МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інтелектуальна технологія виявлення шахрайських дій в системах електронної комерції»
здобувачки групи ІН.м – 22 Піддубної Дарини Іванівни

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Дарина ПІДДУБНА
(підпис)

Керівник,
в.о. завідувача кафедри,
кандидат технічних наук, доцент
Ігор ШЕЛЕХОВ

(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-22 Піддубної Дарини Іванівни

1. Тема роботи: «Інтелектуальна технологія виявлення шахрайських дій в системах електронної комерції.»
затверджую наказом по СумДУ від *«01» грудня 2023 року № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 18 грудня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розпізнавання шахрійства в системах електронної комерції . 3) Розробка інтелектуальної технології з розпізнавання шахрайських дій в системах електронної комерції. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.
Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для розпізнавання шахрійства в системах електронної комерції*		
3	*Розробка інтелектуальної технології з розпізнавання шахрайських дій в системах електронної комерції*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти Керівник
(підпис) (підпис)

АНОТАЦІЯ

Записка: 44 стр., 5 рис., 2 додаток, 17 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі розпізнавання шахрайських дій в системах електронної комерції шляхом розробки відповідних методів, моделей та інформаційної технології.

Об’єкт дослідження — процес розпізнавання шахрайських дій.

Мета роботи — розробка інформаційної технології розпізнавання шахрайських дій з використанням побудови графіків на основі математичної моделі.

Методи дослідження — алгоритми прийняття рішень і прогнозування подій та інструменти побудови математичних моделей.

Результати — розроблено інформаційну технологію, яка за допомогою машинного навчання розпізнає шахрайські дії в системах електронної комерції .

ІНТЕЛЕКТУАЛЬНА СИСТЕМА, РОЗПІЗНАВАННЯ ШАХРАЙСЬКИХ ДІЙ, PYTHON, TENSERFLOW.

ЗМІСТ

ВСТУП 5
1 АНАЛІТИЧНИЙ ОГЛЯД 7
1.1 Сучасний стан нейронних мереж 7
1.2 Аналіз дотичних систем 7
1.3 Постановка задачі 11
2 МАТЕМАТИЧНИЙ МЕТОД РОЗВ’ЯЗАННЯ ЗАДАЧІ 13
2.1 Пакетне навчання та онлайн-навчання 14
2.2 Алгоритм зворотного розповсюдження 15
2.3 Передачі вперед і назад 18
2.4 Чисельний приклад зворотного розповсюдження 19
2.5 L2 Регуляризація 23
3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ 24
3.1 Модель програмування TensorFlow 24
3.2 Реалізація Scikit-learn 30
3.3 Реалізація TensorFlow 35
ВИСНОВКИ 37
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 38
ДОДАТОК А 40
ДОДАТОК Б 42

ВСТУП

Обґрунтування вибору теми роботи.

Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі розпізнавання шахрайських дій в системах електронної комерції шляхом розробки відповідних методів, моделей та інформаційної технології

Актуальність. Онлайн покупки стали надзвичайно популярними, зокрема завдяки зростанню доступу до Інтернету та зручності шопінгу з будь-якого пристрою. Однак, разом зі зростанням електронної комерції, збільшується ймовірність шахрайства та шахрайських дій, які можуть негативно впливати на якість та безпеку онлайн покупок.

Об’єкт дослідження. Машинне навчання - це одна з підгалузей штучного інтелекту, яка представляє людський інтелект за допомогою машин. Глибоке навчання як одна з підгалузей машинного навчання стає найпопулярнішим напрямком досліджень в даний час. Воно використовує штучну нейронну мережу (ШНМ), яка є машиною для обробки інформації, змодельованою на основі структури та дії біологічної нейронної мережі в мозку [6][5]. ШНМ є гнучким і самоадаптивним для вирішення складних проблем, які важко описати за допомогою математичної моделі, таких як розпізнавання і класифікація образів, апроксимація функцій і керування [7]. Останніми роками зростає інтерес до глибоких нейронних (ШНМ), які використовують багато шарів, підвищило потребу в ШНМ як у промислових, так і в академічних галузях. Глибокі нейронні мережі навчаються на досвіді даних, щоб апроксимувати будь-які нелінійні зв'язки між вхідною інформацією та кінцевим результатом. Добре навчена глибока нейронна мережа має здатність вловлювати абстрактні особливості у всьому наборі даних.

Предмет дослідження. Метою цієї роботи є виявлення шахрайських операцій з кредитними картками за допомогою застосуванням нейронних мереж. Застосовуючи алгоритм зворотного розповсюдження для пошуку

оптимальних параметрів, мережа навчається для досягнення стабільності та оптимальності, щоб можна було знайти відповідну модель для виявлення того, чи є здійснена транзакція є нормальною чи шахрайською. Цю проблему виявлення шахрайських транзакцій можна розглядати як проблему класифікації.

Гіпотеза.

Новизна. У зв'язку з відсутністю маркованого набору даних та значною кількістю часу, необхідного для створення такого набору даних, метою цієї магістерської роботи є дослідження та розробка прототипу виявлення шахрайства на основі алгоритмів машинного навчання. Використовуючи різні методи, прототип повинен виявляти нові типи спроби шахрайства, якщо такі спроби існують.

Структура. Дане робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан нейронних мереж

Механізм роботи штучного нейрона змодельований на нейроні в мозку[1]. Нейрон є основним елементом для обробки інформації в мозку. Мозок складається з близько 10 мільярдів нейронів, які з'єднані між собою, утворюючи мережу. Біологічний нейрон отримує сигнали від інших нейронів і обробляє інформацію[3][4]. Якщо оброблений вхідний сигнал перевищує порогове значення, то нейрон спрацьовує і виробляє електричний імпульс для передачі сигналів іншим нейронам; якщо вхідний сигнал нижче порогового значення, нейрон не спрацьовує і вихідний сигнал не виробляється[2].

1.2 Аналіз дотичних систем

Штучний нейрон

Топологія штучної нейронної мережі визначається взаємозв'язком між основними одиницями процесу, які називаються нейронами. Штучний нейрон - це основна обчислювальна одиниця, яка виконує нелінійну функцію над вхідним сигналом. На нейрон подається вхідний сигнал $x_1, x_2, \dots x_m$ через набір змінних, які можуть або посилювати, або послаблювати певний вхідний сигнал. Сигнал x_j на вході ланки j, що з'єднує нейрон k, множиться на вагу ω_{jk} . Добуток вагових коефіцієнтів підсумовуються як чистий вхід функції активації, яка виробляє вихід, який обмежено деяким скінченним значенням. Крім того, додається зміщення позначене через b_k, також додається до нейрона як компонент чистого входу функції активації. Зсув відіграє важливу роль при проектуванні мережі. Воно дозволяє паралельний рух функції активації, що збільшує можливість розв'язання задач.

Математичний вираз для моделі нейрона k має вигляд

$$u_k = \sum_{j=1}^{m} \omega_{kj} x_j \tag{1.1}$$

Та

$$y_k = \varphi(u_k + b_k) \tag{1.2}$$

де $\omega_{k1}, \omega_{k2}, \dots, \omega_{km}$ – пов'язана вага нейронів k; u_k - сума добутків вхідних ваг; b_k – зміщення; $\varphi(\cdot)$ – функія активації; y_k - вихідний сигнал нейрона k.

Зміщення b_k також можна розглядати як змінну вагу, що дорівнює ω_{k0} при фіксованому значенні $x_0 = +1$. Таким чином, рівняння (1.1) можна записати наступним чином:

$$v_k = \sum_{j=0}^{m} \omega_{kj} x_j \tag{1.3}$$

тоді

$$y_k = \varphi(v_k) \tag{1.4}$$

Функція активації

Функція активації виконує нелінійне перетворення вхідного сигналу з метою управління активацією на виході, отже, нескінченний діапазон вхідного сигналу може бути переведений у певний діапазон вихідного значення[8]. Декілька найпоширеніших функцій активації наведено нижче:

1. Порогова функція, що описує вихід нейрона k задається формулою

$$y_k = \begin{cases} 1, \text{якщо } v_k \geq 0; \\ 0, \text{якщо } v_k < 0. \end{cases}$$

де v_k - чистий вхід нейрона k:

$$v_k = \sum_{j=1}^{m} \omega_{kj} x_j + b_k.$$

2. Сигмоїдна функція - це загальновживана функція активації функція, яка має "S"-подібну форму. Це строго монотонна зростаюча функція яка демонструє хороший баланс між лінійними та нелінійними властивостями[1]. Одним з прикладів сигмоїдної функції є логістична функція, яка визначається через

$$f(v) = \frac{1}{1+e^{-av}} \tag{1.5}$$

де a - параметр нахилу сигмоїдної функції, який дорівнює $\frac{4}{a}$ у точці початку координат. Логістична сигмоїдна функція обмежує значення виходу неперервним діапазоном від 0 до 1, що представляє ймовірність бінарної класифікації. Вона дорівнює прийнятне математичне представлення моделі біологічного нейрона, яке

показує, чи вистрілить нейрон чи ні[8]. З наближенням до нескінченності сигмоїдна функція стає пороговою функцією. Більше того, сигмоїдна функція є диференційованою, що є важливими властивостями для теорії навчання нейронної мережі[1]. Одним недоліком сигмоїдної функції є її насичення, що знищує градієнт. Коли на виході на хвості 0 або 1 (насичення), градієнт нейрона наближається до нуля, що робить мережу важкою для навчання.

