Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра моделювання складних систем

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра
за спеціальністю 113 «Прикладна математика»
на тему:

АНАЛІЗ МЕТОДІВ НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ

студента 4 курсу
Калюжного Дениса Володимировича

_____

Науковий керівник:
доцент, кандидат технічних наук
Кулян В. Р.

_____

Робота заслухана на засіданні кафедри моделювання складних систем та рекомендована до захисту, протокол № 10 від 01 червня 2021 р.

Завідувач кафедри МСС канд. фіз.-мат. наук, доц. Черній Д.І.

Київ – 2021

ЗМІСТ

ВСТУП

На сьогоднішній день нейронні мережі вносять свою лепту у розвиток людства. Якщо десятиліття тому штучний інтелект розглядався як щось міфічне та далеке від повсякденного життя, то сьогодні методи даної галузі застосовуються у сфері бізнесу, безпеки, розваг, комп'ютерних ігор та навіть управління державними установами. Станом на зараз нейронні мережі дозволяють розв'язувати широкий спектр задач. До них входять побудови моделей регресії та класифікації цільової метрики на основі табличних даних, побудова моделей класифікації та сегментації зображень, побудова прогнозуючих моделей на основі послідовних даних (часові ряди), побудова генеративних моделей тощо.

Актуальність даної роботи підтверджується стрімким розвитком галузі штучних нейронних мереж та популярністю їх застосування. Підставою виконання даної роботи є необхідність зіставити теоретичні відомості із результатами практичного застосування, вивести відповідні рекомендації щодо вибору відповідних методів при оптимізації навчання нейронних мереж.

Мета даної роботи – дослідити анатомію нейронних мереж, тобто висвітлити їх структуру, методи навчання та оптимізацію навчання, а також шляхом проведення обчислювального експерименту порівняти розглянуті методи та дати відповідні рекомендації щодо їх застосування.

Об'єктом дослідження є штучна нейронна мережа, а метод її дослідження – аналіз та систематизація наукової літератури, узагальнення викладених результатів. У роботі проводиться обчислювальний експеримент, у якому використовується мова програмування Python та додаткові бібліотеки.

Отримані висновки та запропоновані рекомендації даної роботи можуть знайти своє застосування у широкому спектрі задач оптимізації навчання

нейронної мережі. Рекомендації можуть бути корисними при виборі методу оптимізації навчання нейронної мережі, а саме у врахуванні балансу між швидкістю навчання та точністю апроксимації.

В першому розділі даної роботи введені загальні відомості про оптимізацію на основі методів градієнтного спуску. В другому розділі постановка задачі оптимізації звузиться до стохастичного підходу та його аспекти розглянуті більш детально. В третьому розділі введено поняття нейронної мережі та розглянуто особливості її навчання, оптимізації, алгоритм. В четвертому розділі висвітлено додаткові методи оптимізації навчання нейронної мережі, які мають перевагу у швидкості, та проведено обчислювальний експеримент з порівнянням результатів застосування цих методів.

РОЗДІЛ 1. ЗАСТОСУВАННЯ МЕТОДУ ГРАДІЄНТНОГО СПУСКУ ДЛЯ ОПТИМІЗАЦІЇ ФУНКЦІЇ

1.1 Постановка задачі оптимізації функції

У процесі постановки задачі оптимізації ставиться питання визначення найкращих параметрів об'єктів. В математичному розумінні даний об’єкт – це функція. Різновид оптимізації, що полягає в пошуку оптимальних значень параметрів при заданій структурі об'єкта, називається параметричним. Вибір оптимальної структури об’єкта є структурною оптимізацією.

Формулювання стандартної математичної задачі оптимізації виглядає наступним чином: серед елементів x допустимої множини $\mathbb{X}$ знайти такий x^* за якого значення функції F має мінімальне значення, тобто $F(x^*) \leq F(x)\ \forall x \in \mathbb{X}$. Інколи задача оптимізації передбачає максимізацію функції F. В області нейронних мереж частіше розглядають задачу мінімізації, тому якщо природно задача передбачає максимізацію функції, то при побудові математичної моделі розглядають мінімізацію тієї ж функції з від’ємним знаком. Такий підхід усуває необхідність реалізовувати декілька алгоритмів, що однакові з точністю до декількох перетворень.

Для постановки задачі оптимізації необхідно задати:

− допустиму множину $\mathbb{X} = \{x |\ g_i(x) \leq 0, i = 1, \dots, m\} \subset \mathbb{R}^n$

(g_i задають обмеження на $\mathbb{X}$);

− цільову функцію $F: \mathbb{X} \to \mathbb{R}$.

Розв’язати задачу $F(x) \to \min\limits_{x \in \mathbb{X}}$ означає показати одне з:

− $\exists\ x^* \in \mathbb{X}: F(x^*) = \min\limits_{x \in \mathbb{X}}(F(x))$;

− якщо попереднє не виконується, то знайти $\inf\limits_{x \in \mathbb{X}}(F(x))$;

– інакше $\nexists \inf_{x\in\mathbb{X}}\big(F(x)\big)$ або $\mathbb{X}=\emptyset$.

Зрозуміло, що вигляд розв'язку довільної задачі оптимізації є одним із вищеперерахованих. Якщо оптимального розв'язку не існує, то або обмеження $g_i(x)$ визначають $\mathbb{X}=\emptyset$, або цільова функція не обмежена знизу. Якщо оптимальний розв'язок існує, то або це мінімум функції, або інфінум у разі відсутності попереднього. Функція, що обмежена знизу, завжди має інфінум.

Функцію F в різних галузях називають по різному. Частіше називають цільовою функцією (англ. objective function), функцією втрат (англ. loss function) чи функцією витрат (англ. cost function). Якщо F не є опуклою, то часто обмежуються пошуком локальних мінімумів: точок x^* таких, що всюди в деякому їхньому околі $F(x^*)\leq F(x)$.

Якщо обмеження $g_i(x)$ відсутні, тобто визначають $\mathbb{X}=\mathbb{R}^n$, то дану задачу класифікують як задачу безумовної оптимізації, а інакше — умовної оптимізації. Надалі будемо говорити про безумовну параметричну задачу оптимізації.

1.2 Градієнтний спуск

Градієнтом функції F, яка залежить від n змінних $x_1, x_2, \dots, x_n$, за визначенням є вектор

$$\operatorname{grad} F = \nabla F = \left(\frac{dF}{dx_1}, \frac{dF}{dx_2}, \dots, \frac{dF}{dx_n}\right). \qquad (1.2.1)$$

Відповідно до геометричної інтерпретації градієнта функції, він є векторним полем, яке у кожній своїй точці вказує на напрямок найшвидшого зростання функції в цій же точці. Саме (1.2.1) створює фундамент для всіх градієнтних методів. Приклад функції та її градієнту на множині точок

зображено на рисунку 1.2.1. Ліворуч зображено графік деякої функцій, а праворуч її градієнт. Вектори вказують на напрямок найшвидшого зростання функції в точці, при чому їх довжина відображає інтенсивність її зростання (кут нахилу) в цій же точці.

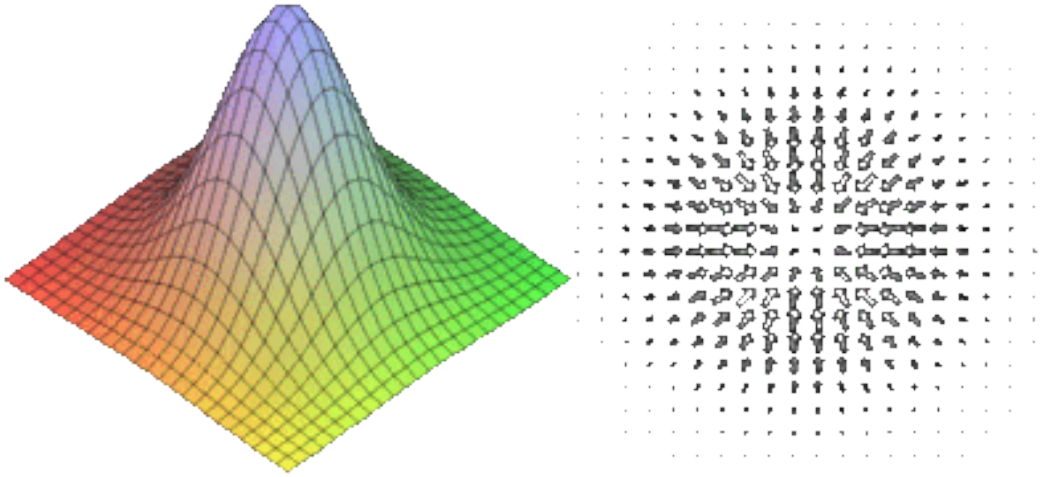

Рисунок 1.2.1 – Ілюстрація градієнту [1]

Градієнтні методи, серед яких найрозповсюдженішими є градієнтні спуски (англ. gradient descent), ґрунтуються на тому факті, що якщо функція декількох змінних $F(x)$ є визначеною та диференційовною в деякому околі точки x_0, то $F(x)$ спадає найшвидше в напрямку, протилежному $\nabla F(x_0)$, тобто в напрямку $-\nabla F(x_0)$.

На основі цього твердження покладемо $x_1 = x_0 - h\nabla F(x_0)$, тоді $\exists\, h > 0 \colon F(x_1) < F(x_0)$. Член $h\nabla F(x_0)$ віднімається від x_0, оскільки треба рухатися проти градієнту, тобто вниз до мінімуму. Починаючи з припущення, що x_0 є точкою локального мінімуму функції F, розглядають таку послідовність $x_1, x_2, \dots$, що

$$x_{n+1} = x_n - h\nabla F(x_n), n \in \mathbb{N}. \qquad (1.2.2)$$

Звідси, маємо співвідношення $F(x_0) > F(x_1) > F(x_2) > \cdots$. Якщо функція F опукла, то $F(x_n) \to \min\limits_{x \in \mathbb{X}}(F(x))$, а інакше послідовність збігається до одного з

локальних мінімумів. Саме (1.2.2) і є найпростішим різновидом методу градієнтного спуску.

Дійсне число h називається розміром кроку (англ. step size) спуску або інтенсивністю спуску. Його оптимальний вибір є дуже важливим. Якщо h дуже мале, то спуск буде проходити вкрай повільно. Якщо h занадто велике, то ризикуємо «перестрибнути» та опинитися в ситуації $F(x_n) \geq F(x_{n-1})$. Тому існує багато методів визначення оптимального кроку. Цей параметр настільки важливий, що через нього виділяють різні види градієнтних спусків. Наприклад, в класичному спуску значення h є сталим. В іншій модифікації значення h зменшується пропорційно наближенню до мінімуму. Метод найшвидшого спуску передбачає

$$h_n = \underset{h}{\operatorname{argmin}} F(x_{n+1}) = \underset{h}{\operatorname{argmin}} F\big(x_n - h\nabla F(x_n)\big), n \in \mathbb{N}. \qquad (1.2.3)$$

Загалом, враховуючи, що h не обов'язково є сталим, рекурентне співвідношення (1.2.2) узагальнюється на

$$x_{n+1} = x_n - h_n \nabla F(x_n), n \in \mathbb{N}. \qquad (1.2.4)$$

Наведемо деякі загальні міркування з приводу збіжності (1.2.2). Формальні умови збіжності будуть наведені для стохастичного градієнтного спуску в другому розділі.

