним доступом є ChatGPT від OpenAI. На даний момент можливе використання ChatGPT 4, що дозволяє працювати не лише з текстом, а й з зображеннями.

Для роботи з GPT-4, необхідно перейти за посиланням: https://chatgpt.com/ Інтерфейс побудовано у вигляді чату, де користувач як і в будь-якому месенджері типе текстове повідомлення. При цьому можливо комбінувати текст і зображення так, наприклад, подавши автопортрет Тараса Григоровича Шевченко та спитавши: «Чим відома ця картина?» можна отримати ґрунтовну відповідь про неї, як показано на рис. 10.7. Зауважимо, що запит не містив імені письменника, модель сама розпізнала картину.



Чим відома ця картина?

Ця картина — автопортрет Тараса Шевченка, намальований олією в 1840 році. Тарас Шевченко відомий як видатний український поет, письменник, художник та національний герой. Він зробив значний внесок у розвиток української літератури та культури. Його творчість мала великий вплив на формування української національної свідомості.

Цей автопортрет є одним з найвідоміших зображень Шевченка, який відображає його талант і як художника, і як поета. Картина є важливою частиною українського мистецтва і часто

Рис. 10.7. Комбінований запит (зображення + текст) до ChatGPT.

Авжеж, ChatGPT може допомогти і з суто текстовими запитаннями, що стосуються навчання, історії. ChatGPT може стати цінним активом для бізнесу, оскільки покращує підтримку клієнтів, автоматизує відповіді на поширені запити та забезпечує цілодобову підтримку. Вона також може оптимізувати створення контенту, допомагаючи командам швидко створювати цікаві публікації в блогах, маркетингові матеріали та вміст соціальних мереж. Крім того, ChatGPT допомагає в аналізі даних, узагальнюючи відгуки клієнтів і витягуючи інформацію для прийняття рішень. Спрощуючи навчання та адаптацію, це допомагає новим співробітникам адаптуватися ефективніше. Підприємства можуть використовувати ChatGPT для персоналізованих маркетингових кампаній, мозкового штурму інноваційних ідей і автоматизації рутинних завдань, зрештою підвищуючи продуктивність і знижуючи операційні витрати. Її здатність перекладати мови додатково дозволяє компаніям охопити ширшу аудиторію, що робить його універсальним інструментом для різноманітних бізнес-потреб.

10.6. Контрольні питання

1. Як нейронні мережі вирішують задачу генерації зображень?
2. Оцініть якість генерації облич на рис. 10.2. Чи є вони достатньо реалістичними?
3. Яку функцію виконує дискримінатор в генеративних змагальних мережах?
4. Які переваги та недоліки мають генеративні змагальні моделі?
5. Що таке «велика мовна модель»? Які великі мовні моделі ви знаєте?
6. Як механізм уваги дозволяє вирішувати задачі, пов’язані із текстом?
7. З яких кроків складається генерація тексту великою мовною моделлю?
8. Які моделі генеративного штучного інтелекту ви використовували? Опишіть свій досвід.

Рекомендовані джерела інформації

Базові:

1. Субботін С.О. Нейронні мережі: теорія і практика. Навчальний посібник. Житомир, 2020. 184с. URL: http://eir.zntu.edu.ua/bitstream/123456789/6800/1/Subbotin_Neural.pdf
2. І.А. Терейковський, Д.А. Бушуєв, Л.О. Терейковська Штучні нейронні мережі: базові положення. Навчальний посібник. Київ, 2022. 123с. URL: https://ela.kpi.ua/server/api/core/bitstreams/9fee52b6-83fc-4e99-8541-c2767f634c7c/content
3. Bishop C.M., Bishop H. Deep learning – foundations and concepts. Springer, 2024. 649p. DOI: 10.1007/978-3-031-45468-4.
4. TensorFlow documentation. [Online] URL : https://www.tensorflow.org/learn
5. I. Goodfellow, Y. Bengio, A. Courville The Deep Learning Book. MIT Press, 2016. URL: https://www.deeplearningbook.org/
6. Желдак Т.А. Самонавчання складних систем [Електронний ресурс] : методичні рекомендації до виконання практичних робіт для здобувачів ступеня магістра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / Т.А. Желдак, К.С. Хабарлак, Д.М. Гаранжа ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2024. – 66 с.

Додаткові:

1. Khabarlak K., Koriashkina L. Fast Facial Landmark Detection and Applications: A Survey. Journal of Computer Science and Technology. – 2022. – Vol. 22. – № 1. – P. 02. – DOI: 10.24215/16666038.22.E02.
2. Khabarlak K. Post-Train Adaptive MobileNet for Fast Anti-Spoofing // Proceedings of the 3rd International Workshop on Intelligent Information Technologies & Systems of Information Security, Khmelnytskyi, Ukraine, March 23–25. T. 3156. CEUR-WS.org, 2022. C. 44–53. (CEUR Workshop Proceedings). URL: http://ceur-ws.org/Vol-3156/keynote5.pdf
3. Khabarlak K. Post-Train Adaptive U-Net for Image Segmentation. Information Technology: Computer Science, Software Engineering and Cyber Security. 2022. № 2. P. 73—78. DOI: 10.32782/IT/2022-2-8.

Навчальне видання

Хабарлак Костянтин Сергійович
Желдак Тимур Анатолійович

САМОНАВЧАННЯ СКЛАДНИХ СИСТЕМ

Конспект лекцій
для здобувачів ступеня магістра
освітньо-професійної програми «Системний аналіз»
зі спеціальності 124 Системний аналіз

Видано в авторській редакції.

Електронний ресурс.
Підписано до видання 16.07.2024. Авт. арк. 8,1.

Національний технічний університет «Дніпровська політехніка».
49005, м. Дніпро, просп. Дмитра Яворницького, 19.