3. Гіперболічний тангенс більше схожий на логістичну сигмоїдну функцію, за винятком того, що він дає вихід від -1 до 1.Вона є обмеженою та диференційованою. Математичне визначення гіперболічної тангенса задається формулою

$$f(v) = tg(v) = \frac{e^{v}-e^{-v}}{e^{v}+e^{-v}} = \frac{e^{2v}-1}{e^{2v}+1} \quad (1.6)$$

4. Випрямлені лінійні одиниці [9] - це відносно нова функція, яка використовується для навчання глибоких нейронних мереж в останні роки[10]. Випрямляч є лінійним, коли вхідний сигнал позитивний, і нульовим в іншому випадку, що визначається як

$$f(v) = \max(0, v)$$

де v - вхід нейрона. Порівняно з сигмоїдною функцією активації, ця функція розв'язує задачу навчання швидше завдяки своїй ненасичуваній нелінійності. Крім того, її можна просто реалізувати завдяки порогу матриці активацій, який дорівнює нулю, на відміну від сигмоїдної функції, яка спричиняє дорогі обчислення.

Іншим типом є функція softplus, плавне наближення до випрямляча, що визначається

$$f(v) = \ln(1 + e^{v})$$

Одношарова мережа прямого зв'язку

За топологічною структурою зв'язків між нейронами нейронну мережу можна розділити на різні типи. Один з них - це мережа прямого поширення в якій дані, що надходять з вхідного шару, передаються в одному напрямку до вихідного шару і немає циклу між нейронами. Одношарова нейронна мережа

є найпростішою і складається з одного вихідного шару, тобто дані з вхідного шару надходять безпосередньо до вихідного за допомогою вагових коефіцієнтів. На рисунку 1.1 показано однорівневу мережу прямого зв'язку

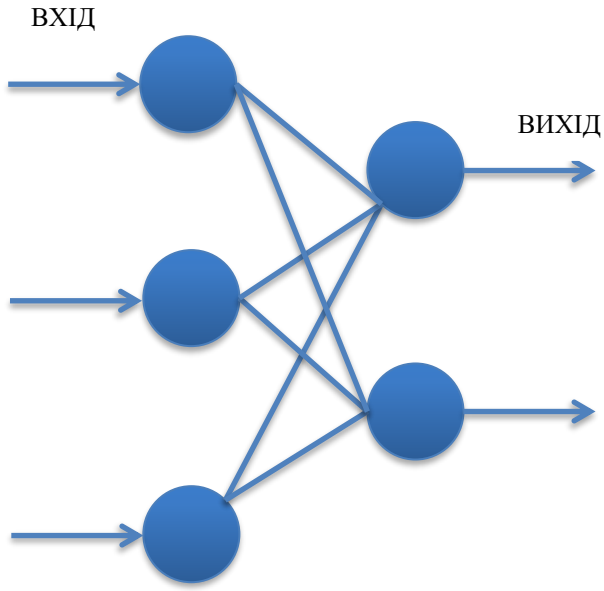


Рисунок 1.1 – Одношарова мережа прямого зв'язку

Персептрон

Персептрон, запропонований Розенблаттом (1958), є найпростішою нейронною мережею для лінійно відокремлюваної класифікації. Персептрон - це один нейрон з регульованими вагами та зміщенням. Розенблат довів, що алгоритм навчання персептрона може бути збіжним, щоб знайти гіперплощину як межу поділу розділити навчальні приклади, які можна класифікувати на два лінійно відокремлювані класи[1]. Цей алгоритм відомий як теорема збіжності персептрона. Включаючи більше одного нейрона у вихідний шар персептрона, мультиперсептрон може класифікувати більше класів, які є лінійно відокремлюваними.

Нейрон з'єднаний з набором входів відповідними регульованими вагами. Сукупність добутків вагових добутків входів використовується як вхід порогової функції активації. Введене зміщення додає ще один вільний параметр, щоб зробити вихід мережі більш досягаємим очікуваної мети. Вихід персептрона класифікується відповідно до чистого входу. Якщо чистий вхід

$v_k = \sum_{j=0}^{m} \omega_i x_i + b$ є додатним, то нейрон видає вихід $+1$, і -1, якщо вхід від'ємний. Таким чином, рішення класифікації може бути представлено гіперплощиною, яка визначається

$$\sum_{i=1}^{m} \omega_i x_i + b = 0 \quad (1.7)$$

коли параметри мережі визначені, то траєкторію руху $\sum_{i=1}^{m} \omega_i x_i + b = 0$ можна намалювати як межу класифікації у просторі, що складається з вхідних векторів $x_1, x_2, \dots, x_m$. Для будь-якого заданого вхідного вектора через вагу та зміщення, він знаходиться або вище, або нижче гіперплощини. Щоб отримати бажану класифікацію вхідних векторів, оптимальні вага та переміщення персептрона можуть бути визначені шляхом ітеративного навчання на наборі даних. Як згадувалося вище, Роузблатт довів, що якщо навчальні приклади вибираються з двох лінійно відокремлюваних класів, то алгоритм оновлення ваг є збіжним. Позначимо часовий крок виконання алгоритму через n. Вага персептрона може бути скоригована згідно з правилом корекції помилок, яке визначається формулою

$$w(n+1) = w(n) + \eta[d(n) - y(n)]x(n) \quad (1.8)$$

де η - швидкість навчання, яка є фіксованим параметром між 0 та 1, $d(n)$ бажане значення, а $y(n)$ - фактичне значення. Один персептрон може лише функціонувати на простій класифікації образів для двох класів, що повністю перетинаються гіперплощиною [6]. Нелінійно відокремлювана класифікація образів знаходиться за межами обчислювальних можливостей персептрона.

1.3 Постановка задачі

Через практичну обмеженість одношарової мережі на лінійній сепарабельній задачі, була введена глибока нейронна мережа для розв'язання задачі довільної класифікації[1]. Вона містить один або більше прихованих шарів, обчислювальні вузли яких називаються прихованими вузлами. Глибина моделі відноситься до кількості прихованих шарів. Вхідна інформація надходить до першого прихованого шару, а виходи цього шару передаються як вхідні дані до другого прихованого шару і так далі. Кожен шар отримує

виходи попереднього шару як входи, таким чином, вхідний сигнал поширюється вперед по шару за шаром, поки не досягне вихідного шару. На нейронах вихідного шару виробляється сигнал помилки, який поширюється назад по мережі.

Кожен нейрон у прихованому та вихідному шарах виконує дві операції. Одна з них полягає в тому, щоб обчислити диференційовану функцію активації на вхідних векторах і вагах, і поширює вихід вперед через приховані шари. Друга операція полягає в обчисленні градієнта помилки щодо ваг, з'єднаних з входами цього нейрона, який протікає назад через мережу.

Глибокі нейронні мережі можна використовувати для розпізнавання образів і класифікації. Вхідний шар складається з ознак вектора, що підлягають класифікації. Кожен прихований шар нейронів обробляє свій власний набір ознак, які є виходом попереднього шару. Чим більше прихованих шарів має мережа, тим складніші ознаки можуть бути виявлені нейронами, оскільки вони збирають і комбінують інформацію, згенеровану попередніми шарами. Нелінійність від навчальних даних дає мережі більшу обчислювальну потужність і може реалізовувати більш складні функції, ніж мережа без прихованих нейронів.

2 МАТЕМАТИЧНИЙ МЕТОД РОЗВ'ЯЗАННЯ ЗАДАЧІ

У цьому розділі представлено оцінювання глибокої нейронної мережі на основі методу Саймона Хейкіна (2009). Процес навчання нейронної мережі полягає в оцінці оптимальних вагових параметрів шляхом навчання на наборі даних для отримання очікуваного результату. Навчання починається з ініціалізації випадково малими вагами. Для того, щоб забезпечити нормальне навчання мережі нормально навчатися, ваги повинні бути ініціалізовані різними значеннями. Вибір великих ваг призведе до перенасичення мережі. Загалом, ефективним способом ініціалізації ваг є випадковий вибір значень з рівномірного розподілу. Навчання здійснюється шляхом подачі вхідних даних у мережу для отримання вихідних даних. Продуктивність нейронної мережі вимірюється різницею між фактичним виходом і бажаним виходом. Ця різниця описується як функція втрат, яку можна мінімізувати за допомогою вагових коефіцієнтів.

Коли мережа навчається на наборі даних, мітка категорії якого відома, вона використовує кероване навчання. Кожен навчальний приклад у навчальному наборі має пару даних, вхідний вектор і відповідний очікуваний вихід. Мережа вивчає функцію, використовуючи маркований навчальний набір, тому вона може зіставити немаркований вхід у правильний вихід для виконання класифікації або регресії[9][10].