Отже, наступні чинники впливають на збіжність градієнтного спуску. На кожному кроці послідовність змінюється в напрямку антиградієнта функції, помноженого на крок h. Як вже було сказано, h має бути достатньо мале для того, щоб крок було виконано вдало, але не настільки, щоб спуск проходив вкрай повільно. Вибір h суттєво впливає не тільки на швидкість збіжності, а на її присутність взагалі. На рисунку 1.2.2 продемонстровано розбіжність методу через великий крок.

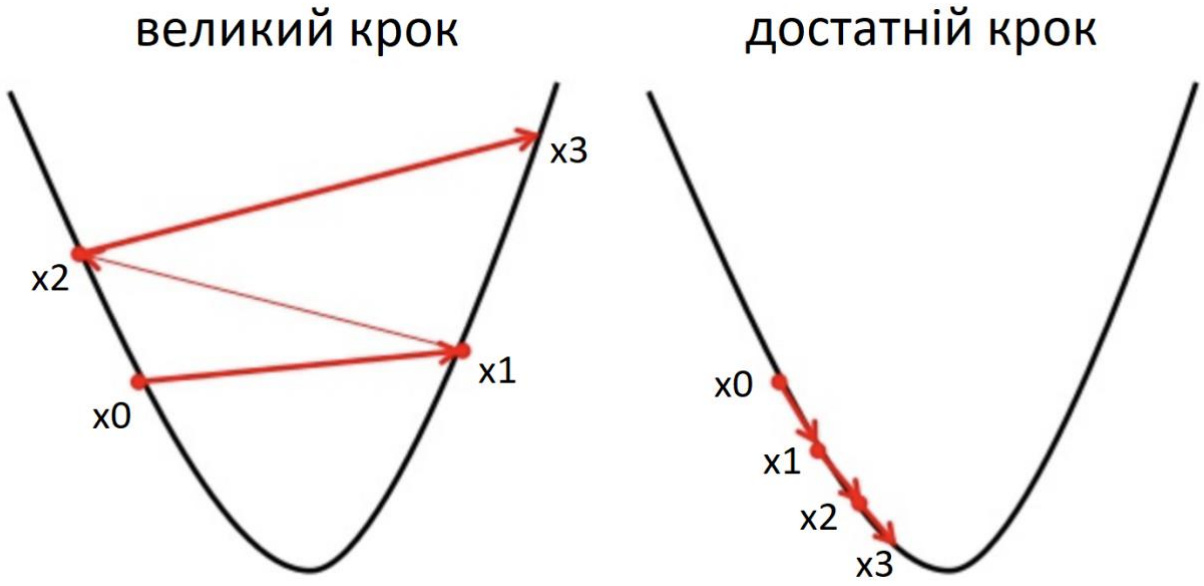

Рисунок 1.2.2 – Розбіжність та збіжність градієнтного спуску в залежності від розміру кроку [2]

Окрім вибору кроку, очевидно, на збіжність також впливає сама цільова функція. В довільній задачі оптимізації дуже важливу роль грає опуклість цільової функції. Опуклість фактично означає, що функція має єдиний глобальний екстремум. Тоді, якщо метод збігається, то однозначно до єдиного найкращого оптимуму.

Процес збіжності у випадку опуклої функції проілюстровано на рисунку 1.2.3. Тут маємо визначену на площині функцію F, графік якої має форму чаші. Сині криві показують області, де значення функції F стале, тобто криві є ізолініями. Червоні стрілки, що виходять з точок x_i, вказують на напрям, протилежний градієнтові в цих же точках. Також відзначимо той факт, що ці стрілки (градієнти) є ортогональними до ізолінії. Даний рисунок наочно ілюструє як спуск градієнтом методом веде послідовність x_i до дна чаші, тобто до точки, де функція F набуває мінімального значення.

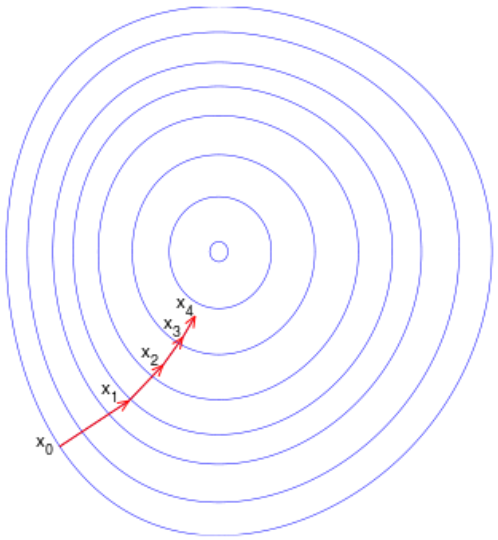

Рисунок 1.2.3 – Послідовність кроків градієнтного спуску [3]

Однак, у випадку, якщо функція F не є опуклою, то немає стовідсоткової гарантії збіжності методу саме до глобального мінімуму. Приклад «проблемної» функції проілюстровано на рисунку 1.2.4. Збіжність до глобального мінімуму у такому випадку залежить від вибору точки x_0 та кроку h.

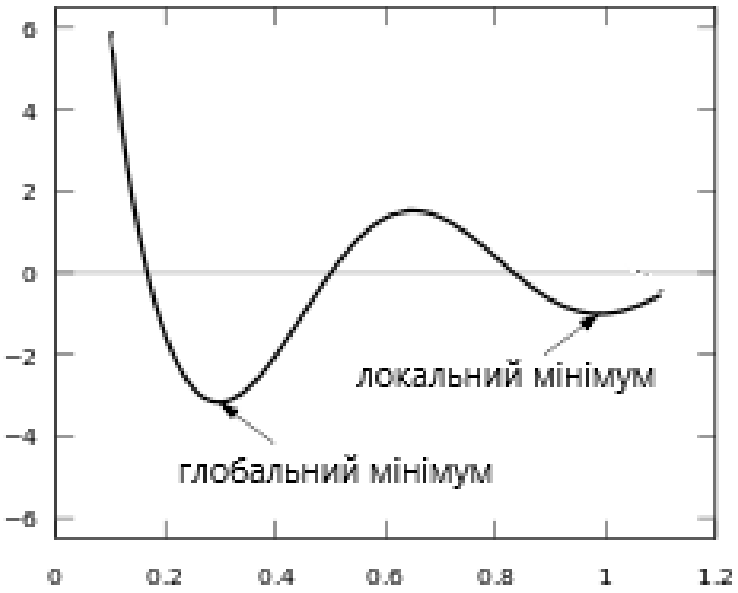

Рисунок 1.2.4 – Функція з декількома локальними мінімумами [4]

Ускладнювати ситуацію може погана обумовленість цільової функції в околі глобальних та локальних мінімумів. Із локальних може бути важко «вистрибнути», а до глобальних важко «застрибнути».

Різні варіації градієнтного спуску більш або менш схильні до потрапляння у «пастку» локального мінімуму. Наприклад, метод стохастичного градієнту має більшу схильність збігатися саме до глобального оптимуму, ніж класичний спуск (1.2.2). Через таку невизначеність гарантії збіжності до глобальних екстремумів, часто постановка задачі градієнтного спуску формулюється в термінах пошуку локального мінімуму.

1.3 Алгоритм градієнтного спуску та приклад застосування

Далі наведено алгоритм класичного градієнтного спуску за (1.2.2):

а) задаємо початкове наближення x_0 та крок h;

б) обираємо точність ε та одну з наступних умов зупинки:

1) $|x_{n+1} - x_n| < \varepsilon$;

2) $|F(x_{n+1}) - F(x_n)| < \varepsilon$;

3) $\|\nabla F(x_{n+1})\| < \varepsilon$.

в) індекс поточного кроку $n = 0$;

г) знаходимо $x_{n+1} = x_n - h\nabla F(x_n)$;

ґ) перевіряємо умови зупинки:

1) якщо обрана з пункту (б) умова не виконується, то $n = n + 1$ та повертаємося до пункту (г);

2) інакше $x^* = x_{n+1}$ та зупиняємо алгоритм.

Хоча метод градієнтного спуску на практиці зазвичай не застосовується для розв’язку систем рівнянь, в даному випадку це гарна можливість

продемонструвати його роботу на простому прикладі. Тому розглянемо приклад розв'язку системи нелінійних рівнянь за алгоритмом для (1.2.2).

Нехай маємо систему, для якої потрібно знайти наближений розв'язок:

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - \dfrac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ e^{-x_1 x_2} + 20x_3 + \dfrac{10\pi - 3}{3} = 0 \end{cases}$$

Відповідно до неї сформуємо вектор-функцію:

$$\text{G(x)} = \begin{pmatrix} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ e^{-x_1 x_2} + 20x_3 + \frac{10\pi - 3}{3} \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Функцією втрат буде $F(x) = \frac{1}{2} G^T(x) G(x),\ h = 0.001, \varepsilon = 10^{-4}$.

Дана функція є сумою квадратів лівих частин рівнянь системи, помножена на $\frac{1}{2}$. Такий вибір пояснюється тим, що цільова функція мінімізується, а відповідно мінімізується і кожен її доданок, який є окремим рівнянням системи, але піднесений до квадрату, щоб уникнути від'ємних значень. Тому рівність $F(x) = 0$ еквівалента рівності всіх рівнянь системи нулю.

Множник $\frac{1}{2}$ не є обов'язковим, але його зазвичай додають для зручності роботи з градієнтом функції F.

Заходимо $\nabla F(x) = J_G(x)^T G(x)$, де $J_G(x)^T -$ матриця Якобі функції G.

$$J_G = \begin{pmatrix} 3 & \sin(x_2 x_3) x_3 & \sin(x_2 x_3) x_2 \\ 8x_1 & -1250x_2 + 2 & 0 \\ -2x_2 e^{-x_1 x_2} & -x_1 e^{-x_1 x_2} & 20 \end{pmatrix}.$$

Задаємо вектор початкового наближення $x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$.

Звідси $\nabla F(x_0) = J_G(x_0)^T G(x_0) = \begin{pmatrix} 3 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 20 \end{pmatrix} \begin{pmatrix} -2.5 \\ -1 \\ 10.472 \end{pmatrix} = \begin{pmatrix} -7.5 \\ -2 \\ 209.44 \end{pmatrix}$.