Розглянемо навчальну множину, яку позначимо як $D = \{(x^1, d^1), \dots, (x^N, d^N)\}$, вона використовується для навчання мережі з одним або декількома прихованими шарами. Позначимо вихід нейрона j через y_j у вихідному шарі, отриманий на основі даних $x(n)$ у вхідному шарі, відповідну функція втрат на виході нейрона j обчислюємо за формулою

$$e_j(n) = d_j(n) - y_j(n) \tag{2.1}$$

Для зручності знаходження похідної функції втрат додамо коефіцієнт $\frac{1}{2}$ до e, і підсумовуємо втрати всіх нейронів вихідного шару, тоді сумарні втрати всієї мережі виражається через

$$E_n(w) = \frac{1}{2}\sum_{j\in C} e_j^2\,(n) \tag{2.2}$$

де множина C містить всі нейрони вихідного шару. З навчальною множиною що складається з N прикладів, середня втрата на всій навчальній множині визначається за формулою

$$E_{av}(w) = \frac{1}{N}\sum_{n=1}^{N} E(w) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C} e_j^2(n) \tag{2.3}$$

Важливим моментом є те, що E є функцією всіх ваг, що модифікуються, тобто вільних параметрів мережі.

2.1 Пакетне навчання та онлайн-навчання

Існує два різних методи реалізації керованого навчання нейронної мережі, а саме: пакетне навчання та навчання в режимі реального часу[1]. Пакетне навчання, яке також називають стандартним градієнтним спуском, виконує коригування вагового простору після представлення всіх прикладів у навчальній вибірці, що створює одну епоху навчання. Іншими словами, функція втрат при пакетному навчанні задається $E_{av}(w)$, а градієнт обчислюється для всієї навчальної вибірки. В онлайн методі навчання кожен приклад обирається випадковим чином з навчальної вибірки, а крок коригування у ваговому просторі для кожного прикладу, що подається мережі. Таким чином, функція втрат, яку потрібно мінімізувати, є $E(w)$.

На практиці пакетне навчання призводить до додаткових обчислень для більшого набору даних оскільки з'являються схожі приклади для обчислення градієнтів[14]. Тому було запропоновано стохастичний градієнтний спуск для обчислення градієнта для кожної підмножини всієї навчальної вибірки. На кожному кроці ми беремо M випадкових вибірок з N навчальних прикладів для обчислення градієнта, а потім оновлюємо ваги. Таким чином, функція втрат, яку потрібно оптимізувати, стає:

$$E_{av}(w) = \frac{1}{2M}\sum_{m=1}^{M}\sum_{j\in C} e_j^2(m) \tag{2.4}$$

де M - ціле число від 1 до N. Стохастичний градієнтний спуск є найпоширенішим методом оптимізації, який призводить до більш швидкої

процедури навчання, а також може використовуватися для навчання в режимі онлайн[14].

2.2 Алгоритм зворотного розповсюдження

Алгоритм зворотного розповсюдження є одним з найпростіших і найпоширеніших методів навчання керованих нейронних мереж при обчисленні градієнта функції втрат. Градієнт - це нахил функції втрат у просторі ваги, який показує, як функція втрат змінюється при зміні ваги. Функція втрат (помилки) вироблена поточною вагою та поширюється назад через мережу для оновлення ваги. Оскільки мережа вчиться на своїх помилках на кожній навчальній ітерації, вона ітеративно коригує вагу, щоб зменшити функцію втрат.

Спочатку починаємо пошук на поверхні втрат з початковою вагою, потім рухаємося з кроком у напрямку, протилежному градієнту, причому розмір кроку визначається як швидкістю навчання, так і нахилом градієнта[14]. Математиче правило оновлення ваг визначається формулою:

$$w = w - \eta \nabla E_{av}(w) \tag{2.5}$$

де η - швидкість навчання, фіксований параметр між 0 і 1. Чим менша швидкість навчання, тим менші зміни у ваговому просторі і довший час навчання для досягнення мінімуму функції втрат. Більша швидкість навчання, призводить до нестабільного результату. Градієнт $\nabla E = (\frac{\partial E}{\partial \omega_0}, \dots, \frac{\partial E}{\partial \omega_n})$ обчислюється за допомогою алгоритма зворотного розповсюдження.

Розглянемо нейрон j, на який подається набір входів $y_1(n), y_2(n), \dots, y_m(n)$, та мережа вхідної функції активації, що виробляється на нейроні j, він має вигляд

$$v_j = \sum_{i=1}^{m} \omega_{ji}(n) y_i(n) \tag{2.6}$$

де ω_{j0}, що відповідає фіксованому входу $y_0 = +1$, та дорівнює змішенню b_j, підключеному до нейрона j. Тому вихід функції активації f, що виробляється на нейроні j має вигляд

$$y_j(n) = f_j(v_j(n)) \tag{2.7}$$

За ланцюговим правилом представимо цей градієнт у вигляді

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \omega_{ji}(n)} = -e_j(n) f_j'\left(v_j(n)\right) y_j(n) \tag{2.8}$$

де ми множимо похідні втрат всієї мережі $E(n) = \frac{1}{2} e_j^2$, на сигнал втрат $e_j(n) = d_j(n)$, активна функція $y_j(n)$ та чистий вхід $v_j(n)$.

Поправка $\Delta\omega_{ji}(n)$до $\omega_{ji}(n)$ визначається за правилом дельти:

$$\Delta\omega_{ji}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{ji}(n)} \tag{2.9}$$

де знову ж таки η представляє швидкість навчання, а знак мінус показує, що вона змінюється у напрямку зменшення втрат. Підставимо рівняння (2.8) у рівняння (2.9) і отримуємо

$$\Delta\omega_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{2.10}$$

де локальний градієнт $\delta_j(n)$ задається формулою

$$\delta_j(n) = e_j(n) f_j'\left(v_j(n)\right) \tag{2.11}$$

Для обчислення вагового коригування $\Delta\omega_{ji}(n)$ потрібен сигнал похибки $e_j(n)$ на виході нейрона j. Існує два різних випадки, які залежать від того, чи знаходиться нейрон j у вихідному шарі чи у прихованому шарі

Випадок 1. Коли нейрон j є вихідним вузлом, похибка $e_j(n)$ нейрона j, обчислена за формулою (2.1), дає явне обчислення локального градієнта $\delta_j(n)$ за формулою (2.11).

Випадок 2. Коли нейрон j знаходиться у прихованому шарі, тут локальний градієнт для виходу на нейрон j визначається

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)} f_j'\left(v_j(n)\right) \tag{2.12}$$

де у другому рядку використовується рівняння (2.7). Сигнал помилки для прихованого нейрона j робить внесок у помилки всіх нейронів, з'єднаних з j на наступному шарі, таким чином втрати визначаються $E(n) = \frac{1}{2} \sum_{k \in C} e_k^2$, де k - вихідний нейрон. Диференціювання функції втрат за $y_j(n)$ дає

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \quad (2.13)$$

Похибка на нейроні k дорівнює

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - f_k(v_k(n)) \quad (2.14)$$

Таким чином,

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -f_k{}'(v_k(n)) \quad (2.15)$$

Чистий вхід на нейроні k має вигляд

$$v_k(n) \sum_{i=0}^{m} \omega_{kj}(n) y_j(n) \quad (2.16)$$

де m - кількість входів за виключенням зміщення, що надходить на нейрон k і $\omega_{k0} = \delta_k$ з фіксованим входом значення +1. Таким чином

$$\frac{\partial v_k(n)}{\partial y_j(n)} = \omega_{kj}(n) \quad (2.17)$$

Використовуючи рівняння (23), (25) в рівнянні (21), отримаємо

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k e_k(n) f_k'\big(v_k(n)\big)\omega_{kj}(n) = -\sum_k \delta_k(n)\omega_{kj}(n) \quad (2.18)$$

Підставивши рівняння (26) у рівняння (20), отримаємо формулу зворотного поширення для локального градієнта $\delta_j(n)$ на прихованому нейроні j

$$\delta_j(n) = f_j'\left(v_j(n)\right)\sum_k \delta_k(n)\omega_{kj}(n) \quad (2.19)$$

Щоб підсумувати співвідношення, отримані з алгоритму зворотного розповсюдження, спочатку зазначимо, що поправка на вагу $\Delta\omega_{kj}(n)$визначається за правилом дельти:

$$\Delta\omega_{kj}(n) = \eta\delta_j(n)y_j(n) \quad (2.20)$$

Локальний градієнт $\delta_j(n)$ відрізняється залежно від того, чи знаходиться нейрон j у вихідному чи прихованому шарі.

Якщо нейрон j є вихідним вузлом, локальний градієнт обчислюється за формулою $\delta_j(n) = e_j(n)f_j'\left(v_j(n)\right)$;

якщо нейрон j є прихованим вузлом, то $\delta_j(n) = f_j'\left(v_j(n)\right)\sum_k \delta_k(n)\omega_{kj}(n)$.

2.3 Передачі вперед і назад

Хейкін (2009) стверджував, що існує два різних способи передачі інформації при застосуванні алгоритму зворотного розповсюдження, а саме прямий і зворотний. Під час прямого проходу вхідний вектор разом з вагами потрапляє до першого прихованого шару; потім вхідні дані передаються до другого шару як вхід.Вихід для нейрона j обчислюється як

$$y_j(n) = f(\sum_{i=0}^{m} \omega_{ij}(n) y_i(n))$$

де m - кількість нейронів у попередньому шарі; f - функція активації нейрона j; $\omega_{ij}(n)$, що з'єднує нейрон i з нейроном j; y_i вхідний сигнал.

Якщо нейрон j знаходиться в першому прихованому шарі, то y_i - це і-й елемент вхідного вектора; якщо j знаходиться у вихідному шарі, то y_i - це j-нейрон у вихідному шарі. Вхідна інформація поширюється шар за шаром до вихідного шару, щоб виробити сигнал помилки для кожного нейрона вихідного шару , який обчислюється як різниця між вихідним значенням та фактичним значенням. вагові коефіцієнти є фіксованими при прямому поширенні.

Зворотний прохід починається з вихідного шару шляхом розповсюдження втрат назад через мережу і обчислення локального градієнта δ для кожного нейрона рекурсивно. Ваги будуть скориговані відповідно до дельта-правила згідно рівняння (2.18) рекурсивно при зворотному проході. Якщо нейрон j знаходиться у вихідному шарі, то локальний градієнт дорівнює добутку похідної функції активації, що відповідає нейрону j. Потім оновлення для ваги, що з'єднують нейрони вихідного шару, можна обчислити безпосередньо за формулою(2.19). Далі ми визначаємо локальні градієнти для нейронів у другому останньому шарі згідно з рівнянням (2.19), а потім оновлюємо всі ваги, що подаються на цей нейрон. Зміна ваги обчислюється рекурсивно і поширюються назад, доки не будуть оновлені вся вага в мережі. Слід зазначити, що початкова вага використовуються для виконання зворотного розповсюдження до оновлення всієї ваги у мережі.

Алгоритми навчання нейронної мережі та структура моделі, виражені в математиці, повинні бути реалізовані комп'ютерними програмами для реального слововживання. Тому серія комерційних програмних бібліотек машинного навчання з відкритим вихідним кодом для особливо глибокого навчання, яке використовує глибокі нейронні мережі з величезною кількістю прихованих шарів.

2.4 Чисельний приклад зворотного розповсюдження

Використаємо числовий приклад, який показаний на рисунку 2.1 з двома шарами, для того, щоб проілюструвати алгоритм зворотного розповсюдження. Вхідний вектор $x = [0.1, 0.5]$ з бажаним вихідним вектором $d = 1$. Сигмоїдну функцію активації тут було застосовано для прихованого та вихідного шарів. Припустимо, що швидкість навчання η дорівнює 0,1. Позначаємо $\omega_{ji}^{(l)}$ як вагу, що з'єднує нейрон i з нейроном j у шарі l.

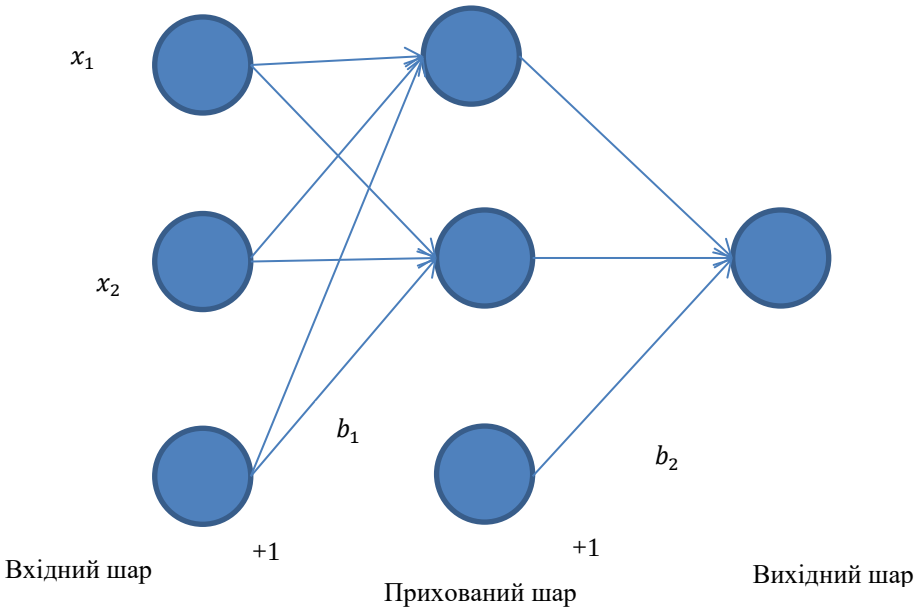


Рисунок 2.1 – Двошарове зворотне розповсюдження

Задаємо початкові значення ваги:

$w_{10}^{(1)}$	$w_{11}^{(1)}$	$w_{12}^{(1)}$	$w_{20}^{(1)}$	$w_{21}^{(1)}$	$w_{22}^{(1)}$	$w_{10}^{(2)}$	$w_{11}^{(2)}$	$w_{12}^{(2)}$
0.1	0.2	0.3	-0.1	-0.5	0.4	0.3	-0.2	0.1

Розраховуємо вхід для першого нейрона у прихованому шарі:

$$in_1^{(1)} = w_{11}^1 * x_1 + w_{12}^1 * x_2 + w_{10}^{(1)} * b_1$$

$$in_1^{(1)} = 0.2 * 0.1 + 0.3 * 0.5 + 0.1 * 1 = 0.27$$

Застосовуємо сигмоїдну функцію для отримання результату:

$$out_1^{(1)} = \frac{1}{1 + e^{-0.27}} = 0.567$$

Повторюємо кроки вище для другого нейрона в прихованому шарі, отримуємо:

$$out_2^{(1)} = \frac{1}{1 + e^{-0.5}} = 0.512$$

Отриманих нами результати прихованого шару використовуємо як вихідні дані і повторюємо процес для нейрона вихідного шару:

$$in^{(2)} = w_{11}^2 * out_1^{(1)} + w_{12}^2 * out_2^{(1)} + w_{10}^{(2)} * b_2$$

$$in^{(2)} = -0.2 * 0.567 + 0.1 * 0.512 + 0.3 * 1 = 0.2378$$

$$out^{(2)} = \frac{1}{1 + e^{-0.2378}} = 0.56$$

Похибка нейрона вихідного шару дорівнює $d - out^{(2)} = 0.44$. Нам відомо, що похідна сигмоїдної функції σ дорівнює $\sigma * (1 - \sigma)$. Градієнт для нейрона вихідного шару дорівнює

$$\delta^{(2)} = \sigma'\left(in^{(2)}\right) * \left(d - out^{(2)}\right) = \sigma(in^{(2)}) * (1 - \sigma\left(in^{(2)}\right)) * (d - \sigma\left(in^{(2)}\right))$$

$$\delta^{(2)} = 0.56(1 - 0.56)0.44 = 0.108$$

Похідна помилки нейрона у вихідному шарі має вигляд

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta^{(2)} * out_j^{(1)} ,$$

де $k = 1$.

Таким чином, за правилом дельти, оновлені ваги, що з'єднуються з нейроном вихідного шару є

$$w_{11}^{(2)} = 0.2 - (0.1)(0.108)(0.567) = -0.206$$

$$w_{12}^{(2)} = 0.1 - (0.1)(0.108)(0.512) = 0.094$$

$$w_{13}^{(2)} = 0.3 - (0.1)(0.108) = 0.289$$

Градієнт для нейронів прихованого шару задаємо формулою:

$$\delta_j^{(1)} = \sigma'\left(in_j^{(1)}\right)\sum_k w_{kj}^{(2)}\delta_k = \sigma\left(in_j^{(1)}\right)(1 - \sigma\left(in_j^{(1)}\right))]\sum_k w_{kj}^{(2)}\delta_k$$

$$\delta_2^{(1)} = (0.567)(1 - 0.567)(-0.2)(0.108) = -0.0053$$

$$\delta_2^{(1)} = (0.512)(1 - 0.512)(0.1)(0.108) = 0.0027$$

Градієнт помилки для нейронів у прихованому шарі задається формулою

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_J^{(1)} x_i.$$

Оновлені ваги з'єднань, що додаються до прихованого шару, дорівнюють

$$w_{11}^{(1)} = 0.2 - (0.1)(-0.0053)(0.1) = 0.20005$$

$$w_{12}^{(1)} = 0.3 - (0.1)(-0.0053)(0.5) = 0.30027$$

$$w_{21}^{(1)} = -0.5 - (0.1)(-0.0027(0.1) = -0.49997$$

$$w_{22}^{(1)} = 0.4 - (0.1)(-0.0027)(0.5) = 0.40014$$

$$w_{10}^{(1)} = 0.1 - (0.1)(-0.0053) = 0.10053$$

$$w_{20}^{(1)} = -0.1 - (0.1)(-0.0027) = -0.09973$$

Також зазначу, що з правила дельти, швидкість навчання відображає розмір змін у вазі. Менша швидкість навчання дає меншу зміну ваги між ітераціями, що призводить до більш плавної траєкторії руху в просторі ваги. Це, однак, збільшує час навчання для досягнення локального мінімуму на поверхні похибки. Більша швидкість навчання дає нестабільний результат роботи мережі. Щоб уникнути нестабільності зі збільшенням швидкості навчання, дельта-правило для оновлення ваги модифікується узагальненим дельта-правилом наступним чином:

$$\Delta w_{ji}(n) = \alpha\Delta w_{ji}(n-1) + \eta\delta_j(n)y_j(n) \qquad (2.21)$$