Тоді $x_1 = x_0 - h\nabla F(x_0) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - 0.001 \begin{pmatrix} -7.5 \\ -2 \\ 209.44 \end{pmatrix} = \begin{pmatrix} 0.0075 \\ 0.002 \\ -0.20944 \end{pmatrix}$.

Також відзначаємо, що $F(x_0) = 0.5\big((-2.5)^2 + (-1)^2 + (10.472)\big) = 58.456$.

Тепер впевнимося, що виконується $F(x_1) < F(x_0)$.

$F(x_1) = 0.5\big((-2.48)^2 + (-1)^2 + (6.28)\big) = 23.306 \leq 58.456 = F(x_0)$.

Крок виконано вдало. Продовжуємо ітераційний процес, доки не буде виконуватись умова зупинки для заданого ε.

РОЗДІЛ 2. ЗАСТОСУВАННЯ МЕТОДУ СТОХАСТИЧНОГО ГРАДІЄНТНОГО СПУСКУ ДЛЯ ОПТИМІЗАЦІЇ ФУНКЦІЇ

2.1 Постановка задачі оптимізації випадкової функції

Раніше розглянуто оптимізацію методом класичного градієнтного спуску, що працює за принципом $x_{n+1} = x_n - h\nabla C(x_n)$, де $C -$ цільова функція, $h -$ крок. Проте, існують задачі, для яких пошук градієнту є дуже витратною обчислювальною процедурою. Тому її виконання на кожному кроці спуску може призвести до дуже тривалого процесу оптимізації. Тоді виникає потреба оцінити функцію C.

Нехай будемо виконувати спуск за принципом [5]

$$x_{n+1} = x_n - hQ(x_n). \tag{2.1.1}$$

Тут $Q(x_n) -$ оцінка $\nabla C(x_n)$, тобто $E[Q(x_n)] = \nabla C(x_n)$. Такий підхід називається методом стохастичного градієнтного спуску. Якщо існує така випадкова функція Q, яка обчислюється набагато простіше за ∇C, та при цьому математичне сподівання Q дорівнює ∇C, то має сенс використовувати саме Q для виконання оптимізації. На практиці так зазвичай і роблять.

Для того, щоб явно продемонструвати алгоритм стохастичного градієнтного спуску, конкретизуємо постановку задачі оптимізації, характерну для застосування даного алгоритму.

Розглянемо функцію втрат $L(z, w) = L_z(w)$, яка оцінює збитки деякої системи, визначеною параметром w, під час обробки спостереження $z \in \Omega$. Тут Ω називатимемо множиною спостережень. Оптимізація полягає в пошуку такого w^*, який мінімізує наступні витрати:

$$C(w) = E_z[L_z(w)] = \int_{z\in\Omega} L_z(w)\, dP(z)\,. \qquad (2.1.2)$$

Вважатимемо, що розподіл $dP(z)$ невідомий, $z\sim U[\Omega]$, а множина $\Omega-$ скінчена, $|\Omega| = N$.

2.2 Стохастичний градієнтний спуск

Для того, щоб оптимізувати (2.1.2) застосуємо наступний підхід. Замість розподілу $dP(z)$ використаємо емпіричний розподіл, визначений за допомогою Ω. Оскільки Ω скінчена, то емпіричний розподіл є дискретним. Тоді цільова функція матиме вигляд

$$C(w) = \frac{1}{N}\sum_{z\in\Omega} L_z\,(w). \qquad (2.2.1)$$

Функцію C називатимемо функціоналом емпіричного ризику (ФЕР).

Тепер можна переформулювати (1.2.2) в контексті нової задачі оптимізації:

$$w_{n+1} = w_n - h\nabla C(w_n) = w_n - h\int_{\Omega} \nabla L_z(w_n)\, dP(z) = w_n - \frac{h}{N}\sum_{z\in\Omega} \nabla L_z(w_n)\,. \quad (2.2.2)$$

Бачимо, що на кожній ітерації маємо обчислювати $L_z(w)$ для кожного $z \in \Omega$. Ключова відмінність стохастичного підходу полягає в тому, що на n-тій ітерації обчислюється $L_z(w)$ для кожного $z \in \mathrm{H}_n$, де $\mathrm{H}_n \subset \Omega$ та її вміст визначається випадково.

Розрізняють два основних підходи в стохастичному спуску – простий та кусковий. В простому випадку $|\mathrm{H}_n| = 1$. Такий спуск матиме вигляд

$$w_{n+1} = w_n - h\frac{1}{|\mathrm{H}_n|}\sum_{z\in\mathrm{H}_n} \nabla L_z(w_n) = w_n - h\nabla L_z(w_n). \qquad (2.2.3)$$

Кусковий (англ. Mini-Batch) стохастичний градієнтний спуск використовує $|\mathrm{H}_n| = m,\ 1 < m < N$. Такий підхід є компромісом між (2.2.2) та (2.2.3).

Формула кускового спуску має вид

$$w_{n+1} = w_n - h\frac{1}{|\mathrm{H}_n|}\sum_{z\in \mathrm{H}_n}\nabla L_z(w_n) = w_n - \frac{h}{m}\sum_{z\in \mathrm{H}_n}\nabla L_z(w_n). \qquad (2.2.4)$$

Розглянемо дві теореми, що забезпечують збіжність (2.1.1).

Теорема 1 (опуклий випадок). Для будь-якого розподілу $dP(z)$, якщо функція $C(w) = E_z[L_z(w)]$ диференційовна та має єдиний мінімум w^*, якщо цей мінімум задовольняє наступну умову

$$\forall h > 0\colon \inf_{|w-w^*|>h}(w - w^*)\nabla C(w) > 0$$

та припущення нижче справджуються:

а) $\forall w\colon\ E_z[Q(w)] = \nabla C(w)$;

б) значення кроків таке, що

$$\sum_{n=1}^{\infty} h_n = \infty,\ \sum_{n=1}^{\infty} h_n^2 < \infty\,;$$

в) $\exists A, B,\ \forall w\colon\ E_z[Q(w)^2] < A + B(w - w^*)^2$.

Тоді послідовність (w_n), визначена (2.1.1), збігається до w^* з ймовірністю 1.

Припущення (в) означає те, що стандартне відхилення випадкової функції $Q(w)$ не зростає швидше ніж лінійно з параметрами w.

Теорема 2 (загальний випадок). Для будь-якого розподілу $dP(z)$, якщо функція $C(w) = E_z[L_z(w)]$ тричі диференційовна та має обмежені другу та третю похідні, і якщо наступні припущення справджуються:

а) $\forall w\colon\ E_z[Q(w)] = \nabla C(w)$;

б) значення кроків таке, що

$$\sum_{n=1}^{\infty} h_n = \infty,\ \sum_{n=1}^{\infty} h_n^2 < \infty;$$

в) $\exists A, B,\ \forall w\colon\ E_z[Q(w)^2] < A + B(w - w^*)^2$;

г) $\exists C_{\min},\ \forall w\colon\ C_{\min} < C(w)$.

Тоді, визначені за (2.1.1), $C(w_n)$ збігається з ймовірністю 1 та $\nabla C(w_n)$ збігається до 0 з ймовірністю 1.

Тут доречно зауважити, що рівність градієнта нулю в деякій точці не означає існування екстремуму в цій точці. Це може бути «сідлова» точка.

2.3 Алгоритм стохастичного градієнтного спуску

Далі наведено алгоритм для (2.1.1):

а) задаємо початкове наближення w_0 та крок h;

б) обираємо $m = |\mathrm{H}_n|\ \ \forall n \in \mathbb{N},\ 1 \le m \le |\Omega|, m \in \mathbb{N}$;

в) обираємо $k \in \mathbb{N}$;

г) індекс поточного кроку $n = 0$;

ґ) циклічно повторюємо наступні кроки:

1) випадково генеруємо $\mathrm{H}_n \subset \Omega$, вибір кожного $z \in \Omega$ рівноможливий;

2) виконуємо крок (2.2.4);

3) знаходимо витрати за поточними спостереженнями

$$\mathcal{E}_n = \sum_{z \in \mathrm{H}_n} L_z(w_n); \qquad (2.3.1)$$

4) якщо $\mathcal{E}_{n-k} \le \mathcal{E}_K\ \forall\ K = \overline{n-k, n}$, тобто оцінка втрат декілька ітерацій підряд не зменшується, то зупиняємо алгоритм.

Метод стохастичного градієнту (2.1.1) має декілька теоретичних та практичних переваг над класичним градієнтним спуском (1.2.2):

– збіжність стохастичного алгоритму набагато швидша у випадку надлишкового Ω. В середньому, тільки декілька її елементів потрібно для представлення еквіваленту кроку класичним градієнтним спуском;

– класичний градієнтний спуск часто збігається до локального мінімуму цільової функції. Алгоритм не зможе «вистрибнути» із цього мінімуму, котрий, скоріш за все, не є вдалим розв'язком задачі оптимізації. На практиці класичний алгоритм часто не може вийти із області, де функція витрат є погано обумовленою, тобто надчутливою до зміни аргументу. Стохастичний алгоритм зазвичай уникає такі «пастки» та більш схильний до збіжності в точці глобального мінімуму;

– якщо потужність Ω замала для повноцінної оптимізації, то на кожному кроці спуску можна генерувати нові z із невідомого розподілу $dP(z)$. Наприклад, практикують застосування суміші гауссівських розподілів для оцінки щільності розподілу $dP(z)$ на базі вибірки Ω;

– можна застосовувати стохастичний спуск навіть якщо функція $L_z(w)$ є диференційовною майже скрізь. Похідна в цій точці може бути замінена на будь-яке значення, якщо

$$E_z[\nabla L_z(w)] = \nabla C(w) = \nabla E_z[L_z(w)].$$

РОЗДІЛ 3. НЕЙРОННІ МЕРЕЖІ ПРЯМОГО РОЗПОВСЮДЖЕННЯ ТА ЇХ НАВЧАННЯ

3.1 Основні поняття нейронної мережі

Нейронна мережа – це сукупність нейронів, структурних одиниць, які зберігають в собі інформацію, а також обмінюються нею за допомогою визначених зв'язків. Дане формулювання є універсальним для довільного контексту, тобто довільної дисципліни, в якій використовується запропоноване визначення. Вперше про нейронні мережі почали говорити як про мозкову структуру людини. Незважаючи на початкову біологічну природу даного визначення, цей термін знайшов свою інтерпретацію у математиці, машинному навчанні, обчислювальній техніці та навіть філософії.

З точки зору математичної моделі нейронна мережа являє собою апроксимуючу функцію, що встановлює залежність між вхідним та вихідним сигналами, $f(x_1, x_2, \dots, x_n) = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{pmatrix}$, де $X = (x_1, x_2, \dots, x_n)$ називають вхідним сигналом або спостереженням (англ. observation), а $Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{pmatrix}$ вихідним сигналом або прогнозом (англ. prediction). Тобто з інтуїтивної точки зору нейронна мережа – це відображення із спостереження у прогноз.

Наступна сукупність об'єктів є фундаментальними складовими, з яких будується довільна нейронна мережа:

– нейрон – це структурна одиниця, котра акумулює інформацію, отриману від інших нейронів або зовнішнього середовища та передає

іншим нейронам чи зовнішньому середовищу. Під інформацією розуміють деяке дійсне число. Інформацію, що отримав нейрон, часто називають активацією (англ. activation) або сигналом. Будемо використовувати ці три терміни як синоніми. Кажучи про передачу інформації, говорять, що деякий нейрон активує інший. Кажучи про отримання інформації, говорять, що деякий нейрон активується іншим. Нейрон, що активує, називають нейроном-активатором. Нейрон, що активується, називають нейроном-приймачем;

– синапс або зв'язок – сполучення двох нейронів. Множина синапсів визначає комунікацію між нейронами, тобто шлях, який сигнал долає від входу до виходу та під час якого зазнає трансформацій. Кожен окремий зв'язок має свій індивідуальний параметр – вагу. Вона визначає міру впливу нейрона-активатора на нейрон-приймач. Також кожна сукупність зв'язків, що веде до одного нейрона, розділяє спільний параметр – зсув. Він, аналогічно вазі, визначає міру впливу, проте не одного нейрона на інший, а на сукупність всіх нейронів-активаторів щодо деякого нейрона-приймача;

– цільова функція або функція втрат (витрат) – функція, яка оцінює відповідність вихідного сигналу нейронної мережі очікуваному значенню. Вона є мірою того, на скільки вихід на певній множині спостережень є очікуваним;

– функція активації або функція збудження – функція, яка нормалізує інформацію перед передачею до наступного нейрона. Головною задачею цього елементу є впровадження нелінійної складової у модель, що дозволяє їй апроксимувати складні взаємозв'язки між входом та виходом. Дана процедура також забезпечує потрібну збіжність градієнтного спуску. Деякі її різновиди пришвидшують навчання мережі

та не допускають переповнення (англ. overflow) під час комп’ютерних обчислень.

Значення ваг синапсів та зсувів називають параметрами мережі (моделі), а значення всіх цих параметрів визначають поняття стану мережі. Кількість нейронів, їх організація у структури вищого порядку, структура зв’язку між ними, вид цільової функції та функції активації називають гіперпараметрами мережі (моделі).

3.2 Структура нейронної мережі

Описавши набір складових об’єктів, розглянемо детальніше структуру нейронної мережі [6]. Перш за все зазначимо, що нейронні мережі найчастіше класифікують за типом з’єднань між нейронами та їх загальною організацією у складові вищого порядку. Різні види організацій породжують окремі класи архітектури, що застосовуються для роз’язання досить різних задач. Наприклад, архітектура, де наявні синапси, які з’єднують нейрон з самим собою, називається рекурентною. Рекурентні нейронні мережі підходять для задач прогнозування послідовних даних, одними з яких є часові ряди. Інший вид архітектури розділяє одні й ті самі значення ваг із різними зв’язками. Такий підхід часто застосовується в згорткових нейронних мережах, які застосовуються на даних типу зображень. Крім цього, в останні дні популярними стають графові нейронні мережі, які будуються за окремим алгоритмом [7].

В рамках даної роботи буде розглянуто нейронні мережі прямого розповсюдження (англ. Feedforward Neural Networks). Ключовим в такій архітектурі є те, що сигнал проходить від входу до виходу лише в одному напрямку. Структуру такої мережі утворюють шари, які розташовані послідовно один за одним, а сигнал ітеративно передається між сусідніми шарами. Кожен шар представляє собою сукупність нейронів, які одночасно приймають сигнал на певному кроці роботи системи. Приклад графічного зображення такої системи наведено на рисунку 3.2.1. Кола зображають нейрони, а стрілки – зв'язки та напрямок передачі сигналу.

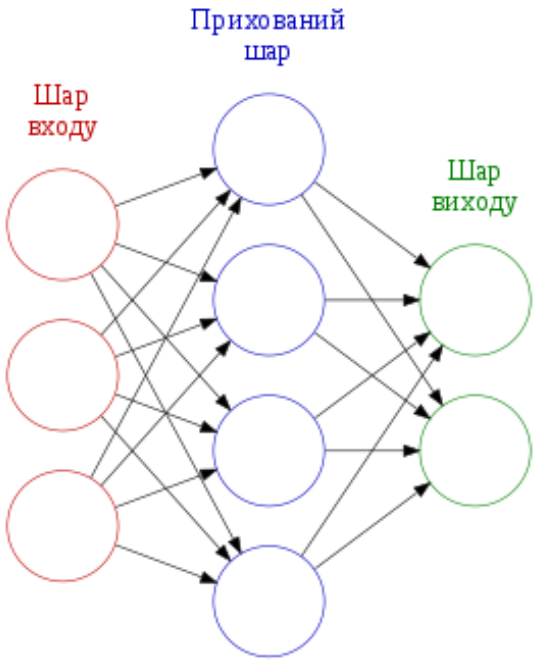

Рисунок 3.2.1 – Структура нейронної мережі прямого розповсюдження [8]

Перший шар, який називається вхідним, в розглядуваній архітектурі є обов'язковим та єдиним. Він приймає вектор вхідних даних *X* та передає його до наступного шару – прихованого (внутрішнього). Його «прихованість» полягає в тому, що поведінка сигналу всередині нього явним чином не

контролюється і взагалі проміжні сигнали не є значущими самі по собі, а служать задля трансформації вхідного сигналу і отримання апроксимації на виході, частіше за все нелінійної. Прихований шар є своєрідною «чорню скринькою» нейронної мережі. Прихованих шарів, а також кількості нейронів в них, може бути довільна кількість. Існує безліч робіт, пов’язаних із їх побудовою та вже готових шаблонних рішень, які спеціалізуються на конкретних типах задач. Проте не існує єдиного оптимального алгоритму визначення найкращої архітектури мережі, оскільки кожній окремій задачі краще підходять різні архітектури. Хоча, тридцять років тому американським математиком Джорджем Цибенко було доведено універсальну теорему апроксимації [9], яка стверджує, що для довільної неперервної функції існує нейронна мережа певної структури, що апроксимує її із довільною точністю. Однак, з практичної точки зору ця теорема ніяким чином не допомагає побудувати ефективну архітектуру. Нейронні мережі, що мають більш ніж один шар у прихованій частині, називають глибинними.

Останній шар називається вихідним. Він видає вихідний сигнал, який може бути представлений як в одному нейроні (одне дійсне число), так і в шарі з декількома нейронами (вектор дійсних чисел). В рамках даної роботи вихідний шар – єдиний. Зазвичай структура вихідного шару детермінується типом задачі. Наприклад, для задачі класифікації достатньо стільки нейронів на виході, скільки існує класів. Кожен нейрон відповідатиме вірогідності належності спостереження відповідному класу. В задачі регресії достатньо одного вихідного нейрона, значення якого і буде чисельний прогноз.

3.3 Принцип навчання нейронної мережі з вчителем

Навчити нейронну мережу означає надати їй такий стан, який забезпечує апроксимацію деякої функції з потрібною точністю. Для того, щоб нейронна мережа апроксимувала величину з такою точністю, вона повинна мати інструмент оцінки якості вихідного сигналу. Функція втрат (витрат) і є цим інструментом. Вона залежить від значення вихідного сигналу мережі та очікуваних значень цього ж сигналу. Кажучи про останнє, необхідно мати набір таких спостережень, точність прогнозування яких можна оцінити. Інакше кажучи, потрібен набір вхідних даних разом із очікуваним вихідним сигналом. Таких набір називається навчальною (тренувальною) вибіркою. Забігаючи наперед, в застосуванні стохастичного градієнтного спуску в навчанні мережі саме ця вибірка і буде множиною Ω. На базі множини таких спостережень, яких також називають маркірованими (англ. labeled observations), навчають мережу, порівнюючи фактичний та очікуваний вихідний сигнал. В ієрархії алгоритмів машинного навчання такий підхід відносять до класу навчання з вчителем (англ. supervised learning). Надалі будемо говорити про навчання мережі в контексті навчання з вчителем.

Щодо функціонування мережі, можна виділити два ключових етапи – навчання та прогнозування. Такий підхід дуже схожий на процес навчання дітей. Спочатку дитина отримує досвід, який передають їй батьки. Звичайно, отриманої інформації недостатньо, щоб покрити всі можливі життєві ситуації. Проте суть навчання полягає в екстраполяції оцінки відомих спостережень на невідомі. Наприклад, якщо дитину навчили малювати червоним олівцем, то далі вона зможе зробити те саме і синім кольором, тобто без необхідності заново вчитися малювати. Така сама ідея і лежить в основі методу навчання з вчителем, де вчителем фактично виступає навчальна вибірка.

Стадія навчання мережі, у свою чергу, має два основних етапи – прямий хід та зворотній хід. Прямий хід полягає в передачі інформації від нейронів вхідного до нейронів вихідного шару, а зворотній хід – навпаки. Метою прямого ходу є отримання оцінки вихідного сигналу, а зворотного – корекція стану мережі задля покращення точності апроксимації на даному наборі спостережень.

Розглянемо детальніше прямий хід. Активація нейрона відбувається за допомогою активацій нейронів попереднього шару та ваг, що їх з'єднують, зсуву, а також функції активації. Таким чином, процес розповсюдження інформації є рекурсивним відносно шару. Передача інформації виконується спочатку між нейронами першого та другого шарів, далі другого та третього і так до самого кінцевого. Ілюстративна активація нейрона зображена на рисунку 3.3.1. Із всіх нейронів, підключених до певного одного, рахується сума попарно перемножених ваг та активацій плюс зсув. Отримане значення передається функції активації та, врешті-решт, нейрон-приймач активується підрахованим числом.