де α - додатне число, яке називається константою моменту імпульсу, що описує коригування ваги w_{ji} на ітерації n та її значення на ітерації n-1. Рівняння (2.21) можна розглянути як часовий ряд з індексом t від початкового

моменту часу до поточного моменту часу n. Розв'язавши це рівняння, отримаємо

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_t(t) y_i(t) \quad (2.22)$$

За допомогою похідних похибок $\frac{\partial E_n}{\partial w_{ij}}$ з попередніх рівнянь отримуємо

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial E(t)}{\partial w_{ij}(t)} \quad (2.23)$$

Метою навчання нейронної мережі для задачі класифікації є отримання хорошої продуктивності на невидимих даних, які мають такий самий розподіл, як і навчальні дані в тому ж наборі даних. Коли нейронна мережа вивчає навчальні дані забагато, то вона буде концентруватися на пошуку ознак, які тільки виходять у навчальних даних, а не на пошуку правжньої базової закономірності. Це призводить до поганого узагальнення, яке називається перенавчанням. Щоб покращити узагальнення та уникнути перенавчання, потрібно оптимізувати час зупинки навчання. Оскільки вага ініціалізується випадковими значеннями, то похибка на навчальній вибірці є великою. Зі збільшенням кількість навчальних ітерацій збільшується, а похибка різко зменшується, вже потім продовжує повільно зменшуватися до того, як мережа досягне необхідної мінімальної похибки для зупинки навчання. Однак, коли помилка навчання поступово зменшується до стабільного значення, помилка на невидимих даних зростає. Тому оптимальне узагальнення зазвичай з'являється до того, як помилка навчання досягає локального або глобального мінімуму. Для того, щоб знайти оптимальний час зупинки, самі дані розбиваються на окремі множини: навчальна множина, перевірочна множина, яка не використовується для навчання, та тестова множина. Після кожного періоду, наприклад, кожні три етапи, навчання проводиться з фіксованими вагами та зміщенням, потім мережа тестується на перевірочній множині. Похибка кожного прикладу в валідаційному наборі оцінюється як помилка валідації. Навчання зупиняється, коли варіант помилки валідації мінімальний.

2.5 L2 Регуляризація

Якщо нейронна мережа надмірно складна, особливо з великою кількістю параметрів, то це може призвести до перенавчання [11]. Щоб спростити мережу для кращого узагальнення, то до функції втрат(поомилок) додається член регуляризації, щоб покарати за складність[12]. Одним з широко використовуваних методів є штрафування суми квадратів ваг, тому функція втрат (помилок), додана з членом регуляризації визначається як:

$$\mathcal{L}(w) = \frac{1}{2}\sum_{i=0}^{m}(d^i - y^i)^2 + \frac{\lambda}{2m}\sum_w w^2 \qquad (2.24)$$

де w - параметри нейронної мережі, m - кількість навчальних прикладів що містяться у навчальній вибірці, а λ - параметр регуляризації для керування компромісом між двома членами функції втрат. Цей підхід до регуляризації називається $L2$ регуляризацією (спаданням ваги). Додавання параметра регуляризації може обмежити діапазон параметрів не надто великим, тому це може зменшити перенавчання до певної міри.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Модель програмування TensorFlow

Алгоритми представлені в розділах вище можуть бути реалізовані комп’ютерними програмами для реального використання. Існує декілька варіантів бібліотек машинного навчання, які використовують глибокі нейронні мережі з використанням великої кількості прихованих шарів. Я б хотіла розглянути комерційну програму Tensorflow та Keras, яку буде використано в реалізації моделі.

Tensorflow є відносно новою бібліотекою програмного забезпечення з відкритим кодом. Дана бібліотека використовується для визначення, навчання та розгортання глибоких нейронних мереж. Ця система програмування використовує обчислювальний граф або граф потоків даних для представлення обчислювальних задач (алгоритмів навчання). Обчислювальний граф складається з вузлів і спрямованих ребер, що з'єднують вузли. Вузли представляють операції, а ребра представляють потік даних між операціями. Існує декілька принципів компоненти обчислювального графа, а саме операції, тензори, змінні та сеанси.

1. У TensorFlow вузли представляють операції. Якщо точніше, то вузли описують, як вхідні дані проходять через них в орієнтованому графі. Операція може отримати нуль або багато входів, а потім дати нуль або багато виходів. Така операцією може бути математичне рівняння, константа або змінна. Константа отримується за допомогою операції, яка не потребує жодних вхідних даних і дає на виході таку саму константу, як і відповідна константа. Аналогічно, змінна - це операція, яка не потребує вхідних даних і видає поточне значення цієї змінної. Будь-яка операція повинна бути реалізована за допомогою ядра, яке може бути виконане на апаратному пристрої, такому як центральний процесор(CPU) або графічний процесор(GPU).
2. Тензори – це дані, що протікають між вузлами у обчислювальному графі. Тензор - це багатовимірний масив зі статичним типом і

динамічними вимірами. Кількість вимірів тензора називається його рангом. Форма тензора описує кількість компонент у кожному вимірі. В обчислювальному графі при створенні операції повертається тензор, який буде надіслано спрямованим ребром на вхід підключеної операції.

3. При виконанні стохастичного градієнтного спуску обчислювальний граф нейронної мережі виконується ітеративно для одного експерименту. В процесі навчання більшість тензорів не виживають, тоді як стан моделі такі як ваги та зсув, потрібно підтримувати. Тому змінні додаються до обчислювального графа як спеціальні операції. Змінні, що зберігають тензори постійно зберігаються в буферах оперативної пам'яті. Значення змінних можна завантажувати під час навчання та оцінювання моделі. При створенні змінної необхідно надати їй тензор як початкове значення при виконанні. Форма і тип даних цього тензора автоматично стає формою і типом змінної. Ініціалізація змінних повинна бути виконана перед навчанням. Це можна зробити додавши операцію ініціалізації всіх змінних і виконати її перед навчання мережі.
4. Виконання операцій та обчислення тензорів відбувається у контексті сеансу. Сеанс використовує процедуру `Run` як вхід для виконання обчислювального графа. При виклику `run` на вхід приймається увесь граф, який потрібно обчислити. Обчислювальний граф буде виконуватися багаторазово для навчання мережі за допомогою виклику процедури `Run`. Сеанс розподіляє операції з графа між пристроями, такими як центральний процесор(CPU) або графічний процесор(GPU) на різних машинах відповідно до алгоритму розміщення TensorFlow. Крім того, в сеансі виконується впорядкування виконання вузлів визначено явно, а саме керуючими залежностями. Оцінювання моделі гарантує, що керуючі залежності зберігаються.

Завдання виконання обчислювального графа поділяється на чотири групи: клієнт, майстер, робітники та набір пристроїв. Клієнт надсилає запит на

виконання графа за призначення завдань набору робітників. Кожен працівник відповідає за моніторинг одного або декількох пристроїв, які є фізичними об'єктами для реалізації ядра операції. Виходячи з кількості машин, існує дві "версії" TensorFlow, а саме локальна та розподілена реалізація. Локальна система полягає в тому, що вузли виконуються на одній машині; розподілена система полягає в реалізації вузлів на багатьох машинах з багатьма пристроями.

Пристрої - це базові та найменші фізичні одиниці для виконання операцій в TensorFlow. Ядро вузлів буде призначено доступним пристроям, таким як центральний процесор(CPU) або графічний(GPU) для виконання. Крім того, TensorFlow дозволяє створювати нові фізичні одиниці реалізації, зареєстровані користувачами. Для моніторингу операцій на пристрої кожен робочий процес відповідає за один або декілька пристроїв на одній машині. Ім'я пристрою визначається його типом та індексом робочої групи в якій він знаходиться.

Алгоритм розміщення полягає у визначенні того, які вузли будуть призначені до якого пристрою. Алгоритм імітує проходження графа від вхідного тензора до вихідного тензора. Алгоритм використовує модель вартості $C_v(d)$ для визначення на якому пристрої $D = \{d_1, \ldots, d_n\}$ виконати конкретну операцію. Оптимальний пристрій визначається мінімумом вартості моделі $\hat{d} = argmin_{d \in D} C_v(d)$ для розміщення заданої вершини під час навчання.

Крос-пристрійне виконання: Тензорпотік зазвичай призначає вузли на різні пристрої, якщо в системі користувача є декілька пристроїв. Цей процес класифікує вузли, а потім призначає вузли, що належать до одного класу, на один пристрій. Тому необхідно розібратися з проблемою залежності вузлів, розподілених між пристроями. Розглянемо два таких пристрої, де вузол A знаходиться на пристрої A. Якщо тензор v, вироблений операцією A, передається як вхідні дані на дві різні операції α і β, які знаходяться на

пристрої B, то існують вихідні ребра $v \to \alpha$ і $v \to \beta$ від пристрою A до пристрою B.

На практиці процес передачі вихідного тензора v від пристрою A, наприклад , графічного процесора, на пристрій B, наприклад, центрального процесора, завершується створенням двох типів вузлів, а саме send та recv. Також, TensorFlow використовує "канонізацію" для оптимізації пар (send,recv), що є еквівалентним, але більш ефективнішим підходом у порівнянні з наведеним вище прикладом. Два вузли recv які з'єднуються з α і β на пристрої B, замінюються лише одним вузлом recv, який отримує вихід з v, що передається на два залежні вузли α і β.