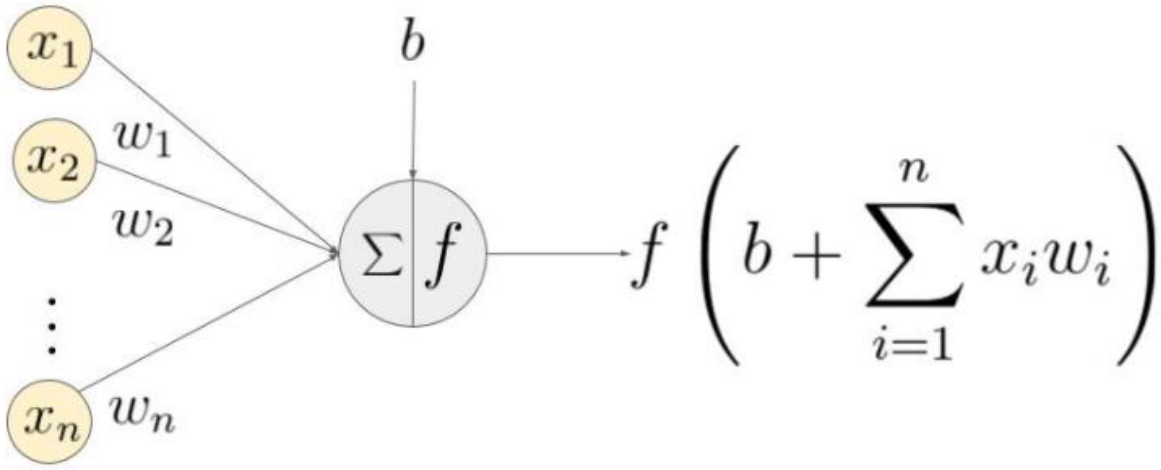

Рисунок 3.3.1 – Підрахунок активації нейрона [10]

Після того, як активувався вихідний шар, починається зворотній хід. Метод, який його виконує, називається зворотнім поширенням помилки. Оскільки вибірка тренувальна, то маємо інформацію про те, як «повинен» активуватись останній шар. Ключовою ідеєю даного методу є розрахунок впливу кожного параметру мережі (складових її сану) на вихідний сигнал, а далі відбувається корекція параметрів, які негативно вплинули на точність апроксимації.

Введемо узагальнене поняття. Помилка параметру нейронної мережі – міра того, на скільки даний параметр негативно впливає на точність прогнозу. Почнемо міркування від меншого до більшого. В контексті запропонованої «помилки» введемо ще декілька понять.

Помилка нейрона по відношенню до зв'язку – це величина, котра визначається комбінацією наступних факторів:

– загальна помилка нейрона-приймача по даному зв'язку;

– вага даного зв'язку;

– міра впливу функції активації на даний зв'язок.

Всі величини, окрім першої, відомі. Тоді визначимо загальну помилку нейрона.

Загальна помилка нейрона – це сума всіх помилок даного нейрона по відношенню до кожного зв'язку, де даний нейрон виступає активатором. Загальна помилка для нейронів вихідного шару визначається тренувальною вибіркою. Загальна помилка нейронів в довільному шарі визначається за допомогою загальної помилки нейронів у наступному шарі. Таким чином, можна встановити загальну помилку для кожного нейрона, виконавши обхід мережі у зворотному напрямку. Хоча, як вже було сказано, немає інструментів прямого впливу на активації нейронів, а, відповідно, і впливу на їх загальну помилку, актуальність даної величини буде зрозуміла далі.

Помилка зв’язку – це величина, котра визначається комбінацією наступних факторів:

– загальна помилка нейрона-приймача по даному зв’якзу;

– активація нейрона, що передає інформацію по зв’язку;

– міра впливу функції активації на даний зв’язок.

Помилка зсуву – це величина, котра визначається комбінацією наступних факторів:

– загальна помилка нейрона-приймача, якому відповідає зсув;

– міра впливу функції активації на даний зсув.

Помилки зв’язку та зсуву – це ключова інформація при навчанні мережі, оскільки зв’язки та зсуви є параметрами мережі, а на них вже можна впливати безпосередньо. Бачимо, що помилки зв’язку та зсуву залежать від загальної помилки нейрона та інших відомих величин. Сукупність помилок для всіх зв’язків та зсувів утворюють градієнт, котрий буде використаний в методі стохастичного градієнтного спуску. Знаходження цього вектору і є метою зворотного ходу. Формальний опис цього процесу буде безпосередньо наведено у алгоритмі навчання нейронної мережі прямого розповсюдження з вчителем.

Повертаючись до другого етапу функціонування мережі, прогнозування за допомогою навченої мережі – це виконання прямого ходу за тим самим принципом, що і під час навчання, але вже на нових вхідних даних, які не є маркірованими та не є частиною тренувальної вибірки. На цьому етапі вагові коефіцієнти та зсуви налаштовані таким чином, що прогноз має певну точність апроксимації.

3.4 Принцип вибору та вплив гіперпараметрів нейронної мережі

Як вже раніше зазначалось, гіперпараметрами нейронної мережі є фактично всі її елементі, які утворюють архітектуру нейронної мережі, а також функції активації та цільові функції. Загалом, гіперпараметрами довільної моделі називають такі її складові, які мають бути задані на етапі конструювання моделі. Наприклад, функція втрат має бути обрана проектувальником мережі відповідно до типу задачі апроксимації, предметної області розв'язуваної задачі, типу тренувальних даних. Так само і структура мережі має бути задекларована перед початком її навчання.

Важливо розуміти, що гіперпараметри моделі дуже часто тісно пов'язані із параметрами моделі, а саме впливають на їх кількість. Наприклад, вибір кількості нейронів в деякому шарі є гіперпараметром моделі, проте цей вибір напряму визначає кількість зв'язків та зсувів, що мають відношення до даного шару. Так само і вибір кількості шарів впливає на кількість параметрів моделі.

Основною причиною надзвичайної потужності нейронних мереж є можливість через гіперпарамтери моделі визначити для неї довільну кількість параметрів, а чим більше параметрів має модель – тим вона складніша, тобто має більшу ємність для зберігання залежностей між входом та виходом. Кількість параметрів моделі інколи також називають ступенями свободи моделі (англ. degrees of freedom). Цей термін більш відомий зі статистики і має дещо спільне із його значенням в контексті моделей. Отже, фактично необмежена ступінь свободи моделі, хіба що обчислювальними ресурсами, робить нейронні мережі поза конкуренцією із багатьма іншими підходами в апроксимації складних взаємозв'язків.

Переходячи до розгляду конкретних гіперпараметрів, один із важливих виборів під час побудови моделі нейронної мережі – функція втрат (витрат).

Вона є головним інструментом маніпуляції вектором градієнту, тобто задає «напрямок» оптимізації.

Розглядаючи задачу регресії, найчастіше використовують середньо-квадратичну помилку (англ. mean squared error, MSE), яка розраховується за формулою

$$MSE(\hat{y}_j, y_j) = \frac{1}{2n}\sum_{j=1}^{n}\left(\hat{y}_j - y_j\right)^2. \tag{3.4.1}$$

Тут y_j – справжнє (очікуване) значення, а $\hat{y}_j$ – відповідне значення прогнозу. Також, але рідше, застосовують середню абсолютну помилку, тобто

$$MAE(\hat{y}_j, y_j) = \frac{1}{n}\sum_{j=1}^{n}|\hat{y}_j - y_j|. \tag{3.4.2}$$

Дана функція втрат більш актуальна для ситуацій, де навчальна вибірка містить багато викидів. Третім, збалансованим по відношенню до попередніх, варіантом є функція втрат Хьюбера, яка є квадратичною на малих втратах та лінійною на великих. Її формула має вигляд

$$L_{\delta}(\hat{y}_j, y_j) = \begin{cases} \frac{1}{2n}\sum_{j=1}^{n}\left(\hat{y}_j - y_j\right)^2, & \frac{1}{n}\sum_{j=1}^{n}|\hat{y}_j - y_j| < \delta \\ \frac{\delta}{n}\sum_{j=1}^{n}|\hat{y}_j - y_j| - \frac{1}{2}\delta^2, & \text{інакше} \end{cases} \tag{3.4.3}$$

Число δ обирається заздалегідь. Даний підхід також застосовують при наявності викидів у тренувальних даних та він зазвичай краще за (3.4.2).

Для задач класифікації найпопулярнішою функцією втрат є логістичні втрати (англ. log loss). Формула перехресної ентропії для бінарної класифікації має вигляд

$$L(\hat{y}_j, y_j) = -\frac{1}{n}\left[\sum_{j=1}^{n} y_j log(\hat{y}_j) + (1 - y_j) log(1 - \hat{y}_j)\right]. \tag{3.4.4}$$

Тут варто зазначити, що поняття логістичних втрат є частковим випадком перехресної ентропії (англ. cross entropy), мова про останнє частіше йдеться у випадку мультикласової задачі. Також окрім згаданих функцій, рідше застосовують показникову функцію помилки та відстань Кульбака-Лейблера (приріст інформації).

Інший важливий гіперпараметр мережі – функція активації. Як вже було зазначено, головна роль даної складової є впровадження нелінійності у модель. Існує численна кількість можливих рішень. Виділимо більш поширені функції активації та розглянемо їх по порядку [11]:

– функція Хевісайда: $f(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases}$;

– лінійна функція $f(x) = x$;

– сигмоїда (логістична крива): $f(x) = \frac{1}{1+e^{-x}}$;

– тангенс гіперболічний: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$;

– ReLU (rectified linear unit): $f(x) = \max(0, x)$.

Щодо східчастої функції Хевісайда, то на практиці її не використовують. Вона була одна з перших функцій активації та відома через так званий «перцептрон Розенблатта», що став історичним об’єктом в історії нейронних мереж . Наразі є набагато кращі альтернативи, перевага яких строго доведена. Суть відносної неефективності функції Хевісайда полягає в тому, що не бажано надавати нейронам дискретні активації.

Вибір же лінійної функції має дуже неприємний наслідок. Якщо функція лінійна, то відповідно її похідна постійна. Тоді градієнт є постійним, тобто він не залежить від входу. Це фактично ліквідує осмислене використання градієнтного спуску та методу зворотного поширення помилки. Більш того, взагалі не має значення скільки шарів є в нейронній мережі при виборі такої функції активації. Останній шар буде лише лінійною комбінацією першого.

Таким чином, всі шари «згортаються» в два – вхідний та вихідний. Це пояснюється тим, що комбінація лінійних функцій – це теж лінійна функція. А нейронна мережа без прихованих шарів – це ніщо інше, як модель лінійної регресії, яка має значно меншу потужність за нейронну мережу з нелінійною складовою. Тим більше, звичайна модель лінійної регресії виконує навчання значно швидше ніж така ірраціональна модель.

Значно кращими альтернативами за попередні є сигмоїда, зображена на рисунку 3.4.1, або тангенс гіперболічний, при чому друга зазвичай переважає першу. Дані функції досить схожі, проте мають різну область значень. Обидві вносять нелінійну складову в модель. Проте, наявність горизонтальних асимптот може призвести до так званого «паралічу мережі». Таке трапляється, коли більшість активацій є великими значеннями, що призводить нульового градієнту та, як наслідок, навчання завмирає. Іншим недоліком цих функцій є відносно повільне обчислення. Тому на сьогоднішній день їх майже не використовують при навчанні нейронних мереж прямого розповсюдження. Проте, вони є актуальними в архітектурах іншого типу, як тангенс гіперболічний яскраво підходить для навчання рекурентних нейронних мереж.

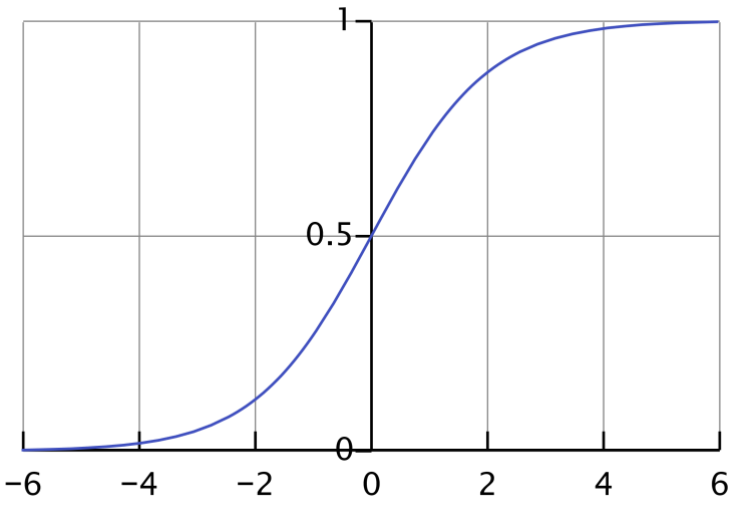

Рисунок 3.4.1 – Графік сигмоїди [12]

На сьогоднішній день фаворитом у світі функцій активацій є ReLU. Її перевага полягає в тому, що вона виконує так звану «розріджену» активацію, яка супроводжується меншими обчислювальними витратами. При чому розрахунок самої функції також є швидким. Емпірично встановлено, що збіжність стохастичного градієнтного спуску із застосуванням ReLU значно швидша ніж при використанні тангенса гіперболічного чи сигмоїди. Недоліком цієї функції активації є «проблема вмираючого ReLU» (англ. The Dying ReLU problem), яка пов'язана із нульовим градієнтом та, як наслідок, унеможливленням навчання. Проте, для подолання цієї проблеми і не тільки було створено ціле сімейство ReLU-подібних функцій, а саме нещільна ReLU (LReLU), параметрична ReLU (PReLU), ReLU з шумом (RReLU), гладка ELU.

Вибір функції активації є дуже важливим аспектом та частіше базується на компромісі між швидкістю навчання та якістю збіжності. Існує багато функцій і не менше проблем, які вони можуть породжувати або вирішувати. Також варто додати, що інколи практикують використання різних функцій активації у різних шарах нейронної мережі.

3.5 Алгоритм навчання нейронної мережі з вчителем

В даному підрозділі наведено алгоритм навчання нейронної мережі з вчителем із деякою заданою структурою та зв'язками, визначеними функціями витрат та активації, навчальною вибіркою.

Введемо наступні позначення:

$-\ k -$ індекс шару. $k \in \overline{1,K},\ K \in \mathbb{N},\ K \geq 2$;

$-\ n_k -$ кількість нейронів у k-му шарі, $k \in \overline{1,K}$;

$-\ a_n^{(k)} -\ n$-тий нейрон у k-му шарі, $k \in \overline{1,K}$;

$-\ a^{(k)} = \left(a_1^{(k)}, a_2^{(k)}, \dots, a_{n_k}^{(k)}\right)^T, k \in \overline{1,K}$;

$-\ b_n^{(k)} -$ n-тий зсув у k-му шарі, $k \in \overline{2,K}$;

$-\ b^{(k)} = \left(b_1^{(k)}, b_2^{(k)}, \dots, b_{n_k}^{(k)}\right)^T, k \in \overline{2,K}$;

$-\ w_{ij}^{(k)} -$ вага синапсу, що з'єднує j-тий та i-тий нейрони $(k-1)$-го та k-го шарів, $k \in \overline{2,K},\ i \in \overline{1,n_k},\ j \in \overline{1,n_{k-1}}$;

$$-\ W^{(k)} = \begin{pmatrix} w_{11}^{(k)} & w_{12}^{(k)} & \dots & w_{1j}^{(k)} & \dots & w_{1n_{k-1}}^{(k)} \\ w_{21}^{(k)} & w_{22}^{(k)} & \dots & w_{2j}^{(k)} & \dots & w_{2n_{k-1}}^{(k)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{i1}^{(k)} & w_{i2}^{(k)} & \dots & w_{ij}^{(k)} & \dots & w_{in_{k-1}}^{(k)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{n_k1}^{(k)} & w_{n_k2}^{(k)} & \dots & w_{n_kj}^{(k)} & \dots & w_{n_kn_{k-1}}^{(k)} \end{pmatrix}, k \in \overline{2,K};$$

$-\ w^{(k)} = \left(w_{11}^{(k)}, \dots, w_{1n_{k-1}}^{(k)}, b_1^{(k)}, w_{21}^{(k)}, \dots, b_2^{(k)}, \dots \dots, w_{n_k1}^{(k)}, \dots, w_{n_kn_{k-1}}^{(k)}, b_{n_k}^{(k)}\right)^T,$ $k \in \overline{2,K}$;

$-\ \sigma$ – функція активації;

$-L -$ функція втрат, $L_z -$ втрати на спостереженні z;

$-\ C -$ функціонал емпіричного ризику;

$-\ \Omega = \{z_j\}_{j=\overline{1,J}}, J \in \mathbb{N}$, де

$z_j = (x_j, y_j) -$ пара, яка визначає спостереження, тобто

$x_j = \left(x_1^{(j)}, x_2^{(j)}, \dots, x_{n_1}^{(j)}\right)^T -$ тестові вхідні дані,

$y_j = \left(y_1^{(j)}, y_2^{(j)}, \dots, y_{n_K}^{(j)}\right)^T -$ вихідні дані для x_j;

$-\ \mathrm{H}_n \subset \Omega,\ |\mathrm{H}_n| = S\ \ \forall n \in \mathbb{N},\ 1 \le S \le J, S \in \mathbb{N}$, вибір кожного $z \in \Omega$ для H_n є рівноможливим.

Алгоритм:

а) ініціалізуємо $W^{(k)}$ за допомогою генератора псевдовипадкових чисел, $b^{(k)} = \vec{0},\ \forall k \in \overline{2,K}$;

б) індекс першої ітерації $n = 0$;

в) поки оцінка втрат L в цілому зменшується впродовж ітерацій:

1) генеруємо H_n;

2) $\forall z = (x, y) \in \mathrm{H}_n$:

– подаємо дані вхідному шару. $a^{(1)} = x$;

– рекурсивно знаходимо $a^{(K)}$ із такого співвідношення:

$a^{(k)} = \sigma\left(q^{(k)}\right),$ де $q^{(k)} = W^{(k)}a^{(k-1)} + b^{(k)},\ k \in \overline{2,K}.$

Увага! При цьому стан мережі не змінюється, метою є знаходження $a^{(K)}$;

– $\hat{y} = a^{(K)}$ представляє прогноз для z, тобто оцінку y;

– задаємо «відправну точку» для методу зворотного поширення помилки: $\forall j \in \overline{1,n_K}$ підраховуємо

$$\frac{\partial L_z}{\partial a_j^{(K)}}(y, \hat{y});$$

– $\forall\ k \in \overline{2,K},\ \forall\ j \in \overline{1,n_k}$ рекурсивно з кінця знаходимо:

$$\frac{\partial L_z}{\partial a_j^{(k-1)}} = \sum_{i=1}^{n_k} \frac{\partial L_z}{\partial a_i^{(k)}} \frac{\partial a_i^{(k)}}{\partial q_i^{(k)}} \frac{\partial q_i^{(k)}}{\partial a_j^{(k-1)}}.$$

Раніше цю величину ми назвали загальною помилкою нейрона;

– тепер для довільного зв'язку $w_{ij}^{(k)}$ та зсуву $b_i^{(k)}$, $k \in \overline{2,K}$, $i \in \overline{1,n_k},\ j \in \overline{1,n_{k-1}}$ можна визначити

$$\frac{\partial L_z}{\partial w_{ij}^{(k)}} = \frac{\partial L_z}{\partial a_i^{(k)}} \frac{\partial a_i^{(k)}}{\partial q_i^{(k)}} \frac{\partial q_i^{(k)}}{\partial w_{ij}^{(k)}},$$

$$\frac{\partial L_z}{\partial b_i^{(k)}} = \frac{\partial L_z}{\partial a_i^{(k)}} \frac{\partial a_i^{(k)}}{\partial q_i^{(k)}} \frac{\partial q_i^{(k)}}{\partial b_i^{(k)}}.$$

Раніше ці величини ми назвали помилкою зв'язку та помилкою зсуву відповідно.

3) підраховуємо середні показники показники помилок

$$\frac{\partial C}{\partial w_{ij}^{(k)}} = \frac{1}{S} \sum_{z \in \mathrm{H}_n} \frac{\partial L_z}{\partial w_{ij}^{(k)}},$$

$$\frac{\partial C}{\partial b_i^{(k)}} = \frac{1}{S} \sum_{z \in \mathrm{H}_n} \frac{\partial L_z}{\partial b_i^{(k)}};$$

4) вектор, що є сукупністю всіх помилок зв'язків та зсувів

$$\left(\frac{\partial L}{\partial w_{11}^{(k)}}, \dots, \frac{\partial L}{\partial w_{1n_{k-1}}^{(k)}}, \frac{\partial L}{\partial b_1^{(k)}}, \frac{\partial L}{\partial w_{21}^{(k)}}, \dots, \frac{\partial L}{\partial b_2^{(k)}}, \dots \dots, \frac{\partial L}{\partial w_{n_k 1}^{(k)}}, \dots, \frac{\partial L}{\partial w_{n_k n_{k-1}}^{(k)}}, \frac{\partial L}{\partial b_{n_k}^{(k)}} \right)^T$$

являє собою ∇C для поточного H_n;

5) робимо крок методом стохастичного градієнту (2.1.1)

$$w_{n+1} = w_n - h\nabla \mathrm{C};$$

6) фіксуємо оцінку сукупних втрат за (2.3.1);

7) $n = n + 1$.

РОЗДІЛ 4. ОБЧИСЛЮВАЛЬНИЙ ЕКСПЕРИМЕНТ ТА АНАЛІЗ МЕТОДІВ НАВЧАННЯ МЕРЕЖІ

4.1 Шляхи вдосконалення класичних методів

Тренування глибинних нейронних мереж буває дуже важким з точки зору обчислювальних ресурсів, що призводить до тривалого процесу оптимізації. У випадку, коли необхідно провести пошук найкращих гіперпараметрів мережі, цей процес може бути дуже повільним або взагалі неможливим в межах доступного часу. Для вирішення цієї проблеми було розроблену цілу низку різновидів градієнтних методів, які випереджають класичний стохастичний градієнтний спуск (2.1.1) у швидкості навчання.