Для забезпечення ефективності та продуктивності моделі виконання TensorFlow, деякі оптимізацій у бібліотеці. Усунення спільних підграфів що виконується у TensorFlow, полягає у канонізації однотипних операцій над ідентичним вхідним тензором до однієї операції при обході обчислювального графа. Потім вихідний тензор передається до всіх залежних вузлів. Інша оптимізація, а саме планування, полягає у виконанні вузлів якомога пізніше що гарантує те, що результат операцій залишиться в пам'яті на необхідний мінімальний час. Це ефективно зменшує споживання пам'яті та покращує продуктивність моделі. Оптимізація стиснення втрат полягає у додаванні до графа вузлів перетворення. Оптимізована робастна модель не змінює вихідний сигнал відгуку через варіацію шумових сигналів, саме тому вимоги до точності арифметики алгоритму знижуються. Виходячи з цього принципу, коли дані передаються між пристроями або машинами, 32-бітне представлення з плаваючою комою у відправника усікається до 16-бітового представлення вузлом перетворення, а потім перетворюється в 32-бітне на приймальному кінці простим додаванням нулів.

Тепер розглянемо функцію, яка являє собою доповнення до базової моделі програмування TensorFlow. Для алгоритмів машинного навчання необхідні обчислення градієнта певних вузлів по відношенню до одного або декількох вузлів обчислювального графа. У нейронної мережі градієнт

вартості по відношенню до ваги повинен бути обчислений на основі навчального прикладу, який подається в мережу. Алгоритм зворотного розповсюдження використовується для обчислення градієнта у зворотному напрямку, починаючи з кінця графа.

Існує два методи обчислення градієнтів зі зворотним поширенням через обчислювальний граф обчислювальним графом у [13]. Перший - це диференціювання від символу до числа. Вхідні дані поширюються вперед по графу для обчислення функції вартості, а потім градієнт обчислюється за допомогою ланцюгового правила у зворотному напрямку через граф. Іншим методом, більш прийнятим у TensorFlow, є похідні від символу до символу, який обчислює градієнт автоматично замість того, щоб явно застосовувати алгоритм зворотного розповсюдження. Таким чином, до обчислювального графа додаються спеціальні вузли для обчислення градієнта кожної операції, включеної в ланцюжок. Для реалізації алгоритму зворотного розповсюдження, виконання градієнтних вузлів відбувається так само, як і інших вузлів, шляхом запуску механізму оцінювання графа. Цей метод надає символьний дескриптор для обчислення похідних замість обчислення похідних як числових значень.

Градієнт певної вершини v по відношенню до іншого тензора α обчислюється у зворотному напрямку від v до α через граф. Це розширює граф шляхом додавання вершини градієнта до кожної операції o, яка є функцією від α, що зустрічається в ланцюжку $(v \circ - - - \circ o \circ - - -)(\alpha)$, що утворює вихідний тензор [15]. Отже TensorFlow додає вершину градієнта для кожної такої операції o шляхом множення похідної його зовнішньої функції на власну похідну. Процедура виконується обернено обчислюється до кінцевої вершини, яка створює символічну ручку до шуканого градієнта $\frac{dv}{d\alpha}$, який неявно виконує метод зворотного розповсюдження.

У [14] диференціювання від символу до символу може призвести до значних обчислювальних витрат, а також збільшення накладних витрат пам'яті. Причиною цього є те, що виходить два різних рівняння для

застосування ланцюгового правила. Перше рівняння повторно використовує попередні обчислення, що вимагає більше часу для зберігання, ніж потрібно для подальшого розповсюдження. Альтернативний спосіб вираження ланцюгового правила показує, що кожна функція повинна перерахувати всі свої аргументи і викликає кожну внутрішню функцію, від якої вона залежить. Наразі TensorFlow застосовує перший підхід, який заснований на статті [14]. Розглядаючи ланцюжок, який має тисячі вузлів, то переобчислювати саму внутрішню функцію майже для кожної операції у ланці здається нерозумним. З іншого боку, не оптимально зберігати тензори довго на пристрої, особливо на такому, як графічний процесор(GPU), ресурс пам'яті якого є дефіцитним. Теоретично тензор, що зберігається на пристрої, звільняється як тільки він буде оброблений графічними залежностями, саме тому переобчислення деяких тензорів замість того, щоб зберігати їх на пристрої, може бути покращено в майбутній роботі.

TensorFlow підтримує мови програмування C++ та Python, які дозволяють користувачам викликати функціональність бекенду. Наразі інтерфейс програмування TensorFlow на мові Python є більш зручним у використанні, оскільки він надає різноманітні функції для спрощення і завершення побудови та виконання обчислювального графа, які ще не підтримуються мовою C++.

Програма TensorFlow складається з двох окремих фаз: фази побудови та фази виконання. У фазі побудови створюється обчислювальний граф, який відображає структуру та алгоритми навчання нейронної мережі. Потім, на етапі виконання, операції на графі виконуються багаторазово, щоб навчити мережу. Для глибокої нейронної мережі з великою кількістю прихованих шарів етапи створення ваги і зсуву, обчислення множення і додавання матриць, застосування нелінійної функції активації не є ефективними. Тому пропонується ряд бібліотек з відкритим вихідним кодом для абстрагування та пакування цих кроків, а також для побудови високорівневих блоків, таких як цілі шари за один раз. Однією з таких бібліотек є keras. Це бібліотека з

відкритим вихідним кодом, яка надає інтерфейс Python для штучних нейронних мереж.

Keras охоплює кожен етап робочого процесу машинного навчання: від обробки даних до налаштування гіперпараметрів і розгортання. Він був розроблений з нахилом на можливість швидкого експериментування.

Keras надає повний доступ до масштабованості та кросплатформних можливостей TensorFlow. Можна запускати Keras на модулі тензорного процесора(TPU) або великих кластерах графічних процесорів(GPU), а також експортувати моделі Keras для запуску в браузері або на мобільних пристроях[14].

3.2 Реалізація Scikit-learn

В даному розділі я б хотіла описати експеримент з виявлення шахрайських операцій з кредитними картками, а також показати результати. Я використала набір даних про операції з кредитними картками, які були здійснені європейськими власниками кредитних карток за два дні у вересні 2013 року. Він містить 284 807 транзакцій, з яких 492 є шахрайськими, що становить 0,172% від усіх транзакцій.

Набір даних містить в собі тридцять ознак, які є числовими значеннями. Вони вже перетворені за допомогою аналізу головних компонент (PCA) для зменшення розмірності простору ознак. Тут оригінальні характеристики споживачів не вказані. Ознаки V1, V2, ... V28 - це головні компоненти, які ми отримані за допомогою методу головних компонент(PCA). Також є дві ознаки "Час" та "Кількість", які не трансформовані як головні компоненти. Ознака "Час" – це час між кожною транзакцією та першою транзакцією в наборі даних. Елемент "Сума" - це сума транзакції. Мітка "Клас" - це цільова мітка зі значенням 1, що представляє випадок шахрайства, а 0 - для нормального випадку.

Стандартизація простору ознак є загальною вимогою для ефективного навчання мережі, оскільки нейронна мережа чутлива до того, як вхідні

вектори масштабуються. У багатьох наборах даних ознаки мають різний масштаб і діапазон значень, що призводить до збільшення часу навчання збіжності мережі. Саме тому корисно стандартизувати всі ознаки таким чином, щоб кожна ознака була приведена до стандартного нормального розподілу шляхом вилучення середнього значення кожної ознаки, а потім масштабування шляхом ділення непостійних знаків на їхні стандартні відхилення.

У наборі даних ми розглядаємо шахрайські транзакції як позитивний клас, а нормальні транзакції як негативний клас. Існує чотири прогнозовані результати, які можна сформулювати в матриці плутанини 2 × 2. Позначаємо їх як TN - це кількість правильної класифікації звичайних операцій (True Negative), FP – кількість нормальних операцій, помилково класифікованих як шахрайські (False Positive), FN - кількість шахрайських операцій, помилково класифікованих як нормальні (False Negative) і TP - кількість правильно класифікованих шахрайських операцій (True Positive).

Ефективність алгоритму навчання нейронної мережі зазвичай оцінюється за допомогою точності прогнозування[16], яка визначається як точність, тобто Accuracy=(T P+T N)/(T P+ F P + T N + F N). На жаль, така проста точність прогнозування не є завжди потрібним показником, оскільки розподіл нормальних і шахрайських транзакцій у наборі даних є надзвичайно незбалансованим. Проста стратегія за замовчуванням, яка класифікує транзакцію як нормальну, дає дуже високу точність. Незважаючи на високу точність, модель не здатна виявити шахрайські транзакції серед усіх транзакцій. Тут же існує мета навчання мережі з отриманням високого показника правильного виявлення шахрайських транзакцій, що дозволяє знизити рівень помилок у нормальних транзакціях., щоб досягти цього [16]. Тому проста точність прогнозування очевидно, не є ефективною метрикою в цьому випадку. Крива операційної характеристики приймача (ROC) є стандартною методикою для оцінки продуктивності моделі для бінарної класифікації.

ROC-крива описує продуктивність моделі, яка представлена співвідношенням між частотою істинних спрацьовувань і частотою хибних спрацьовувань в діапазоні порогів прийняття рішень. На ROC-просторі вісь X визначається як %FP =FP/(TN+FP) (чутливість), а вісь Y визначається як %TP =TP/(TP+FN)(1- специфічність). Випадкове передбачення дає точку вздовж діагональної лінії $y = x$. Чим ближче ROC-крива до лівого верхнього кута, тим вищу точність має модель. Тому за допомогою ROC-кривих можна легко представити порівняння результатів роботи різних моделей. Площа під ROC-кривою (AUC) є ефективною метрикою для узагальнення ефективності ROC-кривої. Як правило, ROC-крива з більшою AUC свідчить про кращу ефективність модель.

На основі матриці плутанини виводяться інші показники оцінки наведені. Тут представлена одна конкретна метрика оцінки, яка називається впізнаваністю і визначається як %TP =TP/(TP+FN). Впізнаваність - це відсоток істинно позитивних класів, які були правильно визначені моделлю серед загальної кількості позитивних класів. Воно використовується для порівняння різних моделей при оцінці ефективності виявлення позитивних класів.

Логістичну регресію обрано як еталонну модель, яку можна вважати як найпростішу нейронну мережу, якщо використовується сигмоїдна функція активації. Далі будуть використовуватися нейронні мережі, які мають складнішу структуру, щоб дослідити, чи можна отримати кращу продуктивність, ніж при використанні моделі логістичної регресії. Експерименти реалізовані в Scikit-learn та TensorFlow.

Логістична регресія - це метод регресії для прогнозування бінарної залежної змінної, яка приймає значення 0 або 1.

У логістичній регресії складність моделі вже є низькою, що робить її менш перенавчанною. Програмна реалізація Scikit-learn показано у додатку А.

Scikitlearn - комерційній бібліотеці машинного навчання з відкритим вихідним кодом. Набір даних розбито на навчальну та тестову частини. Навчання мережі виконується на навчальній вибірці, а продуктивність мережі

оцінюється на тестовій вибірці. Через незбалансованість даних навчання мережі проходить без повторної вибірки та з передискретизацією на навчальній вибірці, щоб дослідити вплив передискретизації на продуктивність мережі.

Проектування топології нейронної мережі є критичним фактором, що впливає на точність системи класифікації. Додавання прихованих вузлів може підвищити точність мережі, однак, надмірна кількість прихованих вузлів може призвести до проблеми перенавчання, що негативно впливає на узагальнення і призводить до відхилення в прогнозі, саме тому для підвищення точності та узагальнення вимагає відповідної кількості прихованих вузлів . Формальної теорії для визначення кількості прихованих вузлів не існує. Рекомендації базуються на попередньому досвіді та повторних експериментах.

Стверджується, що мережа з найкращою продуктивністю - це мережа з однаковою кількістю вузлів у кожному прихованому шарі[17].Тестування проходили з різною кількістю вузлів у прихованих шарах. Тому можна зробити висновок , що оптимально приймати однакову кількість вузлів у прихованих шарах. Починаючи з невеликої мережі з одним прихованим шаром, а потім розширюючи шар мережі до 5 прихованих шарів. Виявилося, що мережа, яка має 4 прихованих шарів з 20 вузлами в кожному прихованому шарі дала кращий результат. Ця мережа було навчено зі швидкістю навчання 0.001 за 400 ітерацій та регуляризацією L2 параметром L2 регуляризації 0.1.

Результат прогнозування на тестовому наборі без передискретизації показано нижче:

```
[[55688  1173]
 [    8    93]]

Accuracy: 97.93%
Recall: 92.08%
```

Щоб збалансувати дані, на навчальній вибірці було здійснено передискретизацію, щоб збільшити клас меншості, щоб зрівняти його з класом

більшості. Одним із підходів є метод синтетичної передискретизації меншості, який з вибірки класу меншості переобирає шляхом створення "синтетичних" прикладів. Для кожного прикладу шахрайства x, синтетичні приклади вводяться вздовж ліній, що з'єднують з k найближчих сусідніх зразків шахрайства. Відповідно до незбалансованого співвідношення визначається кількість необхідних надлишкових вибірок N. Таким чином, лише N сусідів з k найближчих випадковим чином вибираються, і вибірки генеруються напрямку кожна з них.

Результат прогнозування з передискретизацією наведено нижче:

```
Training set score: 0.969356
```

Як видно з наведених вище результатів двох мереж, ми виявили, що результати з використанням передискретизації на навчальній вибірці є кращими, ніж результати без використання методу передискретизації. Звідси випливає висновок, що передискретизація на незбалансованій навчальній вибірці може покращити загальну продуктивність мережі для правильного виявлення шахрайських транзакцій. Порівнюючи результати роботи двох нейронних мереж з використанням методу передискретизації, можна сказати, що друга мережа отримала вищий відгук на тестовому наборі, в той час як кількість нормальних транзакцій, класифікованих як шахрайські, є більшою.

Оскільки витрати на обробку шахрайських транзакцій є значними, ситуація з класифікацією нормальних транзакцій як шахрайських є досить складною. Це може призвести до непотрібних витрат. Тому ми обираємо першу нейронну мережу з чотирма прихованими шарами для проведення наступних експериментів.

Можна зробити висновок, що у Scikit-learn при навчанні не покращуються більше, ніж протягом двох послідовних епох, мережа вважається збіжною і навчання припиняється. Більша швидкість навчання збільшує швидкість збіжності мережі, але водночас призводить до нестабільності мережі. Малі швидкості навчання призводить до повільної

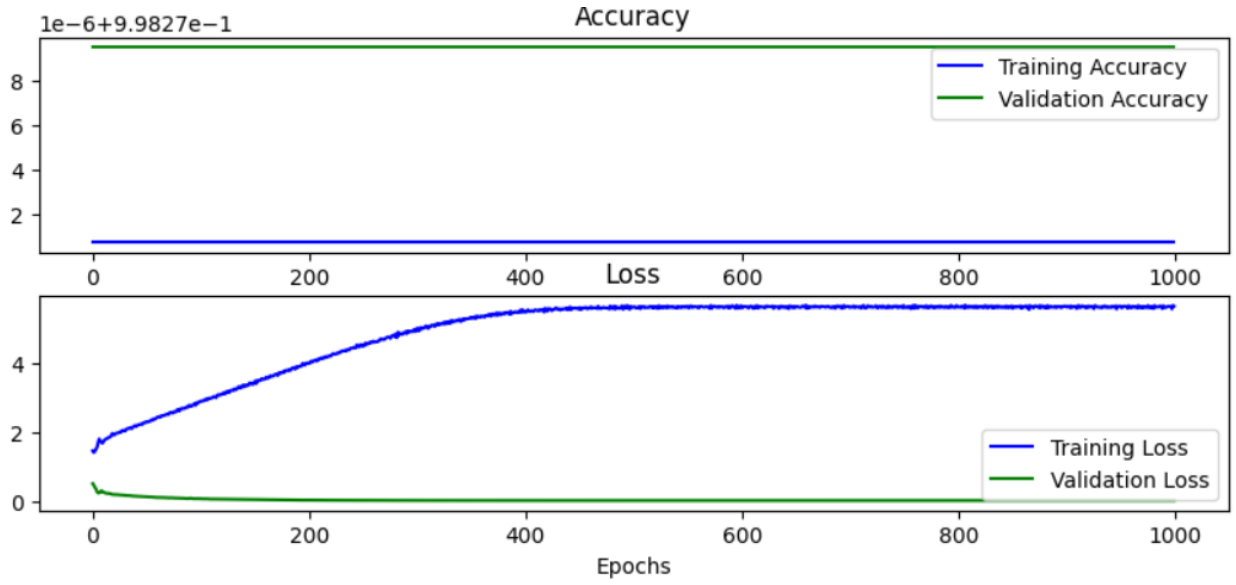
збіжності, що робить шлях збіжності більш плавним. Крім того, втрати є меншими при використанні малих швидкостей навчання.