Як зазначалось, навчання за допомогою стохастичного градієнтного спуску є значно швидшим за класичний градієнтний спуск, хоча і швидкість збіжності є нижчою. Тут варто зазначити, що під швидкістю навчання мається на увазі фактичний час тривалості оптимізації, який можна виміряти у секундах, хвилинах тощо, а під швидкістю оптимізації розуміють кількість виконаних ітерацій градієнтного спуску. Отже, класичний градієнтний спуск потребує меншу кількість кроків для виконання оптимізації ніж стохастична модифікація, проте кожен таких крок значно складніший обчислювально за крок стохастичного градієнтного спуску.

Проте, окрім двох вищезгаданих методів оптимізації навчання нейронної мережі є ще купа інших модифікацій, які є похідними від стохастичного градієнтного спуску. Загалом, їх метою є пришвидшення процесу оптимізації, тобто зменшення кількості кроків. Зазвичай ціною використання таких модифікацій є невелике зменшення якості збіжності, тобто зупинення процесу оптимізації не в точці глобального мінімуму, а десь

поряд. Надалі буде розглянуто метод імпульсної оптимізації, пришвидшений градієнт Нестерова, адаптивний градієнтний алгоритм.

4.2 Градієнтний спуск з імпульсом

Уявімо, як куля для боулінгу котиться по пологому схилу з гладкою поверхнею: спочатку вона почне повільно, але швидко набере імпульс, поки врешті не досягне кінцевої швидкості (якщо є якесь тертя або опір повітря). Це дуже проста ідея поза оптимізацією з імпульсом, запропонована Борисом Поляком в 1964 році [13]. На відміну від цього, звичайний стохастичний градієнтний спуск буде робити невеликі регулярні кроки вниз по схилу, тому алгоритму знадобиться набагато більше кроків, щоб досягти дна (мінімуму).

Нагадаємо, що стохастичний градієнтний спуск оновлює ваги w_n безпосередньо віднімаючи градієнт функції витрат $C(w_n)$ щодо ваг, помножених на швидкість навчання h. Даний підхід не враховує якими були попередні градієнти. Якщо локальний градієнт незначний, то і крок робиться малий.

Оптимізація імпульсним методом враховує якими були попередні градієнти: на кожній ітерації він віднімає локальний градієнт від вектору імпульсу m (помноженого на швидкість навчання h) і оновлює ваги, додаючи цей вектор імпульсу (див. рівняння (4.2.1)). Іншими словами, градієнт використовується для прискорення, а не для швидкості. Щоб імітувати якийсь механізм тертя та запобігати збільшенню імпульсу, алгоритм вводить новий гіперпараметр β, який називається імпульсом та повинен бути встановлений між 0 (високе тертя) та 1 (відсутність тертя). Типове значення імпульсу становить 0,9.

$$\begin{cases} m_{n+1} = \beta m_n - h\nabla \mathrm{C}(w_n) \\ w_{n+1} = w_n + m_{n+1} \end{cases} \quad (4.2.1)$$

Можна легко перевірити, що якщо градієнт залишається незмінним, то кінцева швидкість (тобто максимальний розмір оновлень ваги) дорівнює цьому градієнту, помноженому на швидкість навчання h та на $\frac{1}{1-\beta}$ (ігноруючи знак). Наприклад, якщо β = 0,9, то кінцева швидкість дорівнює 10-кратному градієнту, помноженому на швидкість навчання, тому оптимізація імпульсу виконується у 10 разів швидше, ніж за звичайного стохастичного градієнтного спуску. Це дозволяє методу оптимізації імпульсу проходити ділянки функції з низьким градієнтом, або так звані плато, набагато швидше. Даний підхід також може допомогти пройти повз локальних оптимумів.

Зауважимо, що через сильний імпульс оптимізатор може пролетіти точку мінімуму, потім повернутися, перескочити ще раз і коливатися так багато разів, перш ніж стабілізуватися в цій точці. Це одна з причин, чому добре мати трохи тертя в системі: воно позбавляється від цих коливань і, таким чином, пришвидшує збіжність.

4.3 Метод прискореного градієнта Нестерова

Один невеликий різновид оптимізації імпульсу, запропонований Юрієм Нестеровим у 1983 році [14], майже завжди швидший, ніж оптимізація імпульсним методом (4.2.1). Метод прискореного градієнта Нестерова (англ. Nesterov accelerated gradient, NAG), також відомий як оптимізація імпульсу Нестерова, вимірює градієнт функції витрат не в локальному положенні w, а трохи вперед у напрямку імпульсу, при $w + \beta m$ (див. рівняння (4.3.1)).

$$\begin{cases} m_{n+1} = \beta m_n - h\nabla \mathrm{C}(w_n + \beta m_n) \\ w_{n+1} = w_n + m_{n+1} \end{cases} \quad (4.3.1)$$

Ця невелика зміна корисна, оскільки загалом вектор імпульсу буде спрямований у більш правильному напрямку, тому буде трохи точніше використовувати градієнт, виміряний трохи далі в цьому напрямку, а не градієнт у вихідному положенні. Через деякий час ці невеликі вдосконалення складаються, і NAG виявляється значно швидшим, ніж звичайна оптимізація імпульсу. Більше того, коли імпульс штовхає ваги по плато, звичайний імпульсний градієнт продовжує просуватися далі в тому ж напрямку, тоді як імпульсний градієнт за Несторовим відштовхується до дна плато. Це допомагає зменшити коливання і, таким чином, NAG збігається швидше.

4.4 Адаптивний градієнтний спуск

Розглянемо проблему оптимізації функції втрат з поверхнею у формі витягнутої чаші: градієнтний спуск починається швидким рухом по найкрутішому схилу, який не вказує прямо до загального оптимуму, а потім дуже повільно опускається до дна. Було б добре, якби алгоритм міг виправити свій напрямок раніше, щоб трохи точніше вказувати на глобальний оптимум. Алгоритм адаптивного градієнтного спуску (англ. adaptive gradient algorithm, AdaGrad) [15] досягає цієї корекції шляхом масштабування вектору градієнта вздовж найкрутіших розмірів (див. рівняння (4.4.1)).

$$\begin{cases} s_{n+1} = s_n + \nabla \mathrm{C}(w_n) \otimes \nabla \mathrm{C}(w_n) \\ w_{n+1} = w_n - h\nabla \mathrm{C}(w_n) \oslash \sqrt{s_{n+1} + \varepsilon} \end{cases} \tag{4.4.1}$$

На першому кроці накопичується квадрат градієнтів у векторі s (тут символ $\otimes$ представляє поелементне множення). Другий крок майже ідентичний градієнтному спуску, але з однією великою різницею: вектор градієнта нормується на $\sqrt{s_{n+1} + \varepsilon}$ (символ $\oslash$ представляє поелементне

ділення, а $\varepsilon -$ згладжувальна константа, щоб уникати ділення на нуль, зазвичай встановлюється на 10^{-10}).

Головною концепцією даного алгоритму є те, що він знижує швидкість навчання, але робить це суттєвіше в напрямках крутих схилів (параметрів з великими градієнтами), ніж в напрямках з більш м'якими схилами (параметрів з малими градієнтами). Швидкість навчання (крок h) в даному підході називають адаптивною. Це допомагає направити отримані оновлення більш прямо до загального оптимуму. Також додатковою перевагою є те, що налаштування гіперпараметру швидкості навчання h є менш чутливе та не потребує занадто високої уваги, як в раніше розглянутих методах.

4.5 Порівняльний аналіз методів навчання мережі. Рекомендації щодо застосування результатів аналізу

В даному підрозділі описано обчислювальний експеримент, проведений з метою порівняння різних методів навчання нейронних мереж, та надаються рекомендацій щодо застосування розглянутих методів відповідно до результатів експерименту. Суть експерименту – порівняти швидкість та якість оптимізації нейронної мережі заданої архітектури, застосовуючи різні методи оптимізації її навчання.

Експеримент було реалізовано за допомогою мови програмування Python [16] та додатково встановлених сторонніх бібліотек NumPy [17], Pandas [18], MatplotLib [19], Keras [20], Tensorflow [21]. Середовищем розробки був Jupyter Notebook, який дозволяє в інтерактивному режимі виконувати програму по блокам коду та одразу ж аналізувати отримані проміжні результати.

Для навчання нейронної мережі була узята тренувальна вибірка під назвою CIFAR-10 [22], яка складається з 60000 кольорових зображень розміром 32 на 32 у 10 класах, по 6000 зображень на клас. Окрім 50000 навчальних зображень, вибірка має 10000 тестових зображень. Приклади зображень з вибірки зображено на рисунку 4.5.1.

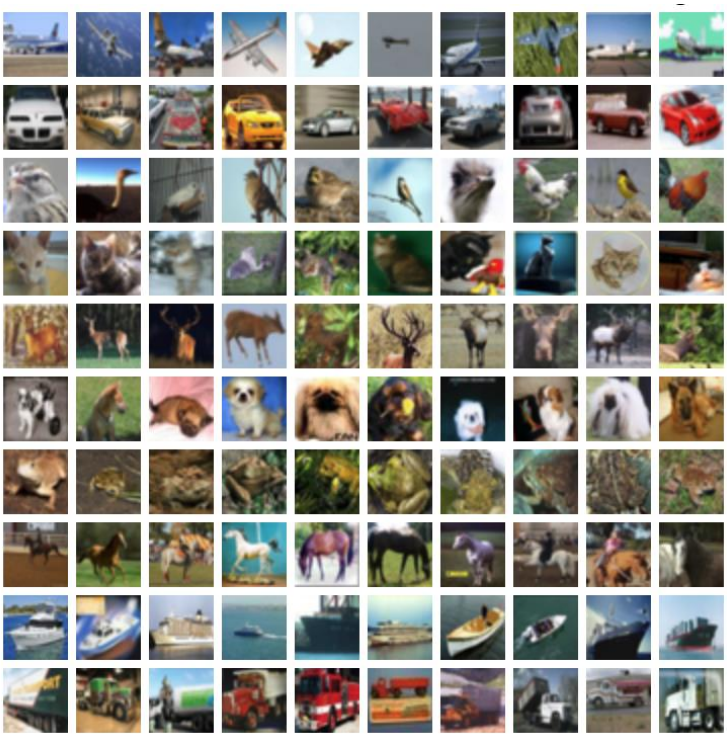

Рисунок 4.5.1 – Зображення з тренувальної вибірки CIFAR-10

Класи повністю взаємовиключні. Наприклад, між легковими та вантажними автомобілями немає перекриття. Клас "автомобіль" включає седани, позашляховики тощо. Клас "вантажівка" включає лише великі вантажівки.