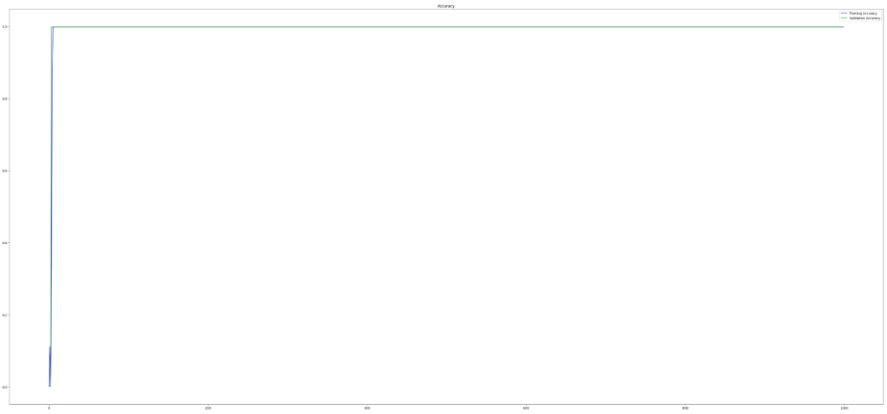
3.3 Реалізація TensorFlow

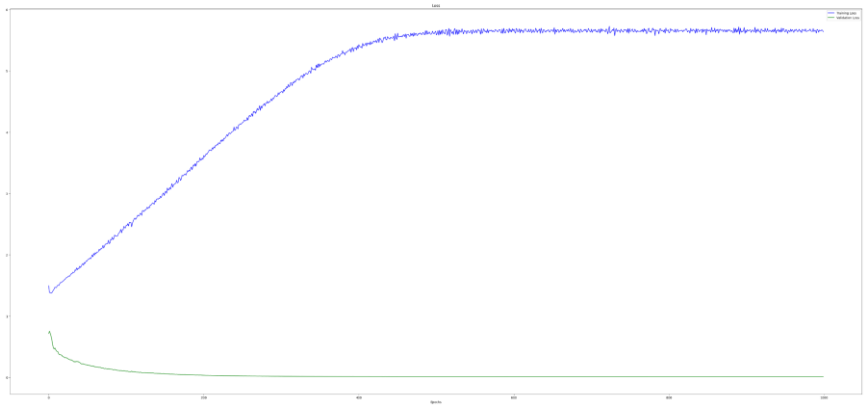
Програмна реалізація TensorFlow та Keras показана в додатку Б

Набір даних розділено на навчальний, валідаційний та тестовий. Навчальний набір збалансовано шляхом дублювання шахрайських транзакцій, щоб кількість двох класів була рівною. Структура мережі така ж, як і в реалізації Scikit-learn яка має 4 приховані шари з 20 вузлами в кожному шарі. Обираємо параметри навчання тестуючи точність валідації.

```
▾                                MLPClassifier
MLPClassifier(alpha=1, hidden_layer_sizes=(20, 20, 20, 20),
              learning_rate_init=0.05, max_iter=400, random_state=1,
              verbose=10)
```







```
Test accuracy: 99.82795715332031 %
```

Рисунок 3.1 – Результати навчання MPLClassifier

Аналіз результатів показує, що мережа сходиться при епосі=200 та швидкості навчання=0.005, щодо до точності валідації.

Порівняно з реалізацією Scikit-learn, обидві реалізації отримали відгук у тому ж діапазоні.

ВИСНОВКИ

Аналіз алгоритмів машинного навчання для прогнозування фроду при онлайн покупках є важливим завданням в сфері кібербезпеки. Вирішення цього завдання допомагає забезпечити безпеку та захист фінансових операцій в електронній комерції.

В даній роботі було проведено аналіз алгоритмів нейронних мереж, розглянуті такі розділи як штучний нейрон, функція активації, одношарова мережа прямого зв’язку, персептрон та глибока нейронна мережа.

На базі отриманих знань була побудована математична модель глибокої нейронної мережі, було представлено програмне забезпечення для машинного навчання та механізм його роботи. При застосуванні нейронної мережі для проведення початкових експериментів при використанні реального набору даних з прикладами шахрайських дій з кредитними картками було порівняно різні моделі.

Були протестовані нейронні мережі з різною кількістю шарів і отримано висновок, що збільшена кількість прихованих шарів не означає кращу ефективність. А мережі, яка містить в собі лише один прихований шар часто показує такий же результат, як і мережа, яка і мережа, в якій міститься декілька.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] S. S. Haykin, Neural networks and learning machines. Upper Saddle River, NJ: Pearson Education, third ed., 2009.

[2] H. Paugam-Moisy and S. Bohte, “Computing with spiking neuron networks,” in Handbook of natural computing, pp. 335–376, Springer, 2012.

[3] J. Stiles and T. L. Jernigan, “The basics of brain development,” Neuropsychology review, vol. 20, no. 4, pp. 327–348, 2010.

[4] A. J. Izenman, Modern multivariate statistical techniques, vol. 1. Springer, 2008.

[5] A. Dongare, R. Kharde, and A. D. Kachare, “Introduction to artificial neural network,”

[6] C. M. Bishop, Neural networks for pattern recognition. Oxford university press, 1995.

[7] J. Jiang, P. Trundle, and J. Ren, “Medical image analysis with artificial neural networks,” Computerized Medical Imaging and Graphics, vol. 34, no. 8, pp. 617–631, 2010.

[8] Y. Chalich and J. Li, “The essential role of nonlinearity in neural networks,” 2017.

[9] D. R. Musicant, J. M. Christensen, and J. F. Olson, “Supervised learning by training on aggregate outputs,” in Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on, pp. 252–261, IEEE, 2007.

[10] T. M. Mitchell, Machine Learning. New York, NY, USA: McGraw-Hill, Inc., 1 ed., 1997.

[11] E. Phaisangittisagul, “An analysis of the regularization between l2 and dropout in single hidden layer neural network,” in Intelligent Systems, Modelling and Simulation (ISMS), 2016 7th International Conference on, pp. 174–179, IEEE, 2016.

[12] S. J. Nowlan and G. E. Hinton, “Simplifying neural networks by soft weightsharing,” Neural computation, vol. 4, no. 4, pp. 473–493, 1992.

[13] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man´e, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vi´egas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2015.

[15] Yifei Lu, "Deep neural networks and fraud detection" , 2017.

[16] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," CoRR, vol. abs/1106.1813, 2011.

[17] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," Journal of Machine Learning Research, vol. 10, no. Jan, pp. 1–40, 2009.

ДОДАТОК А

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn import preprocessing

data = pd.read_csv('./creditcard_B.csv')
columns = data.columns

features_columns = columns.delete(len(columns) - 1)
features = data[features_columns]
labels = data['Class']

X_scaled = preprocessing.scale(features)

features_train, features_test, labels_train, labels_test =
train_test_split(X_scaled, labels, test_size=0.2,
random_state=0)

oversampler = SMOTE(random_state=0)
os_features, os_labels =
oversampler.fit_resample(features_train, labels_train)
```

```
clf = MLPClassifier(solver='adam', alpha=1,
hidden_layer_sizes=(20, 20, 20, 20),
                    learning_rate_init=0.05, verbose=10,
max_iter=400, random_state=1)

clf.fit(os_features, os_labels)

print("Training set score: %f" % clf.score(os_features,
os_labels))

y_pred = clf.predict(features_test)

conf_matrix = confusion_matrix(labels_test, y_pred)
print(conf_matrix)

accuracy = np.round(100 * float((conf_matrix[0][0] +
conf_matrix[1][1])) / float((conf_matrix[0][0] +
conf_matrix[1][1] + conf_matrix[1][0] + conf_matrix[0][1])),
2)
recall = np.round(100 * float((conf_matrix[1][1])) /
float((conf_matrix[1][0] + conf_matrix[1][1])), 2)

print('Accuracy: ' + str(accuracy) + '%')
print('Recall: ' + str(recall) + '%')
```

ДОДАТОК Б

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense

data = pd.read_csv("./creditcard_B.csv")
data['Normal'] = np.where(data['Class'] == 0, 1, 0)
Fraud = data[data.Class == 1]
Normal = data[data.Normal == 1]
X_train = Fraud.sample(frac=0.8, random_state=1)
countFrauds = len(X_train)
X_train = pd.concat([X_train, Normal.sample(frac=0.8,
random_state=1)], axis=0)
X_test = data.loc[~data.index.isin(X_train.index)]
X_train = shuffle(X_train, random_state=1)
X_test = shuffle(X_test, random_state=1)
y_train = X_train[['Class', 'Normal']]
y_test = X_test[['Class', 'Normal']]
X_train = X_train.drop(['Class', 'Normal'], axis=1)
X_test = X_test.drop(['Class', 'Normal'], axis=1)
ratio = len(X_train) / countFrauds
y_train['Class'] *= ratio
```

```
.
features = X_train.columns.values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
split = int(len(y_test) / 2)
inputX = X_train
inputY = y_train.values
inputX_valid = X_test[:split]
inputY_valid = y_test.values[:split]
inputX_test = X_test[split:]
inputY_test = y_test.values[split:]
n_input = X_train.shape[1]
n_hidden1 = 20
n_hidden2 = 20
n_hidden3 = 20
n_hidden4 = 20

model = Sequential([
    Dense(n_hidden1, activation='sigmoid',
input_shape=(n_input,)),
    Dense(n_hidden2, activation='sigmoid'),
    Dense(n_hidden3, activation='sigmoid'),
    Dense(n_hidden4, activation='sigmoid'),
    Dense(2, activation='softmax')
])
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(inputX, inputY, epochs=1000,
batch_size=2048, validation_data=(inputX_valid,
inputY_valid), verbose=0)

plt.figure(figsize=(10, 4))
plt.subplot(2, 1, 1)
plt.plot(history.history['accuracy'], label='Training
Accuracy', color='blue')
plt.plot(history.history['val_accuracy'], label='Validation
Accuracy', color='green')
plt.title('Accuracy')
plt.legend()
plt.subplot(2, 1, 2)
plt.plot(history.history['loss'], label='Training Loss',
color='blue')
plt.plot(history.history['val_loss'], label='Validation
Loss', color='green')
plt.title('Loss')
plt.legend()
plt.xlabel('Epochs')
plt.show()
test_loss, test_accuracy = model.evaluate(inputX_test,
inputY_test, verbose=0)
print('Test accuracy:', test_accuracy * 100, '%')
```