Для проведення експерименту з навчання нейронної мережі за допомогою різних методів оптимізації була обрана наступна її архітектура:

– вхідний шар із 1024 нейронів

– 5 прихованих шарів, кожен з яких містить по 100 нейронів;

– ReLU функція активації в прихованих шарах;

– вихідний шар з 10 нейронів;

– softmax функція активації у вихідному шарі;

– функція втрат перехресна ентропія.

Тут варто навести пояснення з приводу вибору таких гіперпараметрів. Нейронна мережа має 1024 нейрони на вході оскільки це детермінується розмірністю вхідних зображень. В даному підході, на відміну від згорткових нейронних мереж, інформація про структуру зображення не зберігається, матриця розмірності 32 на 32 перетворюється на вектор розмірності 1024.

Кількість шарів та нейронів в них були обрані шляхом попереднього тестування якості апроксимації нейронної мережі на різних комбінаціях. Відповідно були обрані найкращі гіперпараметри.

Було обрано функцію активації ReLU для прихованих шарів через необхідність швидшого навчання, оскільки об’єм набору тренувальних даних та кількість параметрів мережі за визначеною архітектурою потребує велику кількість обчислювальних ресурсів, а функція ReLU, як раніше зазначалось, обчислюється дуже швидко порівняно з іншими альтернативами.

Вихідний шар має 10 нейронів, оскільки в задачі класифікації наявно 10 класів, відповідно по одному нейрону на клас. Такий вибір також узгоджується із запропонованою функцією активації у вихідному шарі. Функція softmax нормує вихід кожного нейрона від нуля до одиниці, а також сума по всім виходам дорівнюватиме одиниці. Дані властивості забезпечують інтерпретацію вихідного сигналу як ймовірність належності відповідному класу.

З приводу перехресної ентропії як функції втрат, достатньо згадати, що вона є найрозповсюдженішим вибором для задачі мультикласової класифікації, тому було використано саме її.

За визначеною архітектурою нейронну мережу було навчено за допомогою градієнтного спуску (1.2.2), стохастичного градієнтного спуску

(2.1.1), градієнтного спуску з імпульсом (4.2.1), методом прискорення градієнту Нестерова (4.3.1) та адаптивним градієнтним спуском (4.4.1). Результати навчання наведено в таблиці 4.5.1.

Метод оптимізації	Швидкість збіжності	Швидкість навчання	Досягнутий оптимум
GD	41	1293 с.	0.01257
SGD	83	121 с.	0.01261
Momentum	76	104 с.	0.01292
Nesterov	72	95 с.	0.01297
AdaGrad	60	83 с.	0.01342

Таблиця 4.5.1 – Результати обчислювального експерименту

Отже, отримані значення добре узгоджуються із теоретичним підґрунтям кожного з методів. Класичний градієнтний спуск потребував найменшу кількість ітерацій до зупинки, має найкращий показник оптимуму цільової функції, проте швидкість навчання за його застосування суттєво нижча за інші альтернативи.

Можливо, найкращим балансом між швидкістю навчання та точністю апроксимації буде застосування стохастичного градієнтного спуску, оскільки досягнутий оптимум за його використання мінімально відрізняється від оптимуму за класичним градієнтним спуском, а швидкість навчання можна вважати задовільною.

Щодо показників останніх трьох методів у таблиці 4.5.1, то можна бачити, що кожен наступний метод має трішки більшу швидкість збіжності разом із швидкістю навчання за попередній, але точність апроксимації знижується.

Звернемо також увагу на значне погіршення точності апроксимації в методі AdaGrad (4.4.1). Причиною цього є те, що AdaGrad зазвичай добре працює для простих квадратичних задач, але часто зупиняється занадто рано при навчанні нейронних мереж. Адаптивна швидкість навчання настільки зменшується, що в кінцевому підсумку алгоритм повністю зупиняється до досягнення глобального оптимуму. Отже, його використання для тренування глибинних нейронних мереж не рекомендоване, проте він може бути ефективним для більш простих задач, таких як лінійна регресія.

Підсумовуючи результати експерименту, не рекомендується застосовувати класичний градієнтний спуск та метод адаптивного градієнту для навчання глибинних нейронних мереж. Доцільним є використання стохастичного градієнтного спуску, методу оптимізації імпульсу чи методу прискорення градієнту Нестерова. Вибір серед останніх залежить від необхідного балансу між швидкістю навчання та точності апроксимації.

ВИСНОВКИ

Результати даної роботи є важливими завдяки актуальності проблематики навчання нейронних мереж. Застосування саме розглянутих методів є досить розповсюдженим на практиці, тому отримані висновки відповідають сучасному рівню наукових і технічних знань і технологій.

В підсумку даної роботи можна відзначити ефективність застосування розглянутих методів оптимізації для навчання нейронних мереж. Висновки з результатів обчислювального експерименту, які добре узгоджуються із теоретичними відомостями, можуть бути корисні при виборі стратегії навчання та оптимізації нейронних мереж, особливо у випадках глибинних мереж, що часто призводить до тривалого процесу оптимізації. Частіше така проблема виникає у галузях обробки зображень, а саме їх розпізнавання, сегментації тощо.

Отримані результати є важливими у прикладному аспекті, оскільки їх застосування можливе у сфері бізнесу, безпеки, розваг, комп’ютерних ігор тощо. Ці сфери все більше впроваджують технології штучного інтелекту у свої процеси, тому соціально-економічний наслідок такої інтеграції є очевидним.

Окрім розглянутих підходів існує багато напрямків для подальших досліджень в даній галузі, зокрема у задачах оптимізації швидкості навчання нейронної мережі. Серед таких напрямків є дослідження методу середньоквадратичного розповсюдження градієнту, методу адаптивної оцінки прискорення тощо. Теоретичні відомості підтверджують те, що їх застосування для навчання нейронної мережі призводить до ще більш швидшого процесу навчання, ніж при застосуванні розглянутих методів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bachman D. Advanced Calculus Demystified / David Bachman. – New York:McGraw-Hill Professional, 2007. – 274 p. – URL: https://www.twirpx.com/file/324646/. – ISBN 978-0-07-148121-2 (accessed: 14.03.2021)
2. Learning Rate in Machine learning. – URL: https://www.educative.io/edpresso/learning-rate-in-machine-learning (accessed: 14.03.2021)
3. Avriel M. Nonlinear Programming: Analysis and Methods / Mordecai Avriel. – Mineola, New York: Dover Publishing INC, 2003. – 497 p. – URL: https://books.google.com.ua/books?hl=en&lr=&id=byF4Xb1QbvMC&oi=fnd&pg=PA1&ots=zLpFDoUS9p&sig=i_DasKQquCwcpMgkXBUE8pb_FA&redir_esc=y#v=onepage&q&f=false (accessed: 12.03.2021)
4. Kumar N. Global and Local Minima in Gradient Descent in Deep Learning / Naresh Kumar. – URL: http://theprofessionalspoint.blogspot.com/2019/06/ (accessed: 03.03.2021)
5. Bottou L. Stochastic Gradient Learning in Neural Networks / Leon Bottou. – URL: https://leon.bottou.org/publications/pdf/nimes-1991.pdf (accessed: 22.03.2021)
6. Isakov S. Как работает нейронная сеть: алгоритмы, обучение, функции активации и потери: [Электронный ресурс] / Stanislav Isakov. – 25.07.2016. – Режим доступа: https://neurohive.io/ru/osnovy-data-science/osnovy-nejronnyh-setej-algoritmy-obuchenie-funkcii-aktivacii-i-poteri/ (дата звернення: 01.03.2021)
7. Yun S., Jeong M., Kim R., Kang J., Kim H.J. Graph Transformer Networks / Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, Hyunwoo J.

Kim // Neural Information Processing Systems. – 2019. – URL: https://papers.nips.cc/paper/2019/file/9d63484abb477c97640154d40595a3bb-Paper.pdf (accessed: 23.02.2021)

8. Bhadeshia H. K. D. H. Neural Networks in Materials Science / H. K. D. H. Bhadeshia // ISIJ International. – 1999. – Vol. 39. No. 10. – Pp. 966–979. – URL: https://www.phase-trans.msm.cam.ac.uk/abstracts/neural.review.pdf DOI: https://doi.org/10.2355/isijinternational.39.966
9. Cybenko G. V. Approximation by Superpositions of a Sigmoidal function / George V. Cybenko // Mathematics of Control Signals and Systems. – 1989. – Vol. 2. – No. 4. – Pp. 303–314. – DOI: 10.1007/BF02551274
10. Payne J. Activation Functions in Artificial Neural Networks / Joshua Payne. – Jan 4, 2020. – URL: https://medium.com/swlh/activation-functions-in-artificial-neural-networks-8aa6a5ddf832 (accessed: 21.02.2021)
11. Милютин И. Функции активации нейросети: сигмоида, линейная, ступенчатая, ReLu, tahn: [Электронный ресурс] / И. Милютин. – 29.11.2018. – Режим доступа: https://neurohive.io/ru/osnovy-data-science/activation-functions/ (дата звернення 21.02.2021)
12. Акулич И. Л. Математическое программирование в примерах и задачах / И. Л. Акулич. – М.: Высшая школа, 1986. – С. 298–310.
13. Polyak B. T. Some Methods of Speeding Up the Convergence of Iteration Methods / Boris Teodorovich Polyak // USSR Computational Mathematics and Mathematical Physics. – 1964. – Vol. 4. – No. 5. – Pp. 1–17.
14. Nesterov Yu. A Method for Unconstrained Convex Minimization Problem with the Rate of Convergence O(1/k2): [Doklady AN USSR 269] / Yurii Nesterov. – 1983. – Pp. 543–547.
15. Duchi J., Hazan E., Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization / John Duchi, Elad Hazan, Yoram

Singer // Journal of Machine Learning Research. – 2011. – Vol. 12 (61). – Pp. 2121–2159.

16. Python. – URL: https://www.python.org (accessed 11.03.2021)
17. NumPy. – URL: https://numpy.org (accessed 11.03.2021)
18. Pandas. – URL: https://pandas.pydata.org (accessed 11.03.2021)
19. Matplotlib: Visualization with Python. – URL: https://matplotlib.org (accessed 11.03.2021)
20. Keras. – URL: https://keras.io (accessed 11.03.2021)
21. TensorFlow. – URL: https://www.tensorflow.org (accessed 11.03.2021)
22. The CIFAR-10 dataset. – URL: https://www.cs.toronto.edu/~kriz/cifar.html (accessed 11.03.2